# QorIQ LS1046A Security (SEC) Reference Manual

Document Number: LS1046ASECRM
Rev. 0, 05/2017

# Contents

**Section number** **Title** **Page**

## Chapter 1
## Overview of SEC (security engine) functionality

## Chapter 2
## Feature summary

## Chapter 3
## SEC implementation

3.1 SEC submodules........................................75

3.2 Cryptographic engines implemented in SEC........................................75

## Chapter 4
## SEC modes of operation

4.1 Security Monitor (SecMon) security states........................................77

4.1.1 The effect of security state on volatile keys........................................78

4.1.2 The effect of security state on non-volatile keys........................................79

4.2 Keys available in different security modes........................................79

4.2.1 Keys available in trusted mode........................................79

4.2.2 Keys available in secure mode........................................80

4.2.3 Keys available in non-secure mode........................................80

4.2.4 Keys available in fail mode........................................81

## Chapter 5
## SEC hardware functional description

5.1 System Bus Interfaces........................................84

5.1.1 AXI master (DMA) interface........................................84

5.1.1.1 DMA read-safe transactions........................................84

5.1.1.2 DMA interface write-safe transactions........................................85

5.1.1.3 DMA write-efficient transactions........................................85

5.1.1.4 DMA bursts that may read past the end of data structures........................................86

5.1.2 Register interface (IP bus)........................................87

5.2 SEC service interface concepts........................................88

Section number Title Page

5.2.1 SEC descriptors..........88

5.2.2 Job termination status/error codes..........90

5.2.3 Frames and flows..........95

5.2.4 Frame descriptors and frames..........96

5.2.5 Frame descriptor flow and flow context..........96

5.2.6 Buffer allocation, release, and reuse..........97

5.2.7 User data access control and isolation..........97

5.3 Service interfaces..........98

5.3.1 Job Ring interface..........99

5.3.1.1 Configuring and managing the input/output rings, overview..........99

5.3.1.2 Managing the input rings..........101

5.3.1.3 Managing the output rings..........102

5.3.1.4 Controlling access to job rings..........103

5.3.1.5 Order of job completion..........103

5.3.1.6 Initializing job rings..........104

5.3.1.7 Job Ring Registers..........104

5.3.1.8 Asserting job ring interrupts..........104

5.3.2 Queue Manager Interface (QI)..........105

5.3.2.1 Requesting and receiving frame descriptors from QMan..........106

5.3.2.2 Building job descriptors for QI jobs..........107

5.3.2.3 Controlling QI access to frame queues and data..........107

5.3.2.4 Tracking the completion order of QI jobs..........108

5.3.2.5 Initializing the Queue Manager Interface..........108

5.3.2.6 Done/error notification for QI jobs..........109

5.3.3 Register-based service interface..........109

5.4 Job scheduling..........111

5.4.1 Job scheduling - default algorithm..........111

5.4.2 Job scheduling - DECO-specific jobs..........114

5.5 Job execution hardware..........115

5.5.1 Descriptor controller (DECO) and cryptographic control block (CCB)....115
5.5.1.1 Alignment blocks....116
5.5.2 Cryptographic hardware accelerators (CHAs) (overview)....117

## Chapter 6
## Frame queues, frame descriptors, and buffers

6.1 Frame queues....119
6.1.1 Dequeue response....120
6.1.1.1 Context_A field (preheader)....121
6.2 Frame descriptors....123
6.2.1 Processing simple frame jobs....124
6.2.2 Processing compound frame jobs....125
6.2.3 Frame descriptor error handling....126
6.2.4 Job descriptor construction from frame descriptor....127

## Chapter 7
## Descriptors and descriptor commands

7.1 Job descriptors....131
7.2 Trusted descriptors....133
7.3 Shared descriptors....135
7.3.1 Executing shared descriptors in proper order....136
7.3.2 Specifying different types of shared descriptor sharing....138
7.3.2.1 Error sharing....139
7.3.3 Changing shared descriptors....139
7.4 Using in-line descriptors....140
7.5 Using replacement job descriptors....141
7.6 Scatter/gather tables (SGTs)....143
7.7 Using descriptor commands....144
7.7.1 Command execution order....145
7.7.1.1 Executing commands when SHR = 0....147
7.7.1.2 Executing commands when SHR = 1....148

| Section number | Title | Page |
|---|---|---|

7.7.1.3 Executing commands when REO = 0 .......... 148
7.7.1.4 Executing commands when REO = 1 .......... 149
7.7.1.5 Executing additional HEADER commands .......... 150
7.7.1.6 Jumping to another job descriptor .......... 151
7.7.2 Command properties .......... 152
7.7.2.1 Blocking commands .......... 152
7.7.2.2 Load/store checkpoint .......... 152
7.7.2.3 Done checkpoint .......... 152
7.7.3 Command types .......... 153
7.7.4 SEQ vs non-SEQ commands .......... 154
7.7.4.1 Creating a sequence .......... 155
7.7.4.2 Using sequences for fixed and variable length data .......... 156
7.7.4.3 Transferring meta data .......... 157
7.7.4.4 Rewinding a sequence .......... 158
7.7.5 Information FIFO entries .......... 158
7.7.6 Output FIFO Operation .......... 159
7.7.7 Output Checksum logic .......... 161
7.7.8 Cryptographic class .......... 162
7.7.9 Address pointers .......... 163
7.8 HEADER command .......... 164
7.9 KEY commands .......... 169
7.10 LOAD commands .......... 173
7.11 FIFO LOAD command .......... 186
7.11.1 Bit length data .......... 189
7.11.2 FIFO LOAD input data type .......... 190
7.12 ECPARAM command .......... 192
7.13 STORE command .......... 196
7.14 FIFO STORE command .......... 204
7.15 MOVE, MOVEB, MOVEDW, and MOVE_LEN commands .......... 210

Section number Title Page

7.16 ALGORITHM OPERATION command...219

7.17 PROTOCOL OPERATION commands...226

7.18 PKHA OPERATION command...245

7.18.1 PKHA OPERATION: clear memory function...247

7.18.2 PKHA OPERATION: Arithmetic Functions...248

7.18.3 PKHA OPERATION: copy memory functions...255

7.18.4 PKHA OPERATION: Elliptic Curve Functions...257

7.19 SIGNATURE command...260

7.20 JUMP (HALT) command...263

7.20.1 Jump type...264

7.20.1.1 Local conditional jump...264

7.20.1.2 Local conditional increment/decrement jump...265

7.20.1.3 Non-local conditional jump...265

7.20.1.4 Conditional halt...266

7.20.1.5 Conditional halt with user-specified status...266

7.20.1.6 Conditional subroutine call...267

7.20.1.7 Conditional subroutine return...267

7.20.2 Test type...268

7.20.3 JSL and TEST CONDITION fields...269

7.20.4 JUMP command format...270

7.21 MATH and MATHI Commands...273

7.22 SEQ IN PTR command...280

7.23 SEQ OUT PTR command...283

## Chapter 8
## Public Key Cryptography Operations

8.1 Conformance considerations...287

8.2 Discrete-log key-pair generation...288

8.2.1 Inputs to the discrete-log key-pair generation function...289

8.2.2 Assumptions of the discrete-log key-pair generation function...289

8.2.3 Outputs from the discrete-log key-pair generation function....289

8.2.4 Operation of the discrete-log key-pair generation function....289

8.2.5 Notes associated with the discrete-log key-pair generation function ....290

8.3 Using the Diffie_Hellman function.... 294

8.3.1 Diffie_Hellman requirements.... 294

8.3.2 Inputs to the Diffie-Hellman function....294

8.3.3 Assumptions of the Diffie-Hellman function....295

8.3.4 Outputs from the Diffie-Hellman function.... 295

8.3.5 Operation of the Diffie-Hellman function.... 295

8.3.6 Notes associated with the Diffie-Hellman function....295

8.4 Generating DSA and ECDSA signatures....296

8.4.1 Inputs to the DSA and ECDSA signature generation function....297

8.4.2 Assumptions of the DSA and ECDSA signature generation function....297

8.4.3 Outputs from the DSA and ECDSA signature generation function....297

8.4.4 Operation of the DSA and ECDSA signature generation function ....298

8.4.5 Notes associated with the DSA and ECDSA Signature Generation function....298

8.5 Verifying DSA and ECDSA signatures....301

8.5.1 Inputs to the DSA and ECDSA signature verification function.... 302

8.5.2 Assumptions of the DSA and ECDSA signature verification function.... 302

8.5.3 Outputs from the DSA and ECDSA signature verification function....302

8.5.4 Operation of the DSA and ECDSA signature verification function .... 302

8.5.5 Notes associated with the DSA and ECDSA Signature Verification function ....303

8.6 RSA Finalize Key Generation (RFKG)....306

8.7 Implementation of the RSA encrypt operation....307

8.8 Implementation of the RSA decrypt operation....309

## Chapter 9
## Protocol acceleration

9.1 IPsec ESP encapsulation and decapsulation overview.... 317

9.1.1 IPsec ESP encapsulation and decapsulation mode support.... 319

**Section number** **Title** **Page**

9.1.2 IPsec ESP error codes.......... 319

9.1.3 Programming for IPsec .......... 320

9.1.3.1 PDB format for IPsec ESP Transport (and Legacy Tunnel) encapsulation.......... 321

9.1.3.2 Common PDB format descriptions for IPsec ESP Transport (and Legacy Tunnel) decapsulation.......... 324

9.1.3.3 Overriding ESP Transport (and legacy Tunnel) PDB content with the DECO Protocol Override Register.......... 327

9.1.3.4 PDB format for IPsec ESP Tunnel encapsulation .......... 328

9.1.3.5 Common PDB format descriptions for IPsec ESP Tunnel decapsulation.......... 331

9.1.3.6 Overriding ESP Tunnel PDB content with the DECO Protocol Override Register.......... 334

9.1.3.7 IPsec ESP encapsulation CBC-specific PDB segment format descriptions.......... 336

9.1.3.8 IPsec ESP encapsulation AES-CTR-specific PDB segment format descriptions.......... 336

9.1.3.9 IPsec ESP encapsulation AES-CCM-specific PDB segment format descriptions.......... 337

9.1.3.10 IPsec ESP encapsulation AES-GCM-specific PDB segment format descriptions.......... 337

9.1.3.11 IPsec ESP decapsulation CBC-specific PDB segment format descriptions.......... 338

9.1.3.12 IPsec ESP decapsulation AES-CTR-specific PDB segment format descriptions.......... 338

9.1.3.13 IPsec ESP decapsulation AES-CCM-specific PDB segment format descriptions.......... 339

9.1.3.14 IPsec ESP decapsulation AES-GCM-specific PDB segment format descriptions.......... 339

9.1.4 IPsec ESP Transport (and Legacy Tunnel) encapsulation overview.......... 340

9.1.4.1 Encapsulating the IP header in tunnel mode.......... 341

9.1.4.2 Encapsulating the IP header in transport mode.......... 341

9.1.4.3 Process for IPsec ESP Transport (and Legacy Tunnel) encapsulation.......... 341

9.1.5 IPsec ESP Cryptographic Encapsulation.......... 343

9.1.5.1 Process for IPsec encapsulation when using AES-CBC or DES-CBC.......... 343

9.1.5.2 Process for IPsec encapsulation when using AES-CTR.......... 344

9.1.5.3 Process for IPsec encapsulation when using AES-CCM.......... 346

9.1.5.4 Process for IPsec encapsulation when using AES-GCM.......... 348

9.1.6 IPsec ESP Transport (and Legacy Tunnel) decapsulation procedure overview.......... 349

9.1.6.1 IPsec ESP Transport Mode outer IP header decapsulation procedure.......... 351

9.1.6.2 IPsec ESP Transport (and Legacy Tunnel) outer IP header decapsulation procedure (tunnel mode)........351

9.1.7 IPsec ESP Cryptographic Decapsulation........352

9.1.7.1 IPsec decapsulation procedure when using AES-CBC or DES-CBC........352

9.1.7.2 Process for IPsec decapsulation when using AES-CTR........354

9.1.7.3 Process for IPsec decapsulation when using AES-CCM........355

9.1.7.4 Process for IPsec decapsulation when using AES-GCM........356

9.1.7.5 Use of SPI and the sequence number in decapsulation........357

9.1.7.6 Optional use of ESN in ESP decapsulation........358

9.1.7.7 Anti-replay checking in IPsec ESP decapsulation........358

9.1.7.7.1 When anti-replay checking is enabled........359

9.1.7.7.2 When anti-replay checking is disabled........359

9.1.7.8 ICV checking during IPsec ESP decapsulation........360

9.1.8 IPsec ESP Tunnel encapsulation overview........360

9.1.8.1 Handling the Outer IP Header during ESP Tunnel encapsulation........360

9.1.8.2 Outer IP Header handling with UDP-encapsulated-ESP........362

9.1.8.3 ESP Tunnel Outer IP Header manipulation........362

9.1.8.4 ESP Tunnel handling of Next Header........363

9.1.9 IPsec ESP tunnel decapsulation overview........363

9.1.9.1 Input material preceding the outer IP header........364

9.1.9.2 Handling the Outer IP Header during ESP Tunnel decapsulation........364

9.1.9.3 Manipulation of the Inner IP Header during ESP Tunnel decapsulation........365

9.1.9.4 Decapsulation Output Frame Length........366

9.2 SSL/TLS/DTLS record encapsulation and decapsulation overview........366

9.2.1 Programming and processing details common to all versions of SSL, TLS, and DTLS........367

9.2.1.1 PDB use and format for SSL, TLS, and DTLS encapsulation and decapsulation........368

9.2.1.1.1 PDB for SSL, TLS, and DTLS when a Block Cipher is used........368

9.2.1.1.2 PDB for SSL, TLS, and DTLS when AES-Counter mode is used........369

9.2.1.1.3 PDB for TLS and DTLS when AES-GCM is used........370

9.2.1.1.4 PDB for TLS and DTLS when AES-CCM is used....371

9.2.1.1.5 Programming the Options byte with the PDB for SSL, TLS and DTLS....372

9.2.1.2 Overriding the PDB for SSL, TLS, and DTLS Encapsulation.... 374

9.2.1.3 Computing the pre-encrypted record length during decapsulation....375

9.2.1.4 SSL, TLS, DTLS Decapsulation Output frame options....376

9.2.1.5 SSL / TLS / DTLS error codes....378

9.2.2 Process for SSL 3.0 and TLS 1.0 record encapsulation....378

9.2.2.1 Differences between SSL 3.0 and TLS 1.0 (record encapsulation).... 379

9.2.2.2 Processing SSL 3.0 and TLS 1.0 record encapsulation with block ciphers....380

9.2.3 Process for SSL 3.0 and TLS 1.0 record decapsulation....381

9.2.3.1 SSL 3.0 and TLS 1.0 Record Decapsulation for block ciphers.... 382

9.2.3.2 Differences between SSL 3.0 and TLS 1.0 (record decapsulation).... 382

9.2.4 Process for TLS 1.1 and TLS 1.2 record encapsulation....383

9.2.4.1 Differences between TLS 1.0, TLS 1.1, and TLS 1.2 Record Encapsulation.... 384

9.2.4.2 Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation....384

9.2.4.3 Processing TLS 1.1 and TLS 1.2 record encapsulation with block ciphers (AES or DES).. 386

9.2.4.4 Processing TLS 1.1 and TLS 1.2 record encapsulation with stream ciphers....387

9.2.4.5 Processing TLS 1.1 and TLS 1.2 record encapsulation with AEAD ciphers.... 388

9.2.5 Process for TLS 1.1 and TLS 1.2 record decapsulation....389

9.2.5.1 Decapsulation of TLS 1.1 and TLS 1.2 records when a stream cipher is used....390

9.2.5.2 Decapsulation of TLS 1.1 and TLS 1.2 records when a block cipher is used.... 392

9.2.5.3 Decapsulation of TLS 1.2 records when an AEAD is used.... 393

9.2.6 Process for DTLS record encapsulation....394

9.2.6.1 Differences between DTLS and TLS....395

9.2.6.2 Process of DTLS Record Encapsulation when using a Block Cipher.... 395

9.2.6.3 Process of DTLS Record Encapsulation when using a Stream Cipher....397

9.2.6.4 DTLS 1.2 Record Encapsulation when using an AEAD Cipher.... 398

9.2.7 Process for DTLS record decapsulation....399

9.2.7.1 Differences between DTLS and TLS....400

Section number Title Page

9.2.7.2 Process of DTLS Record Decapsulation when using a Block Cipher.................................. 400

9.2.7.3 Process of DTLS Record Decapsulation when using a Stream Cipher................................ 402

9.2.7.4 DTLS 1.2 Record Decapsulation when using an AEAD Cipher......................................... 403

9.3 SRTP packet encapsulation and decapsulation.........................................................................................405

9.3.1 Building the initial counter value (Counter IV)........................................................................ 406

9.3.2 Building the AEAD Nonce...................................................................................................... 406

9.3.3 Constructing the AESA context from the SRTP AEAD Nonce for AES-CCM mode.............................407

9.3.4 SRTP encapsulation.................................................................................................................. 408

9.3.4.1 Process for SRTP encapsulation........................................................................................ 409

9.3.4.2 Handling the optional MKI.................................................................................................410

9.3.4.3 SRTP encapsulation PDB format descriptions................................................................... 410

9.3.4.4 SRTP encapsulation error conditions.................................................................................411

9.3.5 SRTP decapsulation overview.................................................................................................. 411

9.3.5.1 Process for SRTP decapsulation........................................................................................ 412

9.3.5.2 SRTP decapsulation PDB format descriptions................................................................... 413

9.3.5.3 SRTP decapsulation error conditions.................................................................................415

9.4 IEEE 802.1AE MACsec encapsulation and decapsulation overview.......................................................415

9.4.1 Process for 802.1AE MACsec encapsulation.......................................................................... 416

9.4.1.1 Using the frame check sequence (FCS)............................................................................. 418

9.4.1.2 Additional notes for GMAC support................................................................................. 419

9.4.2 MACsec encapsulation PDB format descriptions....................................................................420

9.4.3 Process for 802.1AE MACSec decapsulation.........................................................................421

9.4.3.1 Automatically switching between two keys.......................................................................423

9.4.3.2 Additional notes for GMAC support (decapsulation).........................................................424

9.4.4 MACsec decapsulation PDB format descriptions....................................................................424

9.5 IEEE 802.11-2012 WPA2 MPDU encapsulation and decapsulation ..................................................... 425

9.5.1 Processing Common to WPA2 Encapsulation and Decapsulation.......................................... 426

9.5.1.1 Constructing the AAD for WPA2 encapsulation and decapsulation................................... 426

9.5.1.2 Constructing the CCMP Nonce for WPA2 encapsulation and decapsulation...................... 427

9.5.1.3 Constructing the AESA context for WPA2 CCMP encapsulation and decapsulation.......... 427

9.5.2 Process for WPA2 encapsulation.......... 428

9.5.2.1 Constructing the CCMP header for WPA2 encapsulation.......... 429

9.5.2.2 WPA2 Payload Encapsulation.......... 430

9.5.2.3 Computing the FCS for WPA2 encapsulation.......... 430

9.5.2.4 WPA2 encapsulation PDB format descriptions.......... 430

9.5.2.5 WPA2 encapsulation error conditions.......... 432

9.5.3 Process for WPA2 decapsulation.......... 432

9.5.3.1 WPA2 Decapsulation Anti-replay checking.......... 433

9.5.3.2 Using automatic key-switching.......... 434

9.5.3.3 WPA2 decapsulation PDB format descriptions.......... 434

9.5.3.4 WPA2 decapsulation error conditions.......... 435

9.6 IEEE 802.16 WiMAX encapsulation and decapsulation overview.......... 436

9.6.1 Process for IEEE 802.16 WiMAX encapsulation.......... 437

9.6.2 IEEE 802.16 WiMAX encapsulation PDB format descriptions.......... 439

9.6.3 WiMax encapsulation error conditions.......... 440

9.6.4 Procedure for IEEE 802.16 WiMAX decapsulation.......... 440

9.6.4.1 Transforming the GMH (WiMAX decapsulation).......... 442

9.6.4.2 Automatic key switching (WiMAX decapsulation).......... 443

9.6.5 IEEE 802.16 WiMAX decapsulation PDB format descriptions.......... 443

9.6.6 WiMAX decapsulation error conditions.......... 444

9.7 Anti-Replay built-in checking.......... 444

9.8 Process for 3G double-CRC encapsulation and decapsulation.......... 447

9.8.1 3G double-CRC encapsulation process.......... 448

9.8.1.1 Calculating the 7-bit CRC of the PDU header for encapsulation.......... 448

9.8.1.2 Calculating the 11-bit CRC of the PDU header for encapsulation.......... 449

9.8.1.3 Calculating the 16-bit payload CRC for encapsulation.......... 450

9.8.2 3G double-CRC encapsulation PDB format descriptions.......... 450

9.8.3 3G double-CRC decapsulation process.......... 450

Section number Title Page

9.8.3.1 Calculating the 7-bit CRC of the PDU header for decapsulation........................................ 451

9.8.3.2 Calculating the 11-bit CRC of the PDU header for decapsulation...................................... 451

9.8.3.3 Calculating the 16-bit payload CRC for decapsulation....................................................... 452

9.8.4 3G double-CRC decapsulation PDB format descriptions........................................................452

9.9 3G RLC PDU Encapsulation and Decapsulation overview......................................................................453

9.9.1 3G RLC PDU encapsulation overview................................................................................... 453

9.9.2 Process for 3G RLC PDU encapsulation................................................................................ 454

9.9.3 3G RLC PDU encapsulation PDB format descriptions.......................................................... 455

9.9.4 3G RLC PDU decapsulation overview................................................................................... 456

9.9.5 Process for 3G RLC PDU decapsulation................................................................................ 457

9.9.6 3G RLC PDU decapsulation PDB format descriptions.......................................................... 458

9.9.7 Overriding the PDB for 3G RLC PDU encapsulation and decapsulation............................... 458

9.10 LTE PDCP PDU encapsulation and decapsulation overview...................................................................459

9.10.1 LTE PDCP PDU IV generation.............................................................................................. 460

9.10.2 LTE PDCP PDU encapsulation process for confidentiality only........................................... 463

9.10.3 LTE PDCP PDU encapsulation for confidentiality and integrity........................................... 465

9.10.4 LTE PDCP PDU decapsulation process for confidentiality only........................................... 466

9.10.5 LTE PDCP PDU decapsulation for confidentiality and integrity........................................... 467

9.10.6 LTE PDCP shared descriptor PDB format descriptions......................................................... 469

9.10.7 Overriding the PDB for LTE PDCP encapsulation and decapsulation....................................469

## Chapter 10
## Key agreement functions

10.1 IKEv2 PRF overview..........................................................................................................................471

10.1.1 Using IKE PRF to generate SKEYSEED............................................................................... 472

10.1.2 Using IKE PRF+ to generate keying material for the IKEv2 SA........................................... 472

10.1.3 Using IKE PRF+ to generate Child SA key material..............................................................473

10.1.4 Restrictions on programming control blocks......................................................................... 473

10.1.5 IKE PRF PDB format descriptions......................................................................................... 474

10.1.6 Implementation details for IKE PRF function........................................................................ 477

10.1.7 Implementation Details for IKE PRF+ function....478

10.2 SSL/TLS/DTLS pseudo-random functions (PRF)....479

10.2.1 SSL 3.0 PRF overview....479

10.2.1.1 SSL 3.0 PRF definitions....480

10.2.2 Process for SSL 3.0 PRF....481

10.2.3 SSL 3.0 PRF PDB format descriptions....481

10.2.4 TLS 1.0/TLS 1.1/DTLS PRF overview....484

10.2.4.1 TLS PRF RFC definitions....486

10.2.5 Process for TLS 1.0, TLS 1.1, DTLS PRF....487

10.2.5.1 How TLS uses PRF material....487

10.2.5.2 Concatenating input material into one input string (TLS 1.0/1.1/DTLS)....488

10.2.6 TLS 1.0, TLS 1.1, DTLS PRF PDB format descriptions....489

10.2.7 TLS 1.2 PRF overview....492

10.2.8 Process for TLS 1.2 PRF....493

10.2.8.1 Concantenating input material into one input string (TLS 1.2)....494

10.2.8.2 How TLS uses PRF material (TLS 1.2)....494

10.2.9 TLS 1.2 PRF PDB format descriptions....495

10.3 Implementation of the derived key protocol....497

10.3.1 Using DKP with HMAC keys....498

10.3.2 Implementation of the Blob Protocol....499

## Chapter 11
## Cryptographic hardware accelerators (CHAs)

11.1 Public-key hardware accelerator (PKHA) functionality....502

11.1.1 Modular math....503

11.1.2 About Montgomery values....503

11.1.3 Non-modular Math....505

11.1.4 Elliptic-Curve Math....505

11.1.4.1 Point math over a prime field (Fp)....506

11.1.4.2 Point math over a binary field (F2m)....507

11.1.4.3 About Jacobian projective coordinates.......... 508

11.1.4.4 About the Point at Infinity.......... 509

11.1.5 PKHA Mode Register.......... 509

11.1.6 PKHA functions.......... 509

11.1.6.1 Clear Memory (CLEAR_MEMORY) function.......... 510

11.1.6.2 Integer Modular Addition (MOD_ADD) function.......... 512

11.1.6.3 Integer Modular Subtraction (MOD_SUB_1) function.......... 512

11.1.6.4 Integer Modular Subtraction (MOD_SUB_2) function.......... 513

11.1.6.5 Integer Modular Multiplication (MOD_MUL).......... 513

11.1.6.6 Integer Modular Multiplication with Montgomery Inputs (MOD_MUL_IM).......... 514

11.1.6.7 Integer Modular Multiplication with Montgomery Inputs and Outputs (MOD_MUL_IM_OM) Function.......... 515

11.1.6.8 Integer Modular Exponentiation (MOD_EXP and MOD_EXP_TEQ).......... 515

11.1.6.9 Integer Modular Exponentiation, Montgomery Input (MOD_EXP_IM and MOD_EXP_IM_TEQ) Function.......... 516

11.1.6.10 Integer Simultaneous Modular Exponentiation (MOD_SML_EXP).......... 517

11.1.6.11 Integer Modular Square (MOD_SQR and MOD_SQR_TEQ).......... 518

11.1.6.12 Integer Modular Square, Montgomery inputs (MOD_SQR_IM and MOD_SQR_IM_TEQ).......... 518

11.1.6.13 Integer Modular Square, Montgomery inputs and outputs (MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ).......... 519

11.1.6.14 Integer Modular Cube (MOD_CUBE and MOD_CUBE_TEQ).......... 520

11.1.6.15 Integer Modular Cube, Montgomery input (MOD_CUBE_IM and MOD_CUBE_IM_TEQ).......... 520

11.1.6.16 Integer Modular Cube, Montgomery input and output (MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ).......... 521

11.1.6.17 Integer Modular Square Root (MOD_SQRT).......... 522

11.1.6.18 Integer Modulo Reduction (MOD_AMODN).......... 523

11.1.6.19 Integer Modular Inversion (MOD_INV).......... 523

11.1.6.20 Integer Montgomery Factor Computation (MOD_R2).......... 524

Section number Title Page

11.1.6.21 Integer Greatest Common Divisor (MOD_GCD)....................................................................524

11.1.6.22 Miller_Rabin Primality Test (PRIME_TEST)........................................................................525

11.1.6.23 Binary Polynomial (F2m) Addition (F2M_ADD) function...................................................525

11.1.6.24 Binary Polynomial (F2m) Modular Multiplication (F2M_MUL)....................................... 526

11.1.6.25 Binary Polynomial (F2m) Modular Multiplication with Montgomery Inputs
(F2M_MUL_IM) Function............................................................................................ 527

11.1.6.26 Binary Polynomial (F2m) Modular Multiplication with Montgomery Inputs and Outputs
(F2M_MUL_IM_OM) Function....................................................................................528

11.1.6.27 Binary Polynomial (F2m) Modular Exponentiation (F2M_EXP and F2M_EXP_TEQ)...... 528

11.1.6.28 Binary Polynomial (F2m) Simultaneous Modular Exponentiation (F2M_SML_EXP)........ 529

11.1.6.29 Binary Polynomial (F2m) Modular Square (F2M_SQR and F2M_SQR_TEQ).................. 530

11.1.6.30 Binary Polynomial (F2m) Modular Square, Montgomery Input (F2M_SQR_IM and
F2M_SQR_IM_TEQ)..................................................................................................... 531

11.1.6.31 Binary Polynomial (F2m) Modular Square, Montgomery Input and Output
(F2M_SQR_IM_OM and F2M_SQR_IM_OM_TEQ).........................................................531

11.1.6.32 Binary Polynomial (F2m) Modular Cube (F2M_CUBE and F2M_CUBE_TEQ)...............532

11.1.6.33 Binary Polynomial (F2m) Modular Cube, Montgomery Input (F2M_CUBE_IM and
F2M_CUBE_IM_TEQ)................................................................................................... 533

11.1.6.34 Binary Polynomial (F2m) Modular Cube, Montgomery Input and Output
(F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_TEQ)..................................................534

11.1.6.35 Binary Polynomial (F2m) Modulo Reduction (F2M_AMODN)..........................................534

11.1.6.36 Binary Polynomial (F2m) Modular Inversion (F2M_INV).................................................535

11.1.6.37 Binary Polynomial (F2m) R2 Mod N (F2M_R2) Function.................................................536

11.1.6.38 Binary Polynomial (F2m) Greatest Common Divisor (F2M_GCD) Function..................... 536

11.1.6.39 ECC Fp Point Add, Affine Coordinates (ECC_MOD_ADD) Function..............................537

11.1.6.40 ECC Fp Point Add, Affine Coordinates, R2 Mod N Input (ECC_MOD_ADD_R2)
Function.......................................................................................................................... 537

11.1.6.41 ECC Fp Point Double, Affine Coordinates (ECC_MOD_DBL) Function...........................538

11.1.6.42 ECC Fp Point Multiply, Affine Coordinates (ECC_MOD_MUL and
ECC_MOD_MUL_TEQ) Function.................................................................................. 539

11.1.6.43 ECC Fp Point Multiply, R2 Mod N Input, Affine Coordinates (ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ) Function....540

11.1.6.44 ECC Fp Check Point (ECC_MOD_CHECK_POINT) Function....541

11.1.6.45 ECC Fp Check Point, R2 Mod N Input, Affine Coordinates (ECC_MOD_CHECK_POINT_R2) Function....542

11.1.6.46 ECC F2m Point Add, Affine Coordinates (ECC_F2M_ADD) Function.... 543

11.1.6.47 ECC F2m Point Add, Affine Coordinates, R2 Mod N Input (ECC_F2M_ADD_R2) Function.... 544

11.1.6.48 ECC F2m Point Double - Affine Coordinates (ECC_F2M_DBL) Function....545

11.1.6.49 ECC F2m Point Multiply, Affine Coordinates (ECC_F2M_MUL and ECC_F2M_MUL_TEQ) Function....546

11.1.6.50 ECC F2m Point Multiply, R2 Mod N Input, Affine Coordinates (ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ) Function.... 547

11.1.6.51 ECC F2m Check Point (ECC_F2M_CHECK_POINT) Function.... 548

11.1.6.52 ECC F2m Check Point, R2 (ECC_F2M_CHECK_POINT_R2) Function....549

11.1.6.53 ECM Modular Multiplication (ECM_MOD_MUL and ECM_MOD_MUL_TEQ) Function.... 550

11.1.6.54 ECM Fp Point Multiply, R2 Mod N Input, Affine Coordinates (ECM_MOD_MUL_R2 and ECM_MOD_MUL_R2_TEQ) Function.... 551

11.1.6.55 ECT Modular Multiplication (ECT_MOD_MUL and ECT_MOD_MUL_TEQ) Function. 552

11.1.6.56 ECT Fp Point Multiply, R2 Mod N Input, Affine Coordinates (ECT_MOD_MUL_R2 and ECT_MOD_MUL_R2_TEQ) Function....553

11.1.6.57 ECT Fp Point Add, Affine Coordinates (ECT_MOD_ADD) Function.... 555

11.1.6.58 ECT Fp Point Add, Affine Coordinates, R2 Mod N Input (ECT_MOD_ADD_R2) Function.... 555

11.1.6.59 ECT Fp Check Point (ECT_MOD_CHECK_POINT) Function.... 556

11.1.6.60 ECT Fp Check Point, R2 (ECT_MOD_CHECK_POINT_R2) Function....557

11.1.6.61 Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ).... 558

11.1.6.62 Right Shift A (R_SHIFT) function.... 559

11.1.6.63 Compare A B (COMPARE) function.... 559

11.1.6.64 Evaluate A (EVALUATE) function.... 560

11.1.7 Special values for common ECC domains......561

11.2 Kasumi f8 and f9 hardware accelerator(KFHA) functionality...... 580

11.2.1 KFHA use of the Mode Register......580

11.2.2 KFHA use of the Context Register...... 581

11.2.3 KFHA use of the Key Register...... 582

11.2.4 KFHA use of the Data Size Register...... 582

11.2.5 KFHA error conditions...... 582

11.3 Data encryption standard accelerator (DES) functionality...... 583

11.3.1 DESA use of the Mode Register...... 583

11.3.2 DESA use of the Key Register......584

11.3.3 DESA use of the Key Size Register......584

11.3.4 DESA use of the Data Size Register......584

11.3.5 DESA Context Register...... 585

11.3.6 Save and store operations in DESA context data......585

11.4 Cyclic-redundancy check accelerator (CRCA) functionality...... 585

11.4.1 CRCA modes of operation......586

11.4.2 CRCA use of the Mode Register......586

11.4.3 CRCA Key Register......588

11.4.4 CRCA Key Size Register......588

11.4.5 CRCA Data Size Register......588

11.4.6 CRCA Context Register......588

11.4.7 Save and restore operations in CRCA context data...... 589

11.5 Random-number generator (RNG) functionality......589

11.5.1 RNG features summary......589

11.5.2 RNG functional description ...... 590

11.5.2.1 RNG state handles......590

11.5.2.2 RNG NIST certification......590

11.5.3 RNG operations......592

11.5.4 RNG use of the Key Registers...... 593

11.5.5 RNG use of the Context Register....593

11.5.6 RNG use of the Data Size Register....594

11.6 SNOW 3G f8 accelerator functionality....594

11.6.1 Differences between SNOW 3G f8 and SNOW 3G f9....594

11.6.2 SNOW 3G f8 use of the Mode Register.... 595

11.6.3 SNOW 3G f8 use of the Context Register.... 596

11.6.4 SNOW 3G f8 use of the Data Size Register.... 596

11.6.5 SNOW 3G f8 use of the Key Register.... 597

11.6.6 SNOW 3G f8 use of the Key Size Register.... 597

11.7 SNOW 3G f9 accelerator functionality....597

11.7.1 SNOW 3G f9 use of the Mode Register.... 598

11.7.2 SNOW 3G f9 use of the Context Register.... 599

11.7.3 SNOW 3G f9 use of the Data Size Register.... 600

11.7.4 SNOW 3G f9 use of the Key Register.... 600

11.7.5 SNOW 3G f9 use of the Key Size Register.... 601

11.7.6 SNOW 3G f9 use of ICV check....601

11.8 Message digest hardware accelerator (MDHA) functionality.... 601

11.8.1 MDHA use of the Mode Register.... 602

11.8.2 MDHA use of the Key Register....603

11.8.2.1 Using the MDHA Key Register with normal keys.... 603

11.8.2.2 Using the MDHA Key Register with IPAD/OPAD "split keys".... 603

11.8.2.2.1 Definition and function of IPAD/OPAD split keys.... 604

11.8.2.2.2 Process flow of using the Key Register with split keys....604

11.8.2.2.3 Using padding with the split key type to align with storage....604

11.8.2.2.4 Length of a split key.... 604

11.8.2.2.5 Loading/storing a split key with a key command.... 604

11.8.2.2.6 Loading/storing a split key with a FIFO STORE command....605

11.8.2.2.7 Sizes of split keys....605

11.8.2.2.8 Constructing an HMAC-SHA-1 split key in memory.... 605

11.8.2.3 MDHA use of the Key Size Register....606

11.8.3 MDHA use of the Data Size Register....606

11.8.4 MDHA use of the Context Register....607

11.8.5 Save and restore operations in MDHA context data....607

11.9 AES accelerator (AESA) functionality....607

11.9.1 Differences between the AES encrypt and decrypt keys....607

11.9.2 AESA as both Class 1 and Class 2 CHA....608

11.9.3 AESA modes of operation....609

11.9.4 AESA use of registers....610

11.9.5 AESA use of the parity bit....611

11.9.6 AES ECB mode....611

11.9.6.1 AES ECB mode use of the Mode Register....611

11.9.6.2 AES ECB mode use of the Context Register....612

11.9.6.3 AES ECB Mode use of the Data Size Register ....612

11.9.6.4 AES ECB Mode use of the Key Register....612

11.9.6.5 AES ECB Mode use of the Key Size Register....612

11.9.7 AES CBC, OFB, CFB128 modes....613

11.9.7.1 AES CBC, OFB, and CFB128 modes use of the Mode Register....613

11.9.7.2 AES CBC, OFB, and CFB128 modes use of the Context Register....614

11.9.7.3 AES CBC, OFB, and CFB128 modes use of the Data Size Register....614

11.9.7.4 AES CBC, OFB, and CFB128 modes use of the Key Register....615

11.9.7.5 AES CBC, OFB, and CFB128 modes use of the Key Size Register....615

11.9.8 AES CTR mode....615

11.9.8.1 AES CTR mode use of the Mode Register....615

11.9.8.2 AES CTR mode use of the Context Register....616

11.9.8.3 AES CTR mode use of the Data Size Register....616

11.9.8.4 AES CTR mode use of the Key Register....616

11.9.8.5 AES CTR mode use of the Key Size Register....617

11.9.9 AES XTS mode....617

Section number Title Page

11.9.9.1 AES XTS mode use of the Mode Register.......... 617

11.9.9.2 AES XTS mode use of the Context Register.......... 617

11.9.9.3 AES XTS mode use of the Data Size Register.......... 618

11.9.9.4 AES XTS mode use of the Key Register.......... 618

11.9.9.5 AES XTS mode use of the Key Size Register.......... 619

11.9.10 AES XCBC-MAC and CMAC modes.......... 619

11.9.10.1 AES XCBC-MAC and CMAC modes use of the Mode Register.......... 619

11.9.10.2 AES XCBC-MAC and CMAC Modes use of the Context Register.......... 621

11.9.10.3 AES XCBC-MAC and CMAC modes use of the Class 1 ICV Size Register.......... 621

11.9.10.4 AES XCBC-MAC and CMAC modes use of the Data Size Register.......... 622

11.9.10.5 AES XCBC-MAC and CMAC modes use of the Key Register.......... 622

11.9.10.6 AES XCBC-MAC and CMAC modes use of the Key Size Register.......... 622

11.9.10.7 ICV checking in AES XCBC-MAC and CMAC modes.......... 622

11.9.11 AESA CCM mode.......... 623

11.9.11.1 Generation encryption.......... 623

11.9.11.2 Decryption verification.......... 623

11.9.11.3 AES CCM mode use of the Mode Register.......... 624

11.9.11.4 AES CCM mode use of the Context Register.......... 625

11.9.11.5 AES CCM mode use of the Data Size Register.......... 626

11.9.11.6 AES CCM mode use of the Key Register.......... 626

11.9.11.7 AES CCM mode use of the Key Size Register.......... 626

11.9.11.8 AES CCM mode use of the ICV check.......... 627

11.9.12 AES GCM mode.......... 627

11.9.12.1 GMAC.......... 628

11.9.12.2 GCM data types.......... 628

11.9.12.3 IV processing.......... 628

11.9.12.4 GCM initialization.......... 628

11.9.12.5 AES GCM mode use of the Mode Register.......... 628

11.9.12.6 AES GCM mode use of the Context Register.......... 630

11.9.12.7 AES GCM Mode use of the Data Size Register.................................................................. 630

11.9.12.8 AES GCM mode use of the Class 1 IV Size Register......................................................... 631

11.9.12.9 AES GCM mode use of the AAD Size Register...................................................................631

11.9.12.10 AES GCM mode use of the Class 1 ICV Size Register.......................................................631

11.9.12.11 AES GCM mode use of the Key Register............................................................................631

11.9.12.12 AES GCM mode use of the Key Size Register....................................................................631

11.9.12.13 AES GCM mode use of the ICV check.............................................................................. 632

11.9.13 AESA optimization modes.........................................................................................................632

11.9.13.1 CTR-XCBC and CTR-CMAC modes data format.............................................................. 632

11.9.13.2 CTR-XCBC and CTR-CMAC modes message format...................................................... 632

11.9.13.3 CTR-CMAC-LTE for LTE PDCP control-plane processing...............................................633

11.9.13.4 Authentication-only data.......................................................................................................633

11.9.13.5 AES optimization modes use of the Mode Register........................................................... 633

11.9.13.6 AES optimization modes use of the Context Register........................................................635

11.9.13.7 AES optimization modes use of the Data Size Register..................................................... 638

11.9.13.8 AES optimization modes use of the AAD Size Register..................................................... 639

11.9.13.9 AES optimization modes use of the Class 1 ICV Size Register.......................................... 639

11.9.13.10 AES optimization modes use of the Class 1 Key Register................................................. 640

11.9.13.11 AES optimization modes use of the Class 2 Key Register................................................. 640

11.9.13.12 AES optimization modes use of the Class 1 Key Size Register.......................................... 640

11.9.13.13 AES optimization modes use of the Class 2 Key Size Register.......................................... 641

11.9.13.14 AES optimization modes use of the ICV check..................................................................641

11.9.13.15 AES optimization modes error conditions.......................................................................... 641

11.10 ZUC encryption accelerator (ZUCE) functionality.................................................................................. 643

11.10.1 Differences between ZUCE and ZUCA.....................................................................................643

11.10.2 ZUCE use of the Mode Register................................................................................................ 644

11.10.3 ZUCE use of the Context Register.............................................................................................644

11.10.4 ZUCE use of the Data Size Register..........................................................................................645

11.10.5 ZUCE use of the Key Register...................................................................................................645

11.10.6 ZUCE use of the Key Size Register ........ 646

11.11 ZUC authentication accelerator (ZUCA) functionality........ 646

11.11.1 ZUCA use of the Mode Register........ 646

11.11.2 ZUCA use of the Context Register........ 647

11.11.3 ZUCA use of the Data Size Register........ 649

11.11.4 ZUCA use of the Key Register........ 649

11.11.5 ZUCA use of the Key Size Register........ 649

11.11.6 ZUCA use of ICV checking........ 649

## Chapter 12
## Trust Architecture modules

12.1 Run-time integrity checker (RTIC)........ 651

12.1.1 RTIC modes of operation........ 651

12.1.2 RTIC initialization and operation........ 651

12.1.3 RTIC use of the Throttle Register........ 652

12.1.4 RTIC use of command, configuration, and status registers........ 652

12.1.5 Initializing RTIC........ 653

12.1.6 RTIC Memory Block Address/Length Registers........ 653

12.2 SEC virtualization and security domain identifiers (SDIDs)........ 654

12.2.1 Virtualization........ 654

12.2.2 Security domain identifiers (SDIDs)........ 654

12.2.3 TrustZone SecureWorld........ 655

12.3 Special-purpose cryptographic keys........ 655

12.3.1 Initializing and clearing black and trusted descriptor keys........ 656

12.3.2 Black keys and JDKEK/TDKEK........ 656

12.3.3 Trusted descriptors and TDSK........ 656

12.3.4 Master key and blobs........ 656

12.4 Black keys........ 657

12.4.1 Black key encapsulation schemes........ 657

12.4.2 Differences between black and red keys........ 657

12.4.3 Loading red keys....658

12.4.4 Loading black keys....658

12.4.5 Avoiding errors when loading red and black keys....658

12.4.6 Encapsulating and decapsulating black keys....659

12.4.7 Types of black keys and their use....660

12.4.8 Types of blobs for key storage....661

12.5 Trusted descriptors....662

12.5.1 Why trusted descriptors are needed....662

12.5.2 Trusted-descriptor key types and uses....662

12.5.3 Trusted descriptors encrypting/decrypting black keys....663

12.5.4 Trusted-descriptor blob types and uses....663

12.5.5 Configuring the system to create trusted descriptors properly....663

12.5.6 Creating trusted descriptors....664

12.5.6.1 Trusted descriptors and descriptor-header bits....664

12.5.6.2 Trusted-descriptor execution considerations....665

12.6 Blobs....666

12.6.1 Blob protocol....666

12.6.2 Why blobs are needed....667

12.6.3 Blob conformance considerations....667

12.6.4 Encapsulating and decapsulating blobs....669

12.6.5 Blob types....669

12.6.5.1 Blob types differentiated by format....670

12.6.5.2 Blob types differentiated by content....671

12.6.5.2.1 Red blobs (for general data)....671

12.6.5.2.2 Black blobs (for cryptographic keys)....671

12.6.5.2.3 Enforcing blob content type....672

12.6.5.3 Blob types differentiated by security state....672

12.6.6 Blob encapsulation....673

12.6.7 Blob decapsulation....674

Section number | Title | Page

12.7 Critical security parameters.......675

12.8 Manufacturing-protection chip-authentication process.......675

12.8.1 Providing data to the manufacturing-protection authentication process.......677

12.8.1.1 Providing data to the MPPrivK-generation function.......677

12.8.1.2 Providing data to the MPPubK-generation function.......677

12.8.1.3 Providing data to the MPSign function.......678

12.8.1.4 Role of the ROM-resident secure boot firmware.......678

12.8.2 MPPrivK-generation function.......679

12.8.2.1 Differences between the MPPrivK-generation function and the DL KEY PAIR GEN function.......679

12.8.2.2 MPPrivK-generation function parameters and operation.......679

12.8.2.3 Protocol data block (PDB) for the MPPrivK-generation function.......680

12.8.3 MPPubK-generation function.......681

12.8.3.1 Differences between the MPPubK-generation function and the DL KEY PAIR GEN function.......681

12.8.3.2 MPPubK-generation function parameters and operation.......682

12.8.3.3 Protocol data block (PDB) for the MPPubK-generation function.......682

12.8.3.4 Running the MPPubK generation function at the OEM's facility.......683

12.8.4 MPSign function.......683

12.8.4.1 MPSign function parameters and operation.......684

12.8.4.2 Protocol data block (PDB) MPSign function.......685

## Chapter 13
## SEC register descriptions

13.1 SEC Memory map.......689

13.2 Master Configuration Register (MCFGR).......737

13.2.1 Offset.......737

13.2.2 Function.......737

13.2.3 Diagram.......737

13.2.4 Fields.......738

13.3 Security Configuration Register (SCFGR).......741

13.3.1 Offset......741

13.3.2 Function...... 742

13.3.3 Diagram......743

13.3.4 Fields......744

13.4 Job Ring a ICID Register - most significant half (JR0ICID_MS - JR3ICID_MS)...... 745

13.4.1 Offset......745

13.4.2 Function...... 745

13.4.3 Diagram......746

13.4.4 Fields......746

13.5 Job Ring a ICID Register - least significant half (JR0ICID_LS - JR3ICID_LS)......747

13.5.1 Offset......747

13.5.2 Function...... 747

13.5.3 Diagram......748

13.5.4 Fields......748

13.6 Queue Manager Interface SDID Register (QISDID)......749

13.6.1 Offset......749

13.6.2 Function...... 749

13.6.3 Diagram......749

13.6.4 Fields......750

13.7 Debug Control Register (DEBUGCTL)...... 750

13.7.1 Offset......750

13.7.2 Function...... 750

13.7.3 Diagram......751

13.7.4 Fields......751

13.8 Job Ring Start Register (JRSTARTR)......751

13.8.1 Offset......752

13.8.2 Function...... 752

13.8.3 Diagram......752

13.8.4 Fields......752

Section number Title Page

13.9 RTIC ICID Register for Block a - most significant half (RTICAICID_MS - RTICDICID_MS)..............................753

13.9.1 Offset.....754

13.9.2 Function..... 754

13.9.3 Diagram.....754

13.9.4 Fields..... 754

13.10 RTIC ICID Register for Block a - least significant half (RTICAICID_LS - RTICDICID_LS)................................ 755

13.10.1 Offset.....755

13.10.2 Function..... 755

13.10.3 Diagram.....755

13.10.4 Fields.....756

13.11 DECO Request Source Register (DECORSR)..... 756

13.11.1 Offset.....756

13.11.2 Function..... 756

13.11.3 Diagram.....757

13.11.4 Fields.....757

13.12 DECO Request Register (DECORR).....757

13.12.1 Offset.....758

13.12.2 Function..... 758

13.12.3 Diagram.....758

13.12.4 Fields.....759

13.13 DECOa ICID Register - most significant half (DECO0ICID_MS - DECO2ICID_MS)......................................... 759

13.13.1 Offset.....760

13.13.2 Function..... 760

13.13.3 Diagram.....760

13.13.4 Fields.....760

13.14 DECOa ICID Register - least significant half (DECO0ICID_LS - DECO2ICID_LS)............................................ 761

13.14.1 Offset.....761

13.14.2 Function..... 761

13.14.3 Diagram.....761

13.14.4 Fields....762

13.15 DECO Availability Register (DAR)....762

13.15.1 Offset....762

13.15.2 Function....763

13.15.3 Diagram....763

13.15.4 Fields....763

13.16 DECO Reset Register (DRR)....764

13.16.1 Offset....764

13.16.2 Function....764

13.16.3 Diagram....764

13.16.4 Fields....764

13.17 DMA Control Register (DMAC - DMA_CTRL)....765

13.17.1 Offset....765

13.17.2 Function....765

13.17.3 Diagram....765

13.17.4 Fields....766

13.18 Peak Bandwidth Smoothing Limit Register (PBSL)....766

13.18.1 Offset....766

13.18.2 Function....766

13.18.3 Diagram....767

13.18.4 Fields....767

13.19 DMA0_AIDL_MAP_MS (DMA0_AIDL_MAP_MS)....767

13.19.1 Offset....767

13.19.2 Function....767

13.19.3 Diagram....768

13.19.4 Fields....769

13.20 DMA0_AIDL_MAP_LS (DMA0_AIDL_MAP_LS)....769

13.20.1 Offset....769

13.20.2 Function....769

13.20.3 Diagram......770

13.20.4 Fields......770

13.21 DMA0_AIDM_MAP_MS (DMA0_AIDM_MAP_MS)......770

13.21.1 Offset......771

13.21.2 Function......771

13.21.3 Diagram......771

13.21.4 Fields......771

13.22 DMA0_AIDM_MAP_LS (DMA0_AIDM_MAP_LS)......772

13.22.1 Offset......772

13.22.2 Function......772

13.22.3 Diagram......772

13.22.4 Fields......773

13.23 DMA0 AXI ID Enable Register (DMA0_AID_ENB)......773

13.23.1 Offset......773

13.23.2 Function......773

13.23.3 Diagram......774

13.23.4 Fields......774

13.24 DMA0 AXI Read Timing Check Register (DMA0_ARD_TC)......775

13.24.1 Offset......775

13.24.2 Function......776

13.24.3 Diagram......776

13.24.4 Fields......777

13.25 DMA0 Read Timing Check Latency Register (DMA0_ARD_LAT)......778

13.25.1 Offset......778

13.25.2 Function......778

13.25.3 Diagram......779

13.25.4 Fields......779

13.26 DMA0 AXI Write Timing Check Register (DMA0_AWR_TC)......779

13.26.1 Offset......779

13.26.2 Function........................................................................................................................ 780

13.26.3 Diagram........................................................................................................................780

13.26.4 Fields........................................................................................................................ 781

13.27 DMA0 Write Timing Check Latency Register (DMA0_AWR_LAT)........................................................................782

13.27.1 Offset........................................................................................................................782

13.27.2 Function........................................................................................................................ 782

13.27.3 Diagram........................................................................................................................783

13.27.4 Fields........................................................................................................................ 783

13.28 Manufacturing Protection Private Key Register (MPPKR0 - MPPKR63)........................................................................783

13.28.1 Offset........................................................................................................................783

13.28.2 Function........................................................................................................................ 783

13.28.3 Diagram........................................................................................................................784

13.28.4 Fields........................................................................................................................ 784

13.29 Manufacturing Protection Message Register (MPMR0 - MPMR31)........................................................................ 784

13.29.1 Offset........................................................................................................................784

13.29.2 Function........................................................................................................................ 785

13.29.3 Diagram........................................................................................................................785

13.29.4 Fields........................................................................................................................ 785

13.30 Manufacturing Protection Test Register (MPTESTR0 - MPTESTR31)........................................................................ 785

13.30.1 Offset........................................................................................................................785

13.30.2 Function........................................................................................................................ 786

13.30.3 Diagram........................................................................................................................786

13.30.4 Fields........................................................................................................................ 786

13.31 Job Descriptor Key Encryption Key Register (JDKEKR0 - JDKEKR7)........................................................................786

13.31.1 Offset........................................................................................................................786

13.31.2 Function........................................................................................................................ 787

13.31.3 Diagram........................................................................................................................787

13.31.4 Fields........................................................................................................................ 788

13.32 Trusted Descriptor Key Encryption Key Register (TDKEKR0 - TDKEKR7)........................................................................788

Section number Title Page

13.32.1 Offset....788

13.32.2 Function.... 788

13.32.3 Diagram....788

13.32.4 Fields....789

13.33 Trusted Descriptor Signing Key Register (TDSKR0 - TDSKR7)....789

13.33.1 Offset....789

13.33.2 Function.... 789

13.33.3 Diagram....790

13.33.4 Fields....790

13.34 Secure Key Nonce Register (SKNR)....790

13.34.1 Offset....790

13.34.2 Function.... 790

13.34.3 Diagram....791

13.34.4 Fields....791

13.35 DMA Status Register (DMA_STA)....792

13.35.1 Offset....792

13.35.2 Diagram....792

13.35.3 Fields....792

13.36 DMA_X_AID_7_4_MAP (DMA_X_AID_7_4_MAP)....793

13.36.1 Offset....793

13.36.2 Function.... 793

13.36.3 Diagram....794

13.36.4 Fields....794

13.37 DMA_X_AID_3_0_MAP (DMA_X_AID_3_0_MAP)....795

13.37.1 Offset....795

13.37.2 Function.... 795

13.37.3 Diagram....795

13.37.4 Fields....796

13.38 DMA_X_AID_15_12_MAP (DMA_X_AID_15_12_MAP)....796

13.38.1 Offset....796

13.38.2 Function....796

13.38.3 Diagram....797

13.38.4 Fields....797

13.39 DMA_X_AID_11_8_MAP (DMA_X_AID_11_8_MAP)....797

13.39.1 Offset....798

13.39.2 Function....798

13.39.3 Diagram....798

13.39.4 Fields....798

13.40 DMA_X AXI ID Map Enable Register (DMA_X_AID_15_0_EN)....799

13.40.1 Offset....799

13.40.2 Function....799

13.40.3 Diagram....799

13.40.4 Fields....800

13.41 DMA_X AXI Read Timing Check Control Register (DMA_X_ARTC_CTL)....801

13.41.1 Offset....801

13.41.2 Function....801

13.41.3 Diagram....802

13.41.4 Fields....802

13.42 DMA_X AXI Read Timing Check Late Count Register (DMA_X_ARTC_LC)....803

13.42.1 Offset....803

13.42.2 Function....803

13.42.3 Diagram....804

13.42.4 Fields....804

13.43 DMA_X AXI Read Timing Check Sample Count Register (DMA_X_ARTC_SC)....805

13.43.1 Offset....805

13.43.2 Function....805

13.43.3 Diagram....806

13.43.4 Fields....806

Section number Title Page

13.44 DMA_X Read Timing Check Latency Register (DMA_X_ARTC_LAT)....806

13.44.1 Offset....806

13.44.2 Function....806

13.44.3 Diagram....807

13.44.4 Fields....807

13.45 DMA_X AXI Write Timing Check Control Register (DMA_X_AWTC_CTL)....807

13.45.1 Offset....808

13.45.2 Function....808

13.45.3 Diagram....808

13.45.4 Fields....809

13.46 DMA_X AXI Write Timing Check Late Count Register (DMA_X_AWTC_LC)....810

13.46.1 Offset....810

13.46.2 Function....810

13.46.3 Diagram....811

13.46.4 Fields....811

13.47 DMA_X AXI Write Timing Check Sample Count Register (DMA_X_AWTC_SC)....811

13.47.1 Offset....811

13.47.2 Function....811

13.47.3 Diagram....812

13.47.4 Fields....812

13.48 DMA_X Write Timing Check Latency Register (DMA_X_AWTC_LAT)....813

13.48.1 Offset....813

13.48.2 Function....813

13.48.3 Diagram....813

13.48.4 Fields....814

13.49 RNG TRNG Miscellaneous Control Register (RTMCTL)....814

13.49.1 Offset....814

13.49.2 Function....814

13.49.3 Diagram....815

13.49.4 Fields....815

13.50 RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC)....816

13.50.1 Offset....817

13.50.2 Function....817

13.50.3 Diagram....817

13.50.4 Fields....817

13.51 RNG TRNG Poker Range Register (RTPKRRNG)....818

13.51.1 Offset....818

13.51.2 Function....818

13.51.3 Diagram....818

13.51.4 Fields....819

13.52 RNG TRNG Poker Square Calculation Result Register (RTPKRSQ)....819

13.52.1 Offset....819

13.52.2 Function....819

13.52.3 Diagram....819

13.52.4 Fields....820

13.53 RNG TRNG Poker Maximum Limit Register (RTPKRMAX)....820

13.53.1 Offset....820

13.53.2 Function....820

13.53.3 Diagram....821

13.53.4 Fields....821

13.54 RNG TRNG Seed Control Register (RTSDCTL)....821

13.54.1 Offset....821

13.54.2 Function....822

13.54.3 Diagram....822

13.54.4 Fields....822

13.55 RNG TRNG Total Samples Register (RTTOTSAM)....822

13.55.1 Offset....823

13.55.2 Function....823

| Section number | Title | Page |
|---|---|---|

13.55.3 Diagram....823

13.55.4 Fields....823

13.56 RNG TRNG Sparse Bit Limit Register (RTSBLIM).... 824

13.56.1 Offset....824

13.56.2 Function.... 824

13.56.3 Diagram....824

13.56.4 Fields....824

13.57 RNG TRNG Frequency Count Minimum Limit Register (RTFRQMIN)....825

13.57.1 Offset....825

13.57.2 Function.... 825

13.57.3 Diagram....825

13.57.4 Fields....826

13.58 RNG TRNG Frequency Count Register (RTFRQCNT)....826

13.58.1 Offset....826

13.58.2 Function.... 826

13.58.3 Diagram....826

13.58.4 Fields....827

13.59 RNG TRNG Frequency Count Maximum Limit Register (RTFRQMAX)....827

13.59.1 Offset....827

13.59.2 Function.... 827

13.59.3 Diagram....828

13.59.4 Fields....828

13.60 RNG TRNG Statistical Check Monobit Count Register (RTSCMC).... 828

13.60.1 Offset....828

13.60.2 Function.... 829

13.60.3 Diagram....829

13.60.4 Fields....829

13.61 RNG TRNG Statistical Check Monobit Limit Register (RTSCML).... 830

13.61.1 Offset....830

13.61.2 Function........................................................................................................................ 830

13.61.3 Diagram........................................................................................................................830

13.61.4 Fields........................................................................................................................... 830

13.62 RNG TRNG Statistical Check Run Length 1 Count Register (RTSCR1C)............................................... 831

13.62.1 Offset...........................................................................................................................831

13.62.2 Function........................................................................................................................ 831

13.62.3 Diagram........................................................................................................................831

13.62.4 Fields........................................................................................................................... 832

13.63 RNG TRNG Statistical Check Run Length 1 Limit Register (RTSCR1L)................................................ 832

13.63.1 Offset...........................................................................................................................832

13.63.2 Function........................................................................................................................ 833

13.63.3 Diagram........................................................................................................................833

13.63.4 Fields........................................................................................................................... 833

13.64 RNG TRNG Statistical Check Run Length 2 Count Register (RTSCR2C)............................................... 834

13.64.1 Offset...........................................................................................................................834

13.64.2 Function........................................................................................................................ 834

13.64.3 Diagram........................................................................................................................834

13.64.4 Fields........................................................................................................................... 835

13.65 RNG TRNG Statistical Check Run Length 2 Limit Register (RTSCR2L)................................................ 835

13.65.1 Offset...........................................................................................................................835

13.65.2 Function........................................................................................................................ 836

13.65.3 Diagram........................................................................................................................836

13.65.4 Fields........................................................................................................................... 836

13.66 RNG TRNG Statistical Check Run Length 3 Limit Register (RTSCR3L)................................................ 837

13.66.1 Offset...........................................................................................................................837

13.66.2 Function........................................................................................................................ 837

13.66.3 Diagram........................................................................................................................837

13.66.4 Fields........................................................................................................................... 838

13.67 RNG TRNG Statistical Check Run Length 3 Count Register (RTSCR3C)............................................... 838

13.67.1 Offset....838

13.67.2 Function.... 838

13.67.3 Diagram....839

13.67.4 Fields....839

13.68 RNG TRNG Statistical Check Run Length 4 Limit Register (RTSCR4L).... 839

13.68.1 Offset....840

13.68.2 Function.... 840

13.68.3 Diagram....840

13.68.4 Fields....840

13.69 RNG TRNG Statistical Check Run Length 4 Count Register (RTSCR4C).... 841

13.69.1 Offset....841

13.69.2 Function.... 841

13.69.3 Diagram....841

13.69.4 Fields....842

13.70 RNG TRNG Statistical Check Run Length 5 Count Register (RTSCR5C).... 842

13.70.1 Offset....842

13.70.2 Function.... 842

13.70.3 Diagram....843

13.70.4 Fields....843

13.71 RNG TRNG Statistical Check Run Length 5 Limit Register (RTSCR5L).... 843

13.71.1 Offset....844

13.71.2 Function.... 844

13.71.3 Diagram....844

13.71.4 Fields....844

13.72 RNG TRNG Statistical Check Run Length 6+ Limit Register (RTSCR6PL)....845

13.72.1 Offset....845

13.72.2 Function.... 845

13.72.3 Diagram....845

13.72.4 Fields....846

13.73 RNG TRNG Statistical Check Run Length 6+ Count Register (RTSCR6PC)....846
13.73.1 Offset....846
13.73.2 Function....846
13.73.3 Diagram....847
13.73.4 Fields....847
13.74 RNG TRNG Status Register (RTSTATUS)....847
13.74.1 Offset....848
13.74.2 Function....848
13.74.3 Diagram....848
13.74.4 Fields....849
13.75 RNG TRNG Entropy Read Register (RTENT0 - RTENT15)....850
13.75.1 Offset....850
13.75.2 Function....850
13.75.3 Diagram....851
13.75.4 Fields....851
13.76 RNG TRNG Statistical Check Poker Count 1 and 0 Register (RTPKRCNT10)....851
13.76.1 Offset....851
13.76.2 Function....851
13.76.3 Diagram....852
13.76.4 Fields....852
13.77 RNG TRNG Statistical Check Poker Count 3 and 2 Register (RTPKRCNT32)....852
13.77.1 Offset....852
13.77.2 Function....853
13.77.3 Diagram....853
13.77.4 Fields....853
13.78 RNG TRNG Statistical Check Poker Count 5 and 4 Register (RTPKRCNT54)....853
13.78.1 Offset....854
13.78.2 Function....854
13.78.3 Diagram....854

Section number Title Page

13.78.4 Fields....854

13.79 RNG TRNG Statistical Check Poker Count 7 and 6 Register (RTPKRCNT76)....855

13.79.1 Offset....855

13.79.2 Function....855

13.79.3 Diagram....855

13.79.4 Fields....855

13.80 RNG TRNG Statistical Check Poker Count 9 and 8 Register (RTPKRCNT98)....856

13.80.1 Offset....856

13.80.2 Function....856

13.80.3 Diagram....856

13.80.4 Fields....856

13.81 RNG TRNG Statistical Check Poker Count B and A Register (RTPKRCNTBA)....857

13.81.1 Offset....857

13.81.2 Function....857

13.81.3 Diagram....857

13.81.4 Fields....858

13.82 RNG TRNG Statistical Check Poker Count D and C Register (RTPKRCNTDC)....858

13.82.1 Offset....858

13.82.2 Function....858

13.82.3 Diagram....859

13.82.4 Fields....859

13.83 RNG TRNG Statistical Check Poker Count F and E Register (RTPKRCNTFE)....859

13.83.1 Offset....859

13.83.2 Function....860

13.83.3 Diagram....860

13.83.4 Fields....860

13.84 RNG DRNG Status Register (RDSTA)....860

13.84.1 Offset....861

13.84.2 Function....861

13.84.3 Diagram....861

13.84.4 Fields....861

13.85 RNG DRNG State Handle 0 Reseed Interval Register (RDINT0)....862

13.85.1 Offset....862

13.85.2 Function....862

13.85.3 Diagram....863

13.85.4 Fields....863

13.86 RNG DRNG State Handle 1 Reseed Interval Register (RDINT1)....863

13.86.1 Offset....863

13.86.2 Function....864

13.86.3 Diagram....864

13.86.4 Fields....864

13.87 RNG DRNG Hash Control Register (RDHCNTL)....865

13.87.1 Offset....865

13.87.2 Function....865

13.87.3 Diagram....865

13.87.4 Fields....865

13.88 RNG DRNG Hash Digest Register (RDHDIG)....866

13.88.1 Offset....866

13.88.2 Function....866

13.88.3 Diagram....866

13.88.4 Fields....867

13.89 RNG DRNG Hash Buffer Register (RDHBUF)....867

13.89.1 Offset....867

13.89.2 Function....867

13.89.3 Diagram....867

13.89.4 Fields....868

13.90 Recoverable Error Indication Status (REIS)....868

13.90.1 Offset....868

13.90.2 Function.......................................................................................................................... 868

13.90.3 Diagram..........................................................................................................................868

13.90.4 Fields.............................................................................................................................. 869

13.91 Recoverable Error Indication Halt (REIH)..........................................................................................869

13.91.1 Offset..............................................................................................................................869

13.91.2 Function.......................................................................................................................... 870

13.91.3 Diagram..........................................................................................................................870

13.91.4 Fields.............................................................................................................................. 870

13.92 SEC Version ID Register, most-significant half (SECVID_MS)...........................................................871

13.92.1 Offset..............................................................................................................................871

13.92.2 Function.......................................................................................................................... 872

13.92.3 Diagram..........................................................................................................................872

13.92.4 Fields.............................................................................................................................. 872

13.93 SEC Version ID Register, least-significant half (SECVID_LS)............................................................. 873

13.93.1 Offset..............................................................................................................................873

13.93.2 Function.......................................................................................................................... 873

13.93.3 Diagram..........................................................................................................................873

13.93.4 Fields.............................................................................................................................. 874

13.94 Holding Tank 0 Job Descriptor Address (HT0_JD_ADDR).................................................................874

13.94.1 Offset..............................................................................................................................874

13.94.2 Function.......................................................................................................................... 874

13.94.3 Diagram..........................................................................................................................875

13.94.4 Fields.............................................................................................................................. 875

13.95 Holding Tank 0 Shared Descriptor Address (HT0_SD_ADDR)...........................................................876

13.95.1 Offset..............................................................................................................................876

13.95.2 Function.......................................................................................................................... 876

13.95.3 Diagram..........................................................................................................................876

13.95.4 Fields.............................................................................................................................. 877

13.96 Holding Tank 0 Job Queue Control, most-significant half (HT0_JQ_CTRL_MS).................................. 877

13.96.1 Offset....877

13.96.2 Function.... 878

13.96.3 Diagram....878

13.96.4 Fields....878

13.97 Holding Tank 0 Job Queue Control, least-significant half (HT0_JQ_CTRL_LS)....880

13.97.1 Offset....880

13.97.2 Function.... 880

13.97.3 Diagram....881

13.97.4 Fields....881

13.98 Holding Tank Status (HT0_STATUS).... 881

13.98.1 Offset....881

13.98.2 Function.... 882

13.98.3 Diagram....882

13.98.4 Fields....882

13.99 Job Queue Debug Select Register (JQ_DEBUG_SEL)....883

13.99.1 Offset....883

13.99.2 Function.... 883

13.99.3 Diagram....884

13.99.4 Fields....884

13.100 Job Ring Job IDs in Use Register, least-significant half (JRJIDU_LS).... 884

13.100.1 Offset....884

13.100.2 Function.... 885

13.100.3 Diagram....885

13.100.4 Fields....885

13.101 Job Ring Job-Done Job ID FIFO BC (JRJDJIFBC)....886

13.101.1 Offset....886

13.101.2 Function.... 886

13.101.3 Diagram....887

13.101.4 Fields....887

Section number Title Page

13.102 Job Ring Job-Done Job ID FIFO (JRJDJIF)....887

13.102.1 Offset....887

13.102.2 Diagram....888

13.102.3 Fields....888

13.103 Job Ring Job-Done Source 1 (JRJDS1)....888

13.103.1 Offset....888

13.103.2 Function....889

13.103.3 Diagram....889

13.103.4 Fields....889

13.104 Job Ring Job-Done Descriptor Address 0 Register (JRJDDA)....889

13.104.1 Offset....889

13.104.2 Function....890

13.104.3 Diagram....890

13.104.4 Fields....891

13.105 Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)....891

13.105.1 Offset....891

13.105.2 Function....892

13.105.3 Diagram....892

13.105.4 Fields....893

13.106 Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)....893

13.106.1 Offset....893

13.106.2 Function....894

13.106.3 Diagram....894

13.106.4 Fields....895

13.107 Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)....895

13.107.1 Offset....895

13.107.2 Function....896

13.107.3 Diagram....897

13.107.4 Fields....897

Section number Title Page

13.108 Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)....897

13.108.1 Offset....898

13.108.2 Function.... 898

13.108.3 Diagram....899

13.108.4 Fields....899

13.109 Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)....900

13.109.1 Offset....900

13.109.2 Function.... 900

13.109.3 Diagram....901

13.109.4 Fields....901

13.110 Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)....902

13.110.1 Offset....902

13.110.2 Function.... 903

13.110.3 Diagram....903

13.110.4 Fields....904

13.111 Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED).... 904

13.111.1 Offset....904

13.111.2 Function.... 905

13.111.3 Diagram....906

13.111.4 Fields....906

13.112 CHA Revision Number Register, most-significant half (CRNR_MS)....907

13.112.1 Offset....907

13.112.2 Function.... 907

13.112.3 Diagram....907

13.112.4 Fields....908

13.113 CHA Revision Number Register, least-significant half (CRNR_LS).... 908

13.113.1 Offset....908

13.113.2 Function.... 909

13.113.3 Diagram....909

Section number Title Page

13.113.4 Fields....909

13.114 Compile Time Parameters Register, most-significant half (CTPR_MS)....910

13.114.1 Offset....910

13.114.2 Function....911

13.114.3 Diagram....911

13.114.4 Fields....911

13.115 Compile Time Parameters Register, least-significant half (CTPR_LS)....913

13.115.1 Offset....913

13.115.2 Function....914

13.115.3 Diagram....914

13.115.4 Fields....914

13.116 Fault Address Register (FAR)....915

13.116.1 Offset....915

13.116.2 Function....916

13.116.3 Diagram....917

13.116.4 Fields....917

13.117 Fault Address ICID Register (FAICID)....917

13.117.1 Offset....917

13.117.2 Function....918

13.117.3 Diagram....918

13.117.4 Fields....919

13.118 Fault Address Detail Register (FADR)....919

13.118.1 Offset....919

13.118.2 Function....919

13.118.3 Diagram....920

13.118.4 Fields....920

13.119 SEC Status Register (SSTA)....921

13.119.1 Offset....921

13.119.2 Function....921

13.119.3 Diagram....922

13.119.4 Fields....922

13.120 RTIC Version ID Register (RVID)....923

13.120.1 Offset....923

13.120.2 Function....923

13.120.3 Diagram....924

13.120.4 Fields....924

13.121 CHA Cluster Block Version ID Register (CCBVID)....925

13.121.1 Offset....925

13.121.2 Function....925

13.121.3 Diagram....925

13.121.4 Fields....926

13.122 CHA Version ID Register, most-significant half (CHAVID_MS)....926

13.122.1 Offset....926

13.122.2 Function....927

13.122.3 Diagram....927

13.122.4 Fields....927

13.123 CHA Version ID Register, least-significant half (CHAVID_LS)....928

13.123.1 Offset....928

13.123.2 Function....928

13.123.3 Diagram....929

13.123.4 Fields....929

13.124 CHA Number Register, most-significant half (CHANUM_MS)....930

13.124.1 Offset....930

13.124.2 Function....930

13.124.3 Diagram....931

13.124.4 Fields....931

13.125 CHA Number Register, least-significant half (CHANUM_LS)....932

13.125.1 Offset....932

| Section number | Title | Page |
|---|---|---|

13.125.2 Function.................................................. 932

13.125.3 Diagram.................................................. 932

13.125.4 Fields.................................................. 933

13.126 Input Ring Base Address Register for Job Ring a (IRBAR_JR0 - IRBAR_JR3).................................................. 933

13.126.1 Offset.................................................. 933

13.126.2 Function.................................................. 933

13.126.3 Diagram.................................................. 934

13.126.4 Fields.................................................. 935

13.127 Input Ring Size Register for Job Ring a (IRSR_JR0 - IRSR_JR3).................................................. 935

13.127.1 Offset.................................................. 935

13.127.2 Function.................................................. 935

13.127.3 Diagram.................................................. 936

13.127.4 Fields.................................................. 936

13.128 Input Ring Slots Available Register for Job Ring a (IRSAR_JR0 - IRSAR_JR3).................................................. 936

13.128.1 Offset.................................................. 936

13.128.2 Function.................................................. 937

13.128.3 Diagram.................................................. 937

13.128.4 Fields.................................................. 937

13.129 Input Ring Jobs Added Register for Job Ringa (IRJAR_JR0 - IRJAR_JR3).................................................. 938

13.129.1 Offset.................................................. 938

13.129.2 Function.................................................. 938

13.129.3 Diagram.................................................. 939

13.129.4 Fields.................................................. 939

13.130 Output Ring Base Address Register for Job Ring a (ORBAR_JR0 - ORBAR_JR3).................................................. 939

13.130.1 Offset.................................................. 939

13.130.2 Function.................................................. 939

13.130.3 Diagram.................................................. 940

13.130.4 Fields.................................................. 941

13.131 Output Ring Size Register for Job Ring a (ORSR_JR0 - ORSR_JR3).................................................. 941

13.131.1 Offset....941

13.131.2 Function.... 942

13.131.3 Diagram....942

13.131.4 Fields....943

13.132 Output Ring Jobs Removed Register for Job Ring a (ORJRR_JR0 - ORJRR_JR3)....943

13.132.1 Offset....943

13.132.2 Function.... 943

13.132.3 Diagram....944

13.132.4 Fields....944

13.133 Output Ring Slots Full Register for Job Ring a (ORSFR_JR0 - ORSFR_JR3).... 945

13.133.1 Offset....945

13.133.2 Function.... 945

13.133.3 Diagram....945

13.133.4 Fields....946

13.134 Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR3)....946

13.134.1 Offset....946

13.134.2 Function.... 946

13.134.3 Diagram....947

13.134.4 Fields....947

13.135 Job Ring Interrupt Status Register for Job Ring a (JRINTR_JR0 - JRINTR_JR3)....948

13.135.1 Offset....948

13.135.2 Function.... 948

13.135.3 Diagram....948

13.135.4 Fields....949

13.136 Job Ring Configuration Register for Job Ring a, most-significant half (JRCFGR_JR0_MS - JRCFGR_JR3_MS). 950

13.136.1 Offset....950

13.136.2 Function.... 951

13.136.3 Diagram....952

13.136.4 Fields....952

| Section number | Title | Page |
|---|---|---|

13.137 Job Ring Configuration Register for Job Ring a, least-significant half (JRCFGR_JR0_LS - JRCFGR_JR3_LS)....957

13.137.1 Offset....957

13.137.2 Function....957

13.137.3 Diagram....957

13.137.4 Fields....957

13.138 Input Ring Read Index Register for Job Ring a (IRRIR_JR0 - IRRIR_JR3)....958

13.138.1 Offset....958

13.138.2 Function....958

13.138.3 Diagram....959

13.138.4 Fields....959

13.139 Output Ring Write Index Register for Job Ring a (ORWIR_JR0 - ORWIR_JR3)....960

13.139.1 Offset....960

13.139.2 Function....960

13.139.3 Diagram....960

13.139.4 Fields....961

13.140 Job Ring Command Register for Job Ring a (JRCR_JR0 - JRCR_JR3)....961

13.140.1 Offset....961

13.140.2 Function....961

13.140.3 Diagram....963

13.140.4 Fields....963

13.141 Job Ring a Address-Array Valid Register (JR0AAV - JR3AAV)....964

13.141.1 Offset....964

13.141.2 Function....964

13.141.3 Diagram....964

13.141.4 Fields....965

13.142 Job Ring a Address-Array Address b Register (JR0AAA0 - JR3AAA7)....966

13.142.1 Offset....966

13.142.2 Function....966

13.142.3 Diagram....966

13.142.4 Fields....967

13.143 Recoverable Error Indication Record 0 for Job Ring a (REIR0JR0 - REIR0JR3)....967

13.143.1 Offset....967

13.143.2 Function....968

13.143.3 Diagram....968

13.143.4 Fields....968

13.144 Recoverable Error Indication Record 2 for Job Ring a (REIR2JR0 - REIR2JR3)....969

13.144.1 Offset....969

13.144.2 Function....969

13.144.3 Diagram....969

13.144.4 Fields....970

13.145 Recoverable Error Indication Record 4 for Job Ring a (REIR4JR0 - REIR4JR3)....970

13.145.1 Offset....970

13.145.2 Function....971

13.145.3 Diagram....971

13.145.4 Fields....971

13.146 Recoverable Error Indication Record 5 for Job Ring a (REIR5JR0 - REIR5JR3)....972

13.146.1 Offset....972

13.146.2 Function....972

13.146.3 Diagram....972

13.146.4 Fields....972

13.147 RTIC Status Register (RSTA)....973

13.147.1 Offset....973

13.147.2 Function....973

13.147.3 Diagram....973

13.147.4 Fields....974

13.148 RTIC Command Register (RCMD)....976

13.148.1 Offset....976

13.148.2 Function....976

13.148.3 Diagram....976

13.148.4 Fields....976

13.149 RTIC Control Register (RCTL)....977

13.149.1 Offset....977

13.149.2 Function....978

13.149.3 Diagram....978

13.149.4 Fields....978

13.150 RTIC Throttle Register (RTHR)....980

13.150.1 Offset....980

13.150.2 Function....980

13.150.3 Diagram....980

13.150.4 Fields....981

13.151 RTIC Watchdog Timer (RWDOG)....981

13.151.1 Offset....981

13.151.2 Function....981

13.151.3 Diagram....982

13.151.4 Fields....982

13.152 RTIC Endian Register (REND)....983

13.152.1 Offset....983

13.152.2 Function....983

13.152.3 Diagram....983

13.152.4 Fields....983

13.153 RTIC Memory Block a Address b Register (RMAA0 - RMDA1)....985

13.153.1 Offset....985

13.153.2 Function....985

13.153.3 Diagram....985

13.153.4 Fields....986

13.154 RTIC Memory Block a Length b Register (RMAL0 - RMDL1)....986

13.154.1 Offset....986

13.154.2 Function.......... 987

13.154.3 Diagram.......... 987

13.154.4 Fields.......... 988

13.155 RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31).......... 988

13.155.1 Offset.......... 988

13.155.2 Function.......... 999

13.155.3 Diagram.......... 999

13.155.4 Fields.......... 1000

13.156 Recoverable Error Indication Record 0 for RTIC (REIR0RTIC).......... 1000

13.156.1 Offset.......... 1000

13.156.2 Function.......... 1000

13.156.3 Diagram.......... 1001

13.156.4 Fields.......... 1001

13.157 Recoverable Error Indication Record 2 for RTIC (REIR2RTIC).......... 1001

13.157.1 Offset.......... 1001

13.157.2 Function.......... 1002

13.157.3 Diagram.......... 1002

13.157.4 Fields.......... 1002

13.158 Recoverable Error Indication Record 4 for RTIC (REIR4RTIC).......... 1003

13.158.1 Offset.......... 1003

13.158.2 Function.......... 1003

13.158.3 Diagram.......... 1003

13.158.4 Fields.......... 1003

13.159 Recoverable Error Indication Record 5 for RTIC (REIR5RTIC).......... 1004

13.159.1 Offset.......... 1004

13.159.2 Function.......... 1004

13.159.3 Diagram.......... 1004

13.159.4 Fields.......... 1005

13.160 Queue Interface Control Register, most-significant (QICTL_MS).......... 1005

Section number Title Page

13.160.1 Offset....1005

13.160.2 Function.... 1006

13.160.3 Diagram....1006

13.160.4 Fields....1006

13.161 Queue Interface Control Register, least-significant (QICTL_LS)....1010

13.161.1 Offset....1010

13.161.2 Function.... 1010

13.161.3 Diagram....1010

13.161.4 Fields....1011

13.162 Queue Interface Status Register (QISTA).... 1011

13.162.1 Offset....1012

13.162.2 Function.... 1012

13.162.3 Diagram....1012

13.162.4 Fields....1012

13.163 Queue Interface Dequeue Configuration Register, most-significant half (QIDQC_MS)....1013

13.163.1 Offset....1013

13.163.2 Function.... 1014

13.163.3 Diagram....1014

13.163.4 Fields....1014

13.164 Queue Interface Dequeue Configuration Register, least-significant half (QIDQC_LS).... 1015

13.164.1 Offset....1015

13.164.2 Function.... 1015

13.164.3 Diagram....1015

13.164.4 Fields....1016

13.165 Queue Interface Enqueue Configuration Register, most-significant half (QIEQC_MS).... 1016

13.165.1 Offset....1016

13.165.2 Function.... 1016

13.165.3 Diagram....1016

13.165.4 Fields....1017

Section number Title Page

13.166 Queue Interface Enqueue Configuration Register, least-significant half (QIEQC_LS)............1017

13.166.1 Offset............1017

13.166.2 Function............1017

13.166.3 Diagram............1018

13.166.4 Fields............1018

13.167 Queue Interface ICID Configuration Register, most-significant half (QIIC_MS)............1018

13.167.1 Offset............1018

13.167.2 Function............1018

13.167.3 Diagram............1019

13.167.4 Fields............1019

13.168 Queue Interface ICID Configuration Register, least-significant half (QIIC_LS)............1019

13.168.1 Offset............1019

13.168.2 Function............1020

13.168.3 Diagram............1020

13.168.4 Fields............1020

13.169 Queue Interface Descriptor Word 0 Register (QIDESC0)............1021

13.169.1 Offset............1021

13.169.2 Function............1021

13.169.3 Diagram............1021

13.169.4 Fields............1022

13.170 Queue Interface Descriptor Word a Registers (QIDESC1 - QIDESC12)............1022

13.170.1 Offset............1022

13.170.2 Diagram............1022

13.170.3 Fields............1022

13.171 Queue Interface Compound Frame Scatter/Gather Table Registers (QICFOFH_MS - QICFIFL_LS)............1023

13.171.1 Offset............1023

13.171.2 Function............1023

13.171.3 Diagram............1023

13.171.4 Fields............1024

13.172 Queue Interface Job ID Valid Register (QIJIDVALID)....1024

13.172.1 Offset....1024

13.172.2 Function.... 1024

13.172.3 Diagram....1025

13.172.4 Fields....1026

13.173 Queue Interface Job ID Job Ready Register (QIJIDRDY)....1026

13.173.1 Offset....1027

13.173.2 Diagram....1027

13.173.3 Fields....1027

13.174 Recoverable Error Indication Record 0 for the Queue Interface (REIR0QI).... 1028

13.174.1 Offset....1028

13.174.2 Function.... 1028

13.174.3 Diagram....1028

13.174.4 Fields....1029

13.175 Recoverable Error Indication Record 1 for the Queue Interface (REIR1QI).... 1029

13.175.1 Offset....1029

13.175.2 Function.... 1030

13.175.3 Diagram....1030

13.175.4 Fields....1030

13.176 Recoverable Error Indication Record 2 for the Queue Interface (REIR2QI).... 1030

13.176.1 Offset....1031

13.176.2 Function.... 1031

13.176.3 Diagram....1031

13.176.4 Fields....1031

13.177 Recoverable Error Indication Record 4 for the Queue Interface (REIR4QI).... 1032

13.177.1 Offset....1032

13.177.2 Function.... 1032

13.177.3 Diagram....1032

13.177.4 Fields....1032

13.178 Recoverable Error Indication Record 5 for the Queue Interface (REIR5QI).......... 1033

13.178.1 Offset.......... 1033

13.178.2 Function.......... 1033

13.178.3 Diagram.......... 1033

13.178.4 Fields.......... 1034

13.179 CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG).......... 1034

13.179.1 Offset.......... 1034

13.179.2 Function.......... 1035

13.179.3 Diagram.......... 1035

13.179.4 Fields.......... 1035

13.180 CCB a Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK - C2C1MR_PK).......... 1038

13.180.1 Offset.......... 1038

13.180.2 Function.......... 1038

13.180.3 Diagram.......... 1039

13.180.4 Fields.......... 1039

13.181 CCB a Class 1 Mode Register Format for Non-Public Key Algorithms (C0C1MR_NPK - C2C1MR_NPK).......... 1040

13.181.1 Offset.......... 1040

13.181.2 Function.......... 1040

13.181.3 Diagram.......... 1041

13.181.4 Fields.......... 1041

13.182 CCB a Class 1 Key Size Register (C0C1KSR - C2C1KSR).......... 1044

13.182.1 Offset.......... 1044

13.182.2 Function.......... 1045

13.182.3 Diagram.......... 1045

13.182.4 Fields.......... 1045

13.183 CCB a Class 1 Data Size Register (C0C1DSR - C2C1DSR).......... 1046

13.183.1 Offset.......... 1046

13.183.2 Function.......... 1046

13.183.3 Diagram.......... 1046

13.183.4 Fields....1047

13.184 CCB a Class 1 ICV Size Register (C0C1ICVSR - C2C1ICVSR)....1047

13.184.1 Offset....1048

13.184.2 Function....1048

13.184.3 Diagram....1048

13.184.4 Fields....1048

13.185 CCB a CHA Control Register (C0CCTRL - C2CCTRL)....1049

13.185.1 Offset....1049

13.185.2 Function....1049

13.185.3 Diagram....1049

13.185.4 Fields....1050

13.186 CCB a Interrupt Control Register (C0ICTL - C2ICTL)....1052

13.186.1 Offset....1052

13.186.2 Function....1052

13.186.3 Diagram....1052

13.186.4 Fields....1053

13.187 CCB a Clear Written Register (C0CWR - C2CWR)....1057

13.187.1 Offset....1057

13.187.2 Function....1057

13.187.3 Diagram....1057

13.187.4 Fields....1057

13.188 CCB a Status and Error Register, most-significant half (C0CSTA_MS - C2CSTA_MS)....1059

13.188.1 Offset....1059

13.188.2 Function....1059

13.188.3 Diagram....1060

13.188.4 Fields....1060

13.189 CCB a Status and Error Register, least-significant half (C0CSTA_LS - C2CSTA_LS)....1061

13.189.1 Offset....1061

13.189.2 Function....1061

| Section number | Title | Page |
|---|---|---|

13.189.3 Diagram....1061

13.189.4 Fields....1062

13.190 CCB a AAD Size Register (C0AADSZR - C2AADSZR)....1064

13.190.1 Offset....1064

13.190.2 Function....1064

13.190.3 Diagram....1064

13.190.4 Fields....1065

13.191 Class 1 IV Size Register (C0C1IVSZR - C2C1IVSZR)....1065

13.191.1 Offset....1065

13.191.2 Function....1065

13.191.3 Diagram....1065

13.191.4 Fields....1066

13.192 PKHA A Size Register (C0PKASZR - C2PKASZR)....1066

13.192.1 Offset....1066

13.192.2 Function....1066

13.192.3 Diagram....1067

13.192.4 Fields....1067

13.193 PKHA B Size Register (C0PKBSZR - C2PKBSZR)....1067

13.193.1 Offset....1067

13.193.2 Function....1068

13.193.3 Diagram....1068

13.193.4 Fields....1068

13.194 PKHA N Size Register (C0PKNSZR - C2PKNSZR)....1068

13.194.1 Offset....1068

13.194.2 Function....1069

13.194.3 Diagram....1069

13.194.4 Fields....1069

13.195 PKHA E Size Register (C0PKESZR - C2PKESZR)....1070

13.195.1 Offset....1070

| Section number | Title | Page |
|---|---|---|

13.195.2 Function....1070

13.195.3 Diagram....1070

13.195.4 Fields....1070

13.196 CCB a Class 1 Context Register Word b (C0C1CTXR0 - C2C1CTXR15)....1071

13.196.1 Offset....1071

13.196.2 Function....1071

13.196.3 Diagram....1072

13.196.4 Fields....1073

13.197 CCB a Class 1 Key Registers Word b (C0C1KR0 - C2C1KR7)....1073

13.197.1 Offset....1073

13.197.2 Function....1073

13.197.3 Diagram....1074

13.197.4 Fields....1075

13.198 CCB a Class 2 Mode Register (C0C2MR - C2C2MR)....1075

13.198.1 Offset....1075

13.198.2 Function....1075

13.198.3 Diagram....1075

13.198.4 Fields....1076

13.199 CCB a Class 2 Key Size Register (C0C2KSR - C2C2KSR)....1077

13.199.1 Offset....1077

13.199.2 Function....1077

13.199.3 Diagram....1078

13.199.4 Fields....1078

13.200 CCB a Class 2 Data Size Register (C0C2DSR - C2C2DSR)....1078

13.200.1 Offset....1078

13.200.2 Function....1079

13.200.3 Diagram....1079

13.200.4 Fields....1080

13.201 CCB a Class 2 ICV Size Register (C0C2ICVSZR - C2C2ICVSZR)....1080

13.201.1 Offset....1081

13.201.2 Function....1081

13.201.3 Diagram....1081

13.201.4 Fields....1081

13.202 CCB a Class 2 Context Register Word b (C0C2CTXR0 - C2C2CTXR17)....1082

13.202.1 Offset....1082

13.202.2 Function....1082

13.202.3 Diagram....1083

13.202.4 Fields....1083

13.203 CCB a Class 2 Key Register Word b (C0C2KEYR0 - C2C2KEYR31)....1083

13.203.1 Offset....1083

13.203.2 Function....1083

13.203.3 Diagram....1084

13.203.4 Fields....1084

13.204 CCB a FIFO Status (C0FIFOSTA - C2FIFOSTA)....1084

13.204.1 Offset....1084

13.204.2 Function....1085

13.204.3 Diagram....1085

13.204.4 Fields....1085

13.205 CCB a iNformation FIFO When STYPE Is Not 10 (C0NFIFO - C2NFIFO)....1086

13.205.1 Offset....1086

13.205.2 Function....1086

13.205.3 Diagram....1087

13.205.4 Fields....1087

13.206 CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C2NFIFO_2)....1090

13.206.1 Offset....1090

13.206.2 Function....1090

13.206.3 Diagram....1090

13.206.4 Fields....1090

Section number Title Page

13.207 CCB a Input Data FIFO (C0IFIFO - C2IFIFO)....1092

13.207.1 Offset....1092

13.207.2 Function....1093

13.207.3 Diagram....1093

13.207.4 Fields....1093

13.208 CCB a Output Data FIFO (C0OFIFO - C2OFIFO)....1094

13.208.1 Offset....1094

13.208.2 Function....1094

13.208.3 Diagram....1095

13.208.4 Fields....1095

13.209 DECOa Job Queue Control Register, most-significant half (D0JQCR_MS - D2JQCR_MS)....1096

13.209.1 Offset....1096

13.209.2 Function....1096

13.209.3 Diagram....1096

13.209.4 Fields....1097

13.210 DECOa Job Queue Control Register, least-significant half (D0JQCR_LS - D2JQCR_LS)....1098

13.210.1 Offset....1098

13.210.2 Function....1099

13.210.3 Diagram....1099

13.210.4 Fields....1099

13.211 DECOa Descriptor Address Register (D0DAR - D2DAR)....1099

13.211.1 Offset....1099

13.211.2 Function....1100

13.211.3 Diagram....1100

13.211.4 Fields....1100

13.212 DECOa Operation Status Register, most-significant half (D0OPSTA_MS - D2OPSTA_MS)....1101

13.212.1 Offset....1101

13.212.2 Function....1101

13.212.3 Diagram....1101

13.212.4 Fields....1102

13.213 DECOa Operation Status Register, least-significant half (D0OPSTA_LS - D2OPSTA_LS)....1103

13.213.1 Offset....1103

13.213.2 Function....1103

13.213.3 Diagram....1104

13.213.4 Fields....1104

13.214 DECOa Checksum Register (D0CKSUMR - D2CKSUMR)....1104

13.214.1 Offset....1104

13.214.2 Function....1105

13.214.3 Diagram....1105

13.214.4 Fields....1105

13.215 DECOa SDID / Trusted ICID Status Register (D0SDIDSR - D2SDIDSR)....1106

13.215.1 Offset....1106

13.215.2 Function....1106

13.215.3 Diagram....1106

13.215.4 Fields....1106

13.216 DECOa ICID Status Register (D0ISR - D2ISR)....1107

13.216.1 Offset....1107

13.216.2 Function....1107

13.216.3 Diagram....1107

13.216.4 Fields....1108

13.217 DECOa Math Register b_MS (D0MTH0_MS - D2MTH7_MS)....1108

13.217.1 Offset....1108

13.217.2 Function....1109

13.217.3 Diagram....1109

13.217.4 Fields....1109

13.218 DECOa Math Register b_LS (D0MTH0_LS - D2MTH7_LS)....1109

13.218.1 Offset....1109

13.218.2 Function....1110

Section number Title Page

13.218.3 Diagram....1110

13.218.4 Fields....1110

13.219 DECOa Gather Table Register b Word 0 (D0GTR0_0 - D2GTR3_0)....1110

13.219.1 Offset....1110

13.219.2 Function.... 1111

13.219.3 Diagram....1111

13.219.4 Fields....1111

13.220 DECOa Gather Table Register b Word 1 (D0GTR0_1 - D2GTR3_1)....1111

13.220.1 Offset....1111

13.220.2 Diagram....1112

13.220.3 Fields....1112

13.221 DECOa Gather Table Register b Word 2 (D0GTR0_2 - D2GTR3_2)....1112

13.221.1 Offset....1112

13.221.2 Diagram....1113

13.221.3 Fields....1113

13.222 DECOa Gather Table Register b Word 3 (D0GTR0_3 - D2GTR3_3)....1113

13.222.1 Offset....1113

13.222.2 Diagram....1114

13.222.3 Fields....1114

13.223 DECOa Scatter Table Register b Word 0 (D0STR0_0 - D2STR3_0)....1114

13.223.1 Offset....1114

13.223.2 Diagram....1115

13.223.3 Fields....1115

13.224 DECOa Scatter Table Register b Word 1 (D0STR0_1 - D2STR3_1)....1115

13.224.1 Offset....1115

13.224.2 Diagram....1116

13.224.3 Fields....1116

13.225 DECOa Scatter Table Register b Word 2 (D0STR0_2 - D2STR3_2)....1116

13.225.1 Offset....1116

13.225.2 Diagram....1116

13.225.3 Fields....1117

13.226 DECOa Scatter Table Register b Word 3 (D0STR0_3 - D2STR3_3)....1117

13.226.1 Offset....1117

13.226.2 Diagram....1117

13.226.3 Fields....1118

13.227 DECOa Descriptor Buffer Word b (D0DESB0 - D2DESB63)....1118

13.227.1 Offset....1118

13.227.2 Function....1118

13.227.3 Diagram....1119

13.227.4 Fields....1119

13.228 DECOa Debug Job (D0DJR - D2DJR)....1120

13.228.1 Offset....1120

13.228.2 Function....1120

13.228.3 Diagram....1120

13.228.4 Fields....1121

13.229 DECOa Debug DECO (D0DDR - D2DDR)....1122

13.229.1 Offset....1122

13.229.2 Function....1122

13.229.3 Diagram....1123

13.229.4 Fields....1123

13.230 DECOa Debug Job Pointer (D0DJP - D2DJP)....1125

13.230.1 Offset....1125

13.230.2 Function....1125

13.230.3 Diagram....1125

13.230.4 Fields....1126

13.231 DECOa Debug Shared Pointer (D0SDP - D2SDP)....1126

13.231.1 Offset....1126

13.231.2 Function....1127

13.231.3 Diagram....1127

13.231.4 Fields....1127

13.232 DECOa Debug_ICID, most-significant half (D0DIR_MS - D2DIR_MS)....1128

13.232.1 Offset....1128

13.232.2 Function....1128

13.232.3 Diagram....1128

13.232.4 Fields....1129

13.233 Sequence Output Length Register (SOL0 - SOL2)....1129

13.233.1 Offset....1129

13.233.2 Function....1130

13.233.3 Diagram....1130

13.233.4 Fields....1130

13.234 Variable Sequence Output Length Register (VSOL0 - VSOL2)....1130

13.234.1 Offset....1130

13.234.2 Function....1131

13.234.3 Diagram....1131

13.234.4 Fields....1131

13.235 Sequence Input Length Register (SIL0 - SIL2)....1132

13.235.1 Offset....1132

13.235.2 Function....1132

13.235.3 Diagram....1132

13.235.4 Fields....1133

13.236 Variable Sequence Input Length Register (VSIL0 - VSIL2)....1133

13.236.1 Offset....1133

13.236.2 Function....1133

13.236.3 Diagram....1134

13.236.4 Fields....1134

13.237 Protocol Override Register (D0POVRD - D2POVRD)....1134

13.237.1 Offset....1134

13.237.2 Function........................................................................ 1135

13.237.3 Diagram........................................................................ 1135

13.237.4 Fields........................................................................ 1135

13.238 Variable Sequence Output Length Register; Upper 32 bits (UVSOL0 - UVSOL2)........................ 1136

13.238.1 Offset........................................................................ 1136

13.238.2 Function........................................................................ 1136

13.238.3 Diagram........................................................................ 1136

13.238.4 Fields........................................................................ 1137

13.239 Variable Sequence Input Length Register; Upper 32 bits (UVSIL0 - UVSIL2)........................ 1137

13.239.1 Offset........................................................................ 1137

13.239.2 Function........................................................................ 1137

13.239.3 Diagram........................................................................ 1137

13.239.4 Fields........................................................................ 1138

# Chapter 1
# Overview of SEC (security engine) functionality

SEC is the chip's cryptographic acceleration and offloading hardware. It combines functions previously implemented in separate modules to create a modular and scalable acceleration and assurance engine. It implements block encryption algorithms, stream cipher algorithms, hashing algorithms, public key algorithms, run-time integrity checking, and a hardware random number generator. SEC performs higher-level cryptographic operations than previous cryptographic accelerators. This provides significant improvement to system-level performance. SEC includes the following interfaces:

- Register interface for the processor to write configuration and command information, and to read status information
- DMA interface that allows SEC to read/write data from external memory
- Queue Manager interface that allows SEC to accept jobs directly from the Queue Manager module

- Job Queue Controller with 4 job rings
- 3 Descriptor Controllers (DECOs):
    - Responsible for executing descriptors and managing sequencing of keys, context, and data through the various CHAs
    - Responsible for performing header and trailer processing as defined by the descriptor
- Run-Time Integrity Checker (RTIC)
- Crypto Hardware Accelerators (CHAs)
    - Public Key Hardware Accelerator (PKHA)
    - A Random Number Generator (RNG)
    - 3 Advanced Encryption Standard Hardware Accelerators (AESA)
    - 3 Message Digest Hardware Accelerators (MDHA)
    - SNOW 3G f9 Hardware Accelerator (SNOW f9)
    - SNOW 3G f8 Hardware Accelerator (SNOW f8)
    - ZUC Encryption Hardware Accelerator (ZUCE)
    - ZUC Authentication Hardware Accelerator (ZUCA)
    - 3 Data Encryption Standard Hardware Accelerators (DESA)

- 3 Cyclic-Redundancy Check Hardware Accelerators (CRCA)
- Kasumi f8 and f9 Hardware Accelerator (KFHA)

This figure shows the block diagram for SEC.

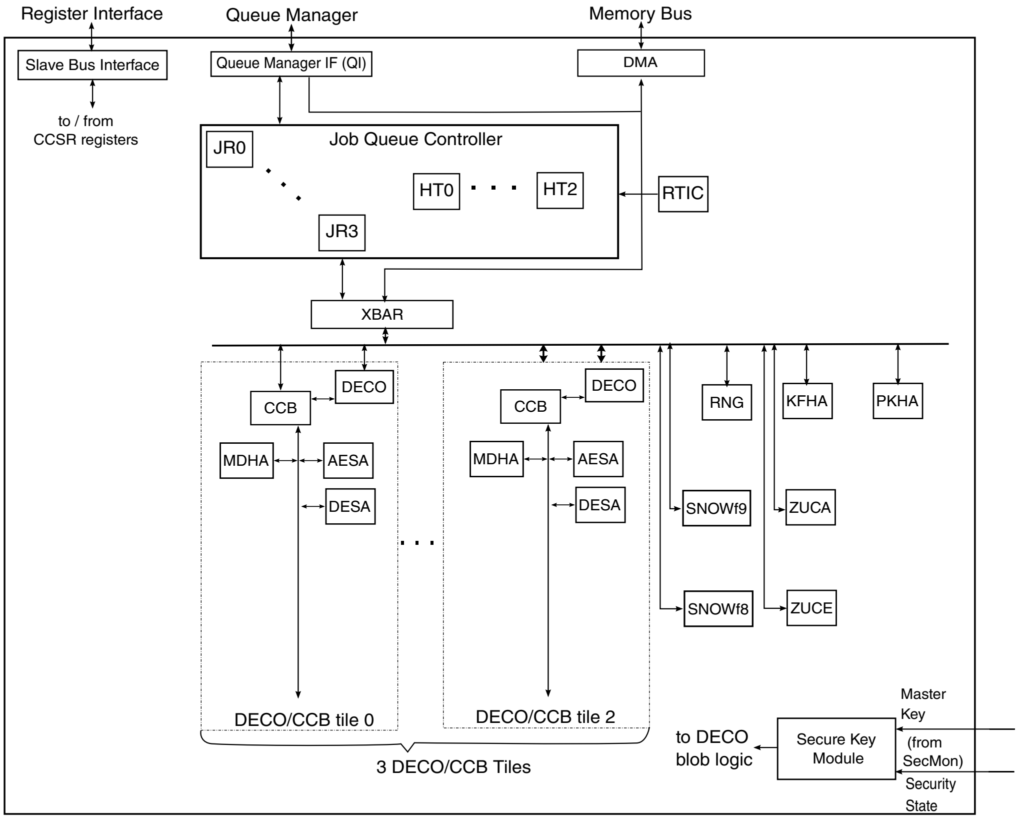


**Figure 1-1. SEC block diagram**

# Chapter 2
# Feature summary

SEC includes the following features:

- Offloading of cryptographic functions via a programmable job descriptor language
    - Job descriptors can contain multiple function commands.
    - Job descriptors can be chained to additional job descriptors.
    - Job descriptors can be submitted via 4 separate hardware-implemented job rings.
    - Job descriptors can be submitted via Data Path Acceleration Architecture (DPAA) Queue Manager portals.
- 32-bit register bus interface
- 1 128-bit DMA interface
    - Automatic byte, half-word, word and double-word ordering of data read/written
    - Scatter/gather support for data
- Special-purpose cryptographic keys
    - Black keys
        - Keys stored in memory in encrypted form and decrypted on-the-fly when used
        - AES_ECB or AES_CCM encryption using a 256-bit key
    - Export and import of cryptographic blobs
        - Data encapsulated in a cryptographic data structure for storage in non-volatile memory
        - AES_CCM encryption using a 256-bit key
        - Each blob encrypted using its own randomly generated blob key.
        - Blob key encrypted using a non-volatile blob key encryption key
        - Blob key encryption key derived from non-volatile master key input
        - Separate blob key encryption keys for trusted mode, secure mode, and non-secure mode
- Public key cryptography
    - Modular Arithmetic
        - Addition, subtraction, multiplication, exponentiation, reduction, inversion, greatest common denominator
        - Both integer and binary polynomial functions
        - Modulus size up to 4096 bits

        - Arithmetic operations performed with 32-bit-digit arithmetic unit
        - Timing-equalized and normal versions of modular exponentiation
    - DSA
        - DSA sign and verify
        - Verify with private key
        - DSA key generation
        - Non-timing-equalized versions of private-key operations
        - Timing-equalized version signing and key generation
        - Non-timing-equalized versions of sign and key generation
    - Diffie-Hellman
        - Diffie-Hellman (DH) key agreement
        - Key generation
        - Timing-equalized versions of key agreement and key generation
        - Non-timing-equalized versions of key agreement and key generation
    - RSA
        - Modulus size up to 4096 bits
        - Public and Private Key operations
        - Private keys in (n,d), (p,q,d), or 5-part (p,q,dp,dq,c) forms
        - Private Key operations (decrypt, sign) timing equalized to thwart side channel attack
        - Non-timing-equalized versions of private-key operations
    - Primality testing
        - Maximum size 4096 bits
    - Elliptic curve cryptography
        - Point add, point double, point multiply
        - Point validation (is point on curve)
        - Timing-equalized and normal versions point multiplication
        - Public Key validation
        - Both prime field and binary polynomial field functions
        - Elliptic curve digital signature algorithm (ECDSA) sign and verify
        - ECDSA verify with private key
        - Elliptic curve Diffie-Hellman key agreement
        - ECDSA and ECDH key generation
        - Modulus size up to 1024 bits
        - Timing-equalized versions of ECDSA sign and key generation
        - Non-timing-equalized versions of sign and key generation
- Cryptographic authentication
    - Hashing algorithms
        - MD5
        - SHA-1
        - SHA-224

        - SHA-256
        - SHA-384
        - SHA-512
        - SHA-512/224
        - SHA-512/256
    - Message authentication codes (MAC)
        - HMAC-all hashing algorithms
        - SSL 3.0 MAC (SMAC-MD5, SHA-1 only)
        - AES-CMAC
        - AES-XCBC-MAC
        - Kasumi f9
        - SNOW 3G f9
        - ZUC authentication
    - Auto padding
    - ICV checking
- Authenticated encryption algorithms
    - AES-CCM (counter with CBC-MAC)
    - AES-GCM (Galois counter mode)
- Symmetric key block ciphers
    - AES (128-bit, 192-bit or 256-bit keys)
    - DES (64-bit keys, including key parity)
    - 3DES (128-bit or 192-bit keys, including key parity)
    - Kasumi f8 with support for 3g, ECSD/EDGE, and GSM
    - SNOW 3G f8
    - ZUC encryption
    - Cipher modes
        - ECB, CBC, CFB, OFB for all block ciphers
        - CTR and XTS for AES
- Random-number generation
    - Entropy is generated via an independent free running ring oscillator
    - For lower-power consumption, oscillator is off when not generating entropy
    - NIST-compliant, pseudo random-number generator seeded using hardware-generated entropy
- Run-time integrity checking
    - SHA-256 message authentication
    - SHA-512 message authentication
    - Segmented data-gathering to support non-contiguous data blocks in memory
    - Support for up to four independent memory blocks
- Advanced protocol support
    - IPsec
    - SSL/TLS

- DTLS
    - SRTP
    - IEEE 802.11-2012 WPA2 MPDU for WiFi
    - IEEE 802.16 WiMAX
    - IEEE 802.1AE MacSec/LinkSec
    - Support for protocol-specific padding
    - 3GPP Release 11 (LTE) PDCP layer protocol
- Extensive virtualization features
    - Job rings can be time-shared by multiple security domains
    - Black keys are cryptographically separated per security domain
    - Blobs are cryptographically separated per security domain
    - Trusted descriptors are cryptographically separated per security domain

# Chapter 3
# SEC implementation

SEC provides platform assurance by working with security monitor (SecMon), which is a companion logic block that tracks the security state of the chip. SEC is programmed using SEC job descriptors (not to be confused with frame descriptors (FDs)) that indicate the operations to be performed and that point to the message and associated data. SEC incorporates a DMA engine to fetch the descriptors, read the message data, and write the results of the operations. The DMA engine provides a scatter/gather capability so that SEC can read and write data scattered in memory. SEC may be configured by means of software for dynamic changes in byte ordering. The default configuration for this version of SEC is big-endian mode.

## 3.1 SEC submodules

The SEC core contains the following submodules:

- Master bus interface
- Register bus interface
- Job queue controller (JQC)
- Queue Manager Interface (QI)
- Run-Time Integrity Checker (RTIC)
- Descriptor Controllers (DECOs)
- Cryptographic control blocks (CCBs)
- Multiple cryptographic hardware accelerators (CHAs)

JQC fetches descriptors that tell SEC which cryptographic operations to perform and on what data to operate. DECO decodes descriptors and executes the commands within them. For those descriptor commands that use CHAs, DECO communicates with the CHAs by means of the CCB.

## 3.2 Cryptographic engines implemented in SEC

The cryptographic engines provided are:

- Public key hardware accelerator (PKHA)
- Data encryption standard (DES) accelerator (DESA)
- Advanced encryption standard (AES) accelerator (AESA)
- Message digest (hashing) hardware accelerator (MDHA)
- Random-number generator (RNG)
- SNOW 3G f8 (SNOW encryption algorithm) Hardware Accelerator (SNOWf8)
- SNOW 3G f9 (SNOW authentication algorithm) Hardware Accelerator (SNOWf9)
- Cyclic redundancy check accelerator (CRCA)
- Kasumi f8 and f9 (Kasumi encryption and authentication) hardware accelerator (KFHA)
- ZUC encryption algorithm hardware accelerator (ZUCE)
- ZUC authentication hardware accelerator (ZUCA)

# Chapter 4
# SEC modes of operation

SEC can operate in the following security modes:

- Trusted
- Secure
- Non-secure
- Fail

SecMon controls these modes based on SecMon's current security state (that is, init, check, trusted, secure, non-secure, and fail). The primary difference between these modes is that they make different cryptographic keys available. Within each mode there are keys that are volatile (that is, a different key value is used for each power-on session) and keys that are non-volatile (that is, the same key value is available during each power-on session).

## 4.1 Security Monitor (SecMon) security states

The current security mode can be identified in the SEC status register's MOO field.

This figure shows an overview of the SecMon security state transitions. Note that SEC can detect certain security alarm conditions and can signal an alarm to SecMon. Depending upon the settings and configuration of SecMon, this may cause the SecMon security state machine to transition to fail state, which would then put SEC into its fail mode.

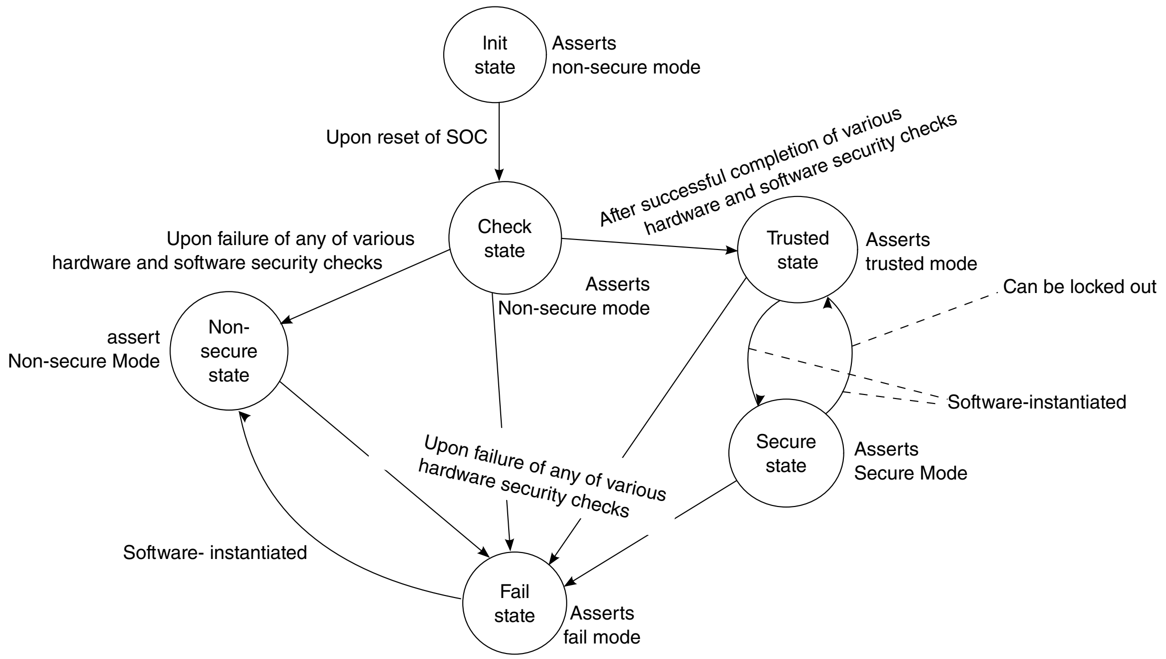


**Figure 4-1. SecMon security state machine diagram**

## 4.1.1 The effect of security state on volatile keys

SEC implements three 256-bit volatile cryptographic keys. At each power up, boot code must test and instantiate the RNG. After instantiation, (or as part of RNG instantiation), the three volatile secret keys must be generated. These values are stored within secure key registers in SEC. The values are zeroized when SEC transitions to fail mode (in other words, when the SecMon's security state machine transitions to fail state).

The available volatile keys, (which are located in SEC's secure key module), are as follows:

- Job descriptor key encryption key (JDKEK) - used by job descriptors for encrypting black keys (encrypted keys)
- Trusted descriptor key encryption key (TDKEK) - used by trusted descriptors for encrypting black keys
- Trusted descriptor signing key (TDSK) - used to authenticate trusted descriptors (digitally signed job descriptors)

Note that the JDKEK, the TDKEK, and the TDSK are all available for use by SEC in trusted mode, secure mode, and non-secure mode[1], but this does not cause any security issue. The reason that this is not a security issue is that the trusted mode and secure mode

[1] The JDKEK, TDKEK, and TDSK are readable and writable while in non-secure mode to facilitate hardware testing.

are intended to use the same values for these keys, and these key values will be different when in non-secure mode (which is not allowed to obtain the trusted/secure state mode values of these keys). The reason that SEC cannot obtain the trusted/secure state mode values of these keys when in non-secure mode is that new values for these keys are generated by SEC's hardware RNG at each POR, and these keys are zeroized when entering fail mode. As shown in Figure 4-1, the only paths from trusted state or secure state to non-secure state pass through fail state or through a hardware reset, and in each case the keys will be cleared. The only path from non-secure state to either trusted state or secure state is through a hardware reset, which clears the keys. Consequently, when operating in non-secure mode SEC does not have access to previous trusted mode/secure mode values of these keys.

### 4.1.2 The effect of security state on non-volatile keys

Data that must be retained when the system is powered off must be stored in external non-volatile storage. Some of this data is disclosure-sensitive (such as data rights management keys) and must be protected even when the system is powered off. SEC implements non-volatile cryptographic keys that can be used to encrypt sensitive data during one power-on cycle, and then decrypt it during a subsequent power-on cycle. These non-volatile keys (blob key encryption keys) are derived from the master key input that SEC receives from SecMon.

When SEC is operating in trusted mode or secure mode, SEC derives blob key encryption keys (BKEKs) from its master key input. When SEC is operating in non-secure mode or fail mode, BKEKs are derived from the non-volatile test key, a hardwired constant used for known-answer testing.

## 4.2 Keys available in different security modes

The primary difference between SEC's security modes is that different cryptographic keys available are available in the different modes. See each mode's section for the description of the mode's special keys.

### 4.2.1 Keys available in trusted mode

While in trusted mode, SEC can use special keys as listed in this table.

Table 4-1. Special keys used in trusted mode

| Key | Characteristic(s) | Function(s) |
|---|---|---|
| Job descriptor key encryption key | • At POR, a new value (shared with secure mode but not shared with non-secure mode) should be generated from the RNG after instantiation<br>• Zeroized when entering fail mode | Used for automatic key encryption and decryption when executing Job Descriptors, Trusted Descriptors and Shared Descriptors |
| Trusted descriptor key encryption key | | Can be used for automatic key encryption and decryption when executing trusted descriptors, including shared descriptors referenced by trusted descriptors. |
| Trusted descriptor signing key | | Used for signing, verifying and re-signing Trusted Descriptors |
| Master key derivation key | Non-volatile, shared with secure mode, but uses a different key derivation function input to generate keys not shared with trusted mode, non-secure mode or fail mode | Used for blob encapsulation or decapsulation operations |

## 4.2.2 Keys available in secure mode

While in secure mode, SEC can use special keys as listed in this table.

Table 4-2. Special keys used in secure mode

| Key | Characteristic(s) | Function(s) |
|---|---|---|
| Job descriptor key encryption key | • At POR a new value (shared with trusted mode but not shared with non-secure mode) should be generated from the RNG after instantiation<br>• Zeroized when entering fail mode | Used for automatic key encryption and decryption when executing job descriptors, trusted descriptors and shared descriptors |
| Trusted descriptor key encryption key | | Can be used for automatic key encryption and decryption when executing trusted descriptors, including shared descriptors referenced by trusted descriptors |
| Trusted descriptor signing key | | Used for signing, verifying and re-signing Trusted Descriptors |
| Master key derivation key | Non-volatile, shared with trusted mode, but uses a different key derivation function input to generate keys not shared with trusted mode, non-secure mode or fail mode | Used for blob encapsulation or decapsulation operations |

## 4.2.3 Keys available in non-secure mode

In non-secure mode a fixed default key with a known value is used in place of the master key derivation key. This allows the cryptographic blob mechanism to be tested using known test results. The volatile key registers are read and write accessible until they are locked, which allows testing using known test results. While in non-secure mode SEC can use special keys as listed below.

Table 4-3. Special keys used in non-secure mode

| Key | Characteristic(s) | Function(s) |
|---|---|---|
| Job descriptor key encryption key | • At POR, a new value (not shared with trusted mode or secure mode) should be generated from the RNG after instantiation<br>• Zeroized when entering fail mode | • Can be read and overwritten for testing<br>• Used for automatic key encryption and decryption when executing job descriptors, trusted descriptors, and shared descriptors |
| Trusted descriptor key encryption key | | • Can be read and overwritten for testing<br>• Can be used for automatic key encryption and decryption when executing trusted descriptors, including shared descriptors referenced by a trusted descriptors |
| Trusted descriptor signing key | | • Can be read and overwritten for testing<br>• Used for testing the signing, verifying and re-signing of trusted descriptors |
| Master key derivation key | Non-volatile, fixed, and not shared with trusted mode or secure mode | Used for testing blob encapsulation or decapsulation operations |

## 4.2.4 Keys available in fail mode

When SEC transitions to fail mode, SEC clears all registers that could potentially hold sensitive data[2]. Because of this, cryptographic operations that are in progress when the transition occurs will likely not produce the correct result. If this is the case, the operation completes with an error indication.

Although SEC cleans up ongoing operations after a transition to fail mode, SEC is not intended to continue operating in this mode. After removing all causes for entering the fail mode, software can initiate a transition from fail mode to non-secure mode by commanding the SecMon security state machine to transition from fail state to non-secure state (unless this transition has previously been locked out via software). However, since all key registers were cleared when SEC entered fail mode, the only useful actions that can be performed after the transition to non-secure mode would be those required to investigate the cause of the transition to Fail mode.

Note that it is not possible to transition from fail mode back to secure mode or trusted mode.

[2] The registers that are cleared include the class 1 and class 2 key registers, the class 1 and class 2 context registers, the math registers, the JDKEK, TDKEK and TDSK registers, the PKHA E memory, the input data FIFO, the output data FIFO, and the descriptor buffer.

# Chapter 5
# SEC hardware functional description

As shown in Figure 1-1, SEC functionality is aligned with several major subcomponents. This table describes these subcomponents.

Table 5-1. SEC subcomponents

| Description | Cross-reference(s) |
|---|---|
| **Interfaces** | |
| Register interface<br>Used for access to configuration, control, status and debugging registers | Register interface (IP bus) |
| Job execution interfaces | |
| Job Ring Interface (JR) | Job Ring interface |
| Queue Manager Interface (QI) | Queue Manager Interface (QI) |
| **Job Queue Controller** | |
| Schedules tasks for the descriptor processor | Job scheduling |
| **Descriptor processor** | |
| Descriptor controller (DECO) | Descriptors and descriptor commands |
| Cryptographic control block (CCB) | Descriptor controller (DECO) and cryptographic control block (CCB) |
| **Cryptographic hardware accelerators (CHAs)** | |
| Public key hardware accelerator (PKHA) | Public-key hardware accelerator (PKHA) functionality |
| Kasumi f8 and f9 hardware accelerator (KFHA) | Kasumi f8 and f9 hardware accelerator(KFHA) functionality |
| DES and 3DES hardware accelerator (DESA) | Data encryption standard accelerator (DES) functionality |
| CRC hardware accelerator (CRCA) | Cyclic-redundancy check accelerator (CRCA) functionality |
| Random Number Generator (RNG) | Random-number generator (RNG) functionality |
| SNOW f8 Hardware Accelerator (SNOWf8) | SNOW 3G f8 accelerator functionality |
| SNOW f9 Hardware Accelerator (SNOWf9 | SNOW 3G f9 accelerator functionality |
| Message Digest Hardware Accelerator (MDHA) | Message digest hardware accelerator (MDHA) functionality |
| AES Hardware Acclerator (AESA) | AES accelerator (AESA) functionality |
| ZUC encryption hardware accelerator (ZUCE) | ZUC encryption accelerator (ZUCE) functionality |
| ZUC authentication hardware accelerator (ZUCA) | ZUC authentication accelerator (ZUCA) functionality |
| **Trust Architecture modules** | |
| Run-time integrity checker (RTIC) | Run-time integrity checker (RTIC) |

*Table continues on the next page...*

Table 5-1. SEC subcomponents (continued)

| Description | Cross-reference(s) |
|---|---|
| Secure key module | Black keys<br>Blobs<br>Trusted descriptors |

## 5.1 System Bus Interfaces

SEC is connected to a SoC-wide bus for access to SEC registers. See Register interface (IP bus). An AXI master interface connects to the SoC bus fabric for DMA access to system memory.

### 5.1.1 AXI master (DMA) interface

DMA access to system memory is implemented through an AXI master interface. SEC DMA always asserts normal (AKA user) mode rather than privileged (AKA supervisor) mode, and always asserts data access rather than instruction access (i.e. fetch). SEC DMA can be configured to assert either TrustZone SecureWorld or TrustZone NonSecureWorld for different bus transactions. SEC DMA can be configured to assert specified ICID values for various bus transactions. For high throughput this interface utilizes a 128-bit data bus.

The AXI master interface configuration defaults are chosen to enhance performance where possible, however ideal configuration for performance is not the default and should not be assumed for any application. The DMA reads and writes data in data-bus-aligned bursts, whenever possible. The LARGE_BURST field default value is '0' but better performance will be achieved with a value of '1' (See the Master Configuration Register (MCFGR)) Other notable, performance enhancements include the use of read-safe, write-safe, and write-efficient transactions, described in the following sections.

#### 5.1.1.1 DMA read-safe transactions

A read-safe transaction may read data preceding and/or following the target bytes to align the starting and ending byte addresses to data bus or burst address boundaries, even if not all of the data read is needed. This improves performance because the bus system is optimized for address-aligned transactions.

**NOTE**

When reading hardware registers, it is not always safe to read extra bytes beyond the limits of the register due to potential side effects. For example, a register at an adjacent address might automatically clear itself when its contents are read. Although SEC's DMA would not normally be directed to read hardware registers, read-safe operation can be disabled if necessary (see the RSE field in the DMA Control Register).

#### 5.1.1.2 DMA interface write-safe transactions

A write-safe transaction is similar to a read-safe transaction in that it accesses a set of memory locations even if not all of those locations need to be accessed. In the case of write-safe transactions, SEC writes zeros to addresses past the targeted locations up to the next data bus or burst address boundary, depending upon the offset within the cache line. This improves performance because the bus system is optimized for cacheline-aligned transactions.

SEC's DMA uses write-safe transactions only when the following conditions are all met:

- A SEQ STORE or SEQ FIFO STORE command is being executed.
- A SEQ OUT PTR command defined the sequence.
- The EWS (Enable Write-Safe) bit was set in the SEQ OUT PTR command.
- The total number of bytes written does not exceed the available buffer space.

Note that even when performing a write-safe transaction, SEC's DMA does not write more bytes than was specified by the LENGTH field of the SEQ OUT PTR command (plus any extensions to the length specified by SEQ OUT PTR commands with the PRE bit set). If a SEQ OUT PTR command with the REW (Rewind) field set to 10b or 11b has been executed, write-safe transactions are not performed unless output length counting has been re-enabled via the DECO Control Register. Note that write-safe transactions can be disabled if necessary (see WSE field in the DMA Control Register).

#### 5.1.1.3 DMA write-efficient transactions

In addition to “write-safe” transactions, the DMA interface also allows “write-efficient” transactions to be used for updating Descriptors in memory. Most of the built-in protocols utilize a Protocol Data Block (PDB) that is located within the Job Descriptor or Shared Descriptor that contains the PROTOCOL OPERATION command. The PDB specifies various options, but may also contain protocol state information (such as sequence numbers) that must be updated for each protocol data unit that is processed. Any updates that were made to portions of the PDB while the Descriptor was executing must be written back to the Descriptor in memory once the Descriptor terminates so that the new state information is available for subsequent executions of the same Descriptor. These write-back operations are more efficient if the write transaction is aligned to data bus and burst address boundaries. Write-efficient bus transactions (specified by STORE command SRC values 45h and 46h) will write out more of the Descriptor Buffer than just the updated portions of the Descriptor if this is both possible and more efficient. Extending (either toward the beginning or toward the end, or both) the portion of the Descriptor Buffer to be written out is considered possible if the extended portion contains only the Shared Descriptor or only the Job Descriptor that must be updated. Extending the portion of the Descriptor Buffer to be written out is considered efficient if this causes the write transaction to be aligned to address boundaries that minimizes the number of bus transactions. If extending the portion to be written is not possible or not efficient, then the bus transactions associated with STORE commands using SRC values 45h and 46h will be identical to bus transactions associated with SRC values 41h and 42h.

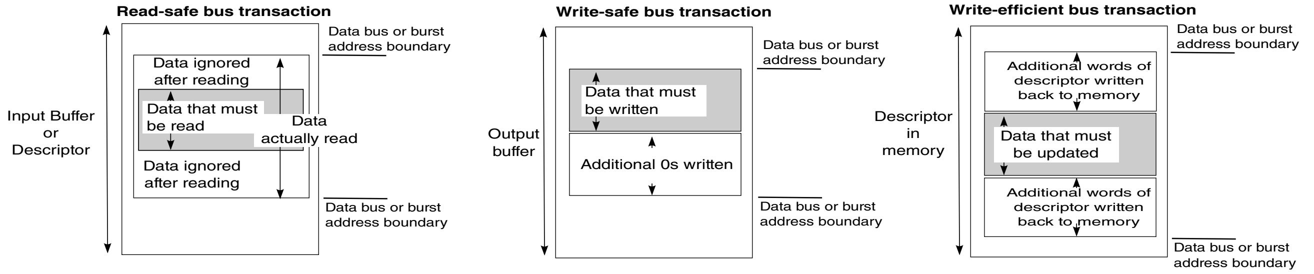

Read-safe bus transaction
Input Buffer or Descriptor
Data bus or burst address boundary
Data ignored after reading
Data that must be read
Data actually read
Data ignored after reading
Data bus or burst address boundary
Write-safe bus transaction
Output buffer
Data bus or burst address boundary
Data that must be written
Additional 0s written
Data bus or burst address boundary
Write-efficient bus transaction
Descriptor in memory
Data bus or burst address boundary
Additional words of descriptor written back to memory
Data that must be updated
Additional words of descriptor written back to memory
Data bus or burst address boundary

**Figure 5-1. Read-Safe, Write-Safe and Write-Efficient Bus Transactions**

### 5.1.1.4 DMA bursts that may read past the end of data structures

SEC DMA accesses do not read a full burst if the read would need to cross a 4 Kbyte address boundary. SEC also does not read a full burst from a job ring input or output ring if it would need to read past the end of the ring. However, as illustrated in the figure below, SEC may read past the end of a descriptor or scatter/gather table (SGT) when fetching them because it does not know the length of the descriptor or SGT before issuing the read transaction.

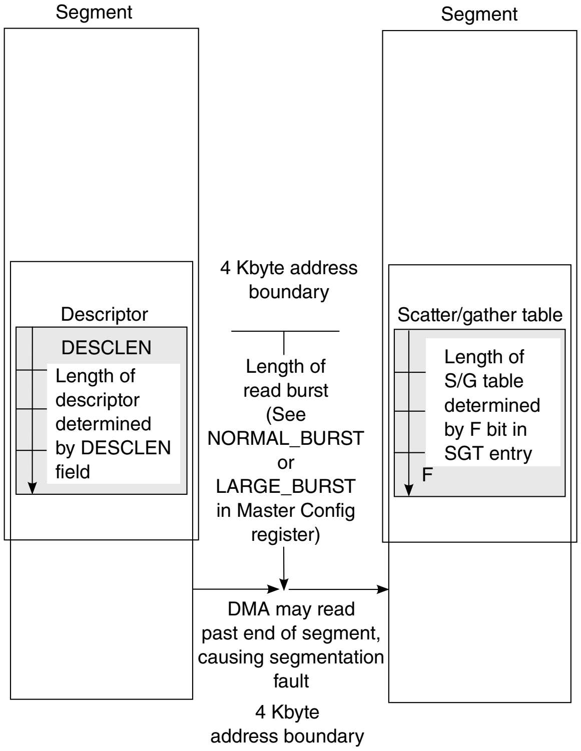

Segment
Segment
4 Kbyte address boundary
Descriptor
DESCLEN
Length of descriptor determined by DESCLEN field
Length of read burst (See NORMAL_BURST or LARGE_BURST in Master Config register)
Scatter/gather table
Length of S/G table determined by F bit in SGT entry
F
DMA may read past end of segment, causing segmentation fault
4 Kbyte address boundary

**Figure 5-2. DMA may read past end of descriptor or SGT**

## 5.1.2 Register interface (IP bus)

SEC's register interface (32-bit IP bus) is used to read and write registers within SEC for the following purposes:

**Table 5-2. Summary of register interface uses**

| Purpose | For more information, see |
|---|---|
| During chip initialization time | |
| To configure SEC, including initialization of the Job Rings and Queue Manager Interface. | • Initializing job rings<br>• Initializing the Queue Manager Interface |
| Change the default settings for SEC's AXI DMA interface | Master Configuration Register (MCFGR) in the memory map |
| Configure RTIC | Initializing RTIC |
| During normal steady-state operations | |
| Manage SEC's job ring interface | Job Ring interface |
| During hardware and software debugging | |
| Read status registers and single-step descriptor commands | • Register-based service interface<br>• SEC Status Register<br>• RNG TRNG Status Register<br>• RNG DRNG Status Register<br>• Holding Tank Status Register<br>• Secure Memory Status Register |

Table 5-2. Summary of register interface uses

| Purpose | For more information, see |
|---|---|
| | • Job Ring Output Status Register<br>• Job Ring Interrupt Status Register<br>• CCB Interrupt Status Register<br>• CCB Status and Error Register<br>• CCB FIFO Status<br>• DECO Operation Status Register |

**NOTE**

Accesses to registers other than the DECO and CCB registers must use full-word (32-bit) reads or writes. Reads and writes to the DECO and CCB registers permit byte access.

## 5.2 SEC service interface concepts

SEC delivers cryptographic services through a set of interfaces optimized for different use models (see Service interfaces). All service interfaces share a number of common objects and concepts, which are explained in more detail in the subsections below.

### 5.2.1 SEC descriptors

SEC provides cryptographic services by executing a series of commands specified in SEC descriptors. Each SEC descriptor is formed from SEC commands and embedded data. The set of available commands includes conditional branches, loops, subroutine calls, or jumps to other descriptors, as well as mathematical, cryptographic and data move operations. Except as specified by the branch and call commands, the commands within SEC descriptors normally execute in sequence until the descriptor has completed (or has been aborted due to external management action). Descriptors cannot change their own execution priority or directly affect the scheduling of other descriptors, but SEC descriptors do have mechanisms to ensure coherency of data shared between descriptors.

SEC descriptors access input, output, and control data by means of memory addresses and job-specific memory access control attributes, or, for jobs submitted via job rings, job-ring-specific memory access control attributes. Descriptors cannot change their memory access control attributes, but they can dynamically select 1 or 2 attribute sets from predefined configurations. SEC descriptors also cannot allocate or release buffers to/from buffer pools.

SEC implements different types of descriptors to address specific processing needs:

- Job Descriptor (JD) (see Job descriptors)

  Every SEC job is defined by at least one JD. The JD may be provided by the SEC service user directly via the register-based service interface or via the job ring-based service interface, or the JD may be created internally within SEC in response to a service request from the Queue Manager Interface (QI) or the Run-Time Integrity Checker (RTIC). It is also possible for a JD to invoke a Shared Descriptor (SD) or to jump to another JD, which allows a job to consist of an arbitrarily large number of commands and data objects.

- Shared Descriptor (SD) (see Shared descriptors)

  SDs provide a mechanism to group and reuse instructions and data that are common in the processing of more than one related job, e.g., processing protocol data units of a network connection. A key feature of SDs is to select and coordinate sharing of the descriptor information between multiple DECOs. Using SDs may also increase performance by improving the probability of finding an SD within SEC that has already been read for a preceding job requiring the same processing.

- Trusted Descriptor (TD) (see Trusted descriptors)

  TDs are essentially the same as JDs, but they are cryptographically signed. When a TD is presented for execution, SEC first checks the signature and executes the TD only if the signature is correct. TDs are intended to ensure that special access privileges are usable only by descriptors that are known to employ those privileges properly. TDs would be created by trusted software (such as secure boot software or a hypervisor), and then cryptographically signed to ensure that they were not altered by untrusted software.

- Inline Job Descriptor (IJD) (see Using in-line descriptors)

  IJDs are simply JDs that are made available to SEC through the input data stream. JDs submitted via job rings may direct SEC to execute commands from an IJD at any time using the SEQ IN PTR command with the INL option selected. QI-generated JDs imply the service user's intent to execute an IJD from the absence of an SD specification (SD length is set to 0). In this case SEC is directed to execute commands from an IJD starting at the first byte of the input data stream and immediately after output and input data stream addresses are defined in the internally generated JD. IJDs are primarily intended to simplify the processing of one-off jobs and job variations.

- Replacement Job Descriptor (RJD and CRJD) (see Using replacement job descriptors)

RJDs and Control RJDs (CRJDs) are intended to support job processing variations or updates of immediate or state data defined in SDs. Both kind of RJDs replace the JD that invoked them and can be executed either before or after the execution of the SD. Thus RJDs and CRJDs provide the capability to permanently update or temporarily change the processing defined by SDs. RJDs may be supplied with 2 methods: The normal RJD is supplied inline (like the IJD) embedded in the input data stream. Alternatively, a CRJD associated with a specific SD may be utilized. A CRJD must be located in memory immediately following the SD. The execution of an RJD is initiated with the SEQ IN PTR command by setting the RJD control bit. A CRJD requires to additionally set the CTRL bit. Note, first generation DPAA SoC products, such as this one, cannot initiate the execution of a CRJD through QI; only jobs submitted through job rings can.

### 5.2.2 Job termination status/error codes

SEC reports the termination status of all jobs, allowing software to determine whether the descriptor completed normally, with warnings, or with an error. The reporting mechanism always involves writing a job termination status word to memory. Depending on the selected service interface, SEC may also update service interface status registers.

An all-zero status indicates that the job completed without warnings or errors. If a warning or an error was encountered, the code in the source field of the status word indicates which component within SEC detected the condition. The remaining status word coding provides additional component-specific detail.

For jobs submitted through the Job Ring interface the job termination status is written to the Job Ring Output Status register and to the output ring in the word following the pointer to the completed job descriptor. The job termination status can be read from the Job Ring Output Status register, but because the termination status of each newly completed job will overwrite the previous job's termination status this mechanism is primarily intended to support debug and ring management (for executing single jobs or after the ring is halted). A selection of severe error conditions (potentially indicating malicious users or software instability) is stored together with additional error and/or access violation attributes in job-ring specific recoverable error record registers.

For jobs submitted through the Queue Manager Interface (QI) the job completion status is written to the STATUS/CMD field in the response/output frame descriptor returned to the user. If QI detects an error, the status word's source field is set to 5. QI-detected error conditions are presented in 1-hot encoded format. This information is also latched in the QI Status register. Software can read this register and obtain the accumulated status of all jobs processed since the last time the QI Status register was read.

## Table 5-3. Job termination status word

| bits 31-28 | bits 27-0 | | | | | |
|---|---|---|---|---|---|---|
| Source | Source-specific error or warning codes | | | | | |
| 0h<br>(None) | 0000000h - No errors or warnings | | | | | |
| **bits 31-28** | **bit 27** | **bit 26** | **bits 25-16** | **bits 15-8** | **bits 7-4** | **bits 3-0** |
| Source | (JMP) | (MLK) | Reserved | (DESC INDEX) | (CHAID) | (ERRID) |
| 2h<br>(CCB) | See footnote [1] | See footnote [2] | 0000h | The number of words from the start of the descriptor where the error was detected. In some cases this value may be off by one or more words due to timing issues. | 0h - CCB<br>1h - AESA (all modes of AES)<br>2h - DESA (DES and 3DES)<br>4h - MDHA (MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256)<br>5h - RNG<br>6h - SNOWf8 (SNOW encrypt/decrypt)<br>7h - KFHA f8/9 Kasumi encrypt/decrypt and authentication)<br>8h - PKHA (all public key operations)<br>9h - CRCA (CRC processing)<br>Ah - SNOWf9 (SNOW authentication)<br>Bh - ZUCE (ZUC encrypt/ decrypt)<br>Ch - ZUCA (ZUC authentication) | 0h - No error<br>1h - Mode error<br>2h - Data size error<br>3h - Key size error<br>3h - (RNG) Instantiate error<br>4h - (RNG) Not instantiated error<br>4h - (PKHA) A size error<br>5h - (RNG) Test instantiate error<br>5h - (PKHA) B size error<br>6h - (RNG) Prediction resistance error<br>6h - Data out-of-sequence error<br>6h - (PKHA) "c" is zero for ECC F2M<br>7h - (RNG) Prediction resistance & test request error<br>7h - (PKHA) Divide by 0 error<br>8h - (PKHA) Modulus even error<br>9h - (DES) key parity error<br>9h - (RNG) Secure Key Generation error<br>Ah - ICV check failed<br>Bh - Hardware error<br>Ch - (AESA) CCM AAD size error<br>Ch - (RNG) Continuous check error<br>Ch - Invalid key write |

*Table continues on the next page...*

### Table 5-3. Job termination status word (continued)

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | Dh - (CCB) Class 1 or class 2 CHA is not reset, or, a second CHA of the same class is selected prior to resetting the first selection<br>Eh - (CCB) Invalid CHA combination selected<br>Fh - (CCB) Invalid CHA |
| **bits 31-28**<br>Source | **bit 27**<br>(JMP) | **bit 26**<br>(MLK) | **bits 25-16**<br>Reserved | **bits 15-8**<br>(DESC INDEX) | **bits 7-0**<br>User-defined value | |
| 3h<br>(Jump Halt User Status) | See footnote [1] | See footnote [2] | 0000h | The number of words from the start of the descriptor where the JUMP HALT Command was encountered. | The value in the LOCAL OFFSET field of the JUMP command is written into these bits of the termination status word. The user is free to assign any interpretation to these bits, such as using them to distinguish among different instances of the JUMP command. | |
| **bits 31-28**<br>Source | **bit 27**<br>(JMP) | **bit 26**<br>(MLK) | **bits 25-16**<br>Reserved | **bits 15-8**<br>(DESC INDEX) | **bits 7-0**<br>Error Code | |
| 4h<br>(DECO) | See footnote [1] | See footnote [2] | 0000h | The number of words from the start of the descriptor where the error was detected. In some cases this value may be off by one or more words due to timing issues. | 00h - No error<br>01h - SGT length error (The descriptor is trying to read more data than is contained in the SGT table.)<br>02h - Unused SGT entry error (Extension bit set in unused SGT entry.)<br>03h - Job Ring Control Error (There is a bad value in the Job Ring Control register.)<br>04h - Invalid Descriptor Command<br>06h - Invalid KEY Command<br>07h - Invalid LOAD Command<br>08h - Invalid STORE Command<br>09h - Invalid OPERATION Command<br>0Ah - Invalid FIFO LOAD Command<br>0Bh - Invalid FIFO STORE Command<br>0Ch - Invalid MOVE/MOVE_LEN Command<br>0Dh - Invalid JUMP Command (a non-local JUMP Command is invalid because the target is not a Job Header Command, or the jump is from a TD to a JD, or because the target descriptor contains an SD)<br>0Eh - Invalid MATH Command<br>0Fh - Invalid SIGNATURE Command<br>10h - Invalid Sequence Command (A SEQ IN PTR or SEQ OUT PTR Command is invalid or a SEQ KEY, SEQ LOAD, SEQ FIFO LOAD, or SEQ FIFO STORE decremented the input or Output Sequence length below 0. This error may result if a built-in PROTOCOL Command has encountered a malformed PDU.) | |

*Table continues on the next page...*

## Table 5-3. Job termination status word

| | | | | | |
|---|---|---|---|---|---|
| | | | | | 11h - Skip data type invalid (The type must be Eh or Fh.) |
| | | | | | 12h - Shared Descriptor Header Error |
| | | | | | 13h - Header Error (Invalid length or parity, or certain other problems.) |
| | | | | | 14h - Burster Error (Burster has gotten into an illegal state.) |
| | | | | | 15h: Context Register Length Error. The descriptor is trying to read or write past the end of the Context Register. A SEQ LOAD or SEQ STORE with the VLF bit set was executed with too large a length in the variable length register (VSOL for SEQ STORE or VSIL for SEQ LOAD). |
| | | | | | 16h - DMA Error |
| | | | | | 1Ah - Job failed due to job ring reset or RTIC error |
| | | | | | 1Bh - Job failed due to transition to Fail Mode |
| | | | | | 1Ch - DECO Watchdog timer timeout error |
| | | | | | 1Dh - Error when copying key from another DECO (other DECO's key registers were locked) |
| | | | | | 1Eh - Error when copying data from another DECO (other DECO had unmasked descriptor error) |
| | | | | | 1Fh - ICID mismatch error (DECO was trying to share from itself or from another DECO but the two Non-SEQ ICID values didn't match or the "shared from" DECO's Descriptor required that the SEQ ICID and TZ/SDID values be the same but they aren't.) |
| | | | | | 20h - DECO has completed a reset initiated via the DRR register |
| | | | | | 21h - Nonce error (When using EKT (CCM) key encryption option in the FIFO STORE Command, the Nonce counter reached its maximum value and this encryption mode can no longer be used.) |
| | | | | | 22h - Leading meta data is too large for rewind operation |
| | | | | | 23h - Read Input Frame error (A read input frame was attempted, but the protocol executed does not support it or a SEQ IN PTR command has not been executed.) |
| | | | | | 24h - JDKEK, TDKEK or TDSK was needed, but value has not yet been initialized. |
| | | | | | 25h - Error while prefetching |
| | | | | | 26h - A job has DECO select value for a different DECO |
| | | | | | 27h - Rewind input frame attempted but input buffers already released |
| | | | | | 30h - DWORD load error |
| | | | | | 31h - DWORD store error |
| | | | | | 32h - Invalid PKCURVE Command |
| | | | | | 33h - Burster buffer reuse error (address or length went negative) |

*Table continues on the next page...*

### Table 5-3. Job termination status word (continued)

| | | | | | |
|---|---|---|---|---|---|
| | | | | | 80h - DNR (do not run) error (A Job Descriptor or Shared Descriptor had the DNR bit set.) |
| | | | | | 81h - undefined protocol command |
| | | | | | 82h - invalid setting in PDB |
| | | | | | 83h - Anti-replay LATE error |
| | | | | | 84h - Anti-replay REPLAY error |
| | | | | | 85h - Sequence number overflow |
| | | | | | 86h - Invalid signature |
| | | | | | 87h - DSA Sign Illegal test descriptor |
| | | | | | 88h - Protocol Format Error (A protocol has seen an error in the format of data received. When running RSA, this means that formatting with random padding was used, and did not follow the form: 00h, 02h, 8-to-N bytes of non-zero pad, 00h, F data.) |
| | | | | | 89h - Protocol size error |
| | | | | | 8Ah - Key not written before start of protocol |
| | | | | | 8Bh - IPsec decap CE DROP (ECN issue) error |
| | | | | | 8Ch - RFKG P & Q upper 100 bits the same |
| | | | | | 8Dh - RFKG computed D too small |
| | | | | | 8Eh - RFKG PDB and computed N sizes differ |
| | | | | | C1h - Undefined Blob mode |
| | | | | | C4h - Black Blob key or input size error |
| | | | | | C5h - Invalid key destination in blob command |
| | | | | | C8h - Trusted/Secure mode error in blob command |
| | | | | | CAh - LTE C-plane ICV error |
| | | | | | F0h - Warning: Descriptor completed normally, but IPsec TTL or hop limit field either came in as 0, or was decremented to 0 |
| | | | | | F1h - Warning: Descriptor completed normally, but HFN matches or exceeds the Threshold |
| | | | | | F2h - Warning: IPsec padding check error found |
| | | | | | FFh - Warning: Output frame length rollover |
| **bits 31-28**<br>Source | **bits 27-9**<br>Reserved | | | | **bits 8-0**<br>Error Code |
| 5h<br>(QI) | 0000h | | | | Bit 8 - TBTSERR: Table buffer too small error |
| | | | | | Bit 7 - TBPDERR: Table buffer pool depletion error |
| | | | | | Bit 6 - OFTLERR: Output Frame too large error |
| | | | | | Bit 5 - CFWRERR: Compound Frame write error |
| | | | | | Bit 4 - BTSERR: Buffer too small error |
| | | | | | Bit 3 - BPDERR: Buffer pool depletion error |
| | | | | | Bit 2 - OFWRERR: Output Frame write error |

*Table continues on the next page...*

**Table 5-3. Job termination status word (continued)**

| | | | | | |
|---|---|---|---|---|---|
| | | | | | Bit 1 - CFRDERR: Compound Frame read error<br>Bit 0 - PHRDERR: Preheader read error |
| **bits 31-28**<br>Source | **bits 27-12**<br>Reserved | | | **bits 11-8**<br>NADDR | **bits 7-0**<br>Error code |
| 6h<br>(Job Ring) | 0000h | | | Number of descriptor addresses requested for error code 1Eh, otherwise 0h | 00h - No error<br>1Eh - Error reading the Descriptor address<br>1Fh - Error reading the Descriptor |
| **bits 31-28**<br>Source | **bit 27**<br>(JMP) | **bit 26**<br>(MLK) | **bits 25-16**<br>Reserved | **bits 15-8**<br>(DESC INDEX) | **bits 7-0**<br>(COND) |
| 7h<br>(Jump Halt with Condition Codes) | See footnote [1] | See footnote [2] | 0000h | The number of words from the start of the descriptor where the JUMP HALT Command was encountered. | PKHA/Math condition codes field from JUMP HALT Command. |

1. If JMP = 1, the descriptor made a jump to another descriptor. When this bit is 1, the DESC INDEX field will contain the index into the descriptor that was jumped to rather than into the original descriptor.
2. Memory Leak. If MLK=1 a memory leak has occurred due to an error that prevents user requested buffer releases, i.e., this condition can only be detected for jobs generated by QI.

## 5.2.3 Frames and flows

SEC borrows the term ‘frame’ from network protocols. A frame simply refers to some number of sequential bytes that are usually, but not necessarily, part of a segmented byte stream and delimited by implicit or explicit start and end markers. Explicit markers are formed by so-called protocol headers and/or trailers of protocol-specific length including a possible length of 0. Implicit markers are out-of-band information defining where frame data starts and ends. For SEC the meaning of a frame is generalized to also include designated space into which SEC-generated frame data can be stored, as well as data that is completely unrelated to networking protocols, e.g., a piece of program code that needs to be cryptographically signed or a sequence of SEC-generated random data.

While frames define a logical sequence of bytes, the frame data itself does not need to be necessarily stored in a single, contiguous region of memory (also referred to as a buffer). Segmented, multi-buffer frames can be formed by utilizing scatter/gather tables (stored in additional buffers) where table entries are used to keep track of frame segment address, offset, length, and other segment attributes. For details see Scatter/gather tables (SGTs).

A SEC flow simply refers to a sequence of two or more frames requiring the same kind of processing. Whether the frame data is stored in single buffer frames, multi-buffer frames, or a mix of both is irrelevant. The key criteria of a flow is that all frames are processed in the same fashion. Some flows require that the frames be processed in a specific order. By using shared descriptors, SEC can control the order in which frames are processed and delivered to the service user by selecting an appropriate sharing type. For details see Shared descriptors.

### 5.2.4 Frame descriptors and frames

A Frame Descriptor (FD) is a standardized Data Path Acceleration Architecture (DPAA) data structure used to pass information between services users and service providers. The FD may convey input and/or output data, output data buffers, how input data should be or has been processed, and a number of controls to support FD queuing options, congestion and buffer management, and tracing of FDs for debug and performance analysis. For a detailed generic description of FDs see the DPAA overview chapter of your SoC reference manual.

As described in Frames and flows, a frame refers to a defined sequence of bytes and a SEC FD may convey 0, 1, or 2 frame definitions. A frame in an FD is either defining input data to be consumed, output buffer(s) to be filled, output data that has been produced, or combinations thereof. The frame specific information of the FD may refer to a single or a multi-buffer frame structure and identifies command options a recipient may need to process the frame data or status information on how the data was processed.

The Queue Manager Interface of SEC support 3 different types of FDs:

- FDs with no input or output frame specification
- Simple frame input or output FDs
- Compound (input/output) FDs

and 2 types of frames:

- Single buffer frames
- Multi-buffer frames

For detailed information on SEC FDs see Frame descriptors

### 5.2.5 Frame descriptor flow and flow context

For SEC services provided through QI, a flow refers to two or more Frame Descriptors (FDs) utilizing the same context to process frames. Whether FDs are providing one or two frames (or no frame at all) or whether the frame data is stored in one or multiple buffers is irrelevant. The key criteria of such flows is that all FDs of the flow are processed with the same SEC descriptor(s) and flow context data or controls.

A SEC flow context consists of one or more data structures used to define flow specific frame processing parameters. In all use cases a preheader precedes an optional, but usually present, Shared Descriptor (SD). A SEC flow context defines information in the following categories:

- Data and job processing attributes to optimize frame data access and job scheduling
- Input buffer release and output buffer allocation controls
- An optional, but usually present SD and the SD length (0 if SD is not present)

For detailed information on SEC preheaders see Context_A field (preheader).

### 5.2.6 Buffer allocation, release, and reuse

One of the key functions of the SEC preheader is to define how frame input and output buffers are managed. In combination with frame descriptor attributes, SEC provides the following options:

- SEC service users may use compound frame descriptors to provide both input and output frames. This enables the user to fully control both the input and the output buffer structures (and bypass any SEC buffer management limitations). Note, this approach may also be utilized to reuse part or all of the input buffers as output buffers. In this case it is the user's responsibility to provide enough output buffer space to hold the output data produced by the requested SEC service and ensure that SEC output does not destroy any input data needed to produce the output data before it can be consumed by SEC.
- SEC service users may use preheader parameter settings and configure SEC buffer management to optionally release input buffers to their associated buffer pool, allocate output buffers from up to two user-defined buffer pools, and form a variety of output frame structures. In this use case it is the user's responsibility to maintain a sufficient amount of output buffers in the utilized buffer pool(s).

For detailed information on SEC buffer management options see Context_A field (preheader).

### 5.2.7 User data access control and isolation

SEC supports user data access control and SEC-service user isolation through its ability to use service interface-specific Isolation Context Identifiers (ICIDs) to tag all related memory transactions. The ICID tag is utilized by the SMMU as a Stream ID to select table entries that define system software enforced address validity checking, optional address and attribute translation as well as other SMMU functions. For details see the SMMU chapter of your SoC reference manual.

SEC can be configured to assign different tags for sequentially accessed data (typically input/output data) and for non-sequentially accessed data (typically control or context data) when performing associated memory accesses. Control data refers to any SEC instructions or data utilized to process an input data stream in order to produce an output data stream or to just output status (e.g., an indication that the signature of some input data is correct). Depending on the selected service interface and user application, access to SEC-utilized memory is managed by up to two ICIDs.

When accessing memory the RTIC service interface utilizes only one ICID per memory block, while the job ring and register service interfaces may utilize up to two different ICIDs defined by a trusted management entity in interface-specific SEC registers.

The SEC Queue Manager service Interface (QI) supports use of up to two ICIDs. The ICID value is conveyed to SEC by QMan and expanded to up to two ICIDs utilizing SEC mask and base registers configured by a trusted management entity.

The SEC QI receives the ICID information for up to 3 Frame Descriptors from a single queue in the so-called dequeue summary information. The ICID always originate from the Queue Manager's access control attributes, which are configured by a trusted management entity.

## 5.3 Service interfaces

SEC services may be invoked via the following types of service interfaces:

- A Register-based service interface
- A Job Ring interface
- A Queue Manager Interface (QI)
- A Run-time integrity checker (RTIC)

The register-based interface is primarily intended for management entities to use for simple one-off jobs during startup, run-time testing of SEC functionality, or debugging SEC descriptors. It is not intended for repetitive or high throughput activities.

The job ring interface provides single user/driver job queuing, job completion interrupt services, and support for dynamic service interface virtualization via a software management entity. SEC implements 4 job ring interfaces that can be independently assigned (and re-assigned) to different users. This service interface type is primarily intended to be (at least temporarily) assigned to either the ARM TrustZone (TZ), system management entities or application entities. Note that creating trusted descriptors can only be accomplished via the job ring interface.

QI provides efficient sharing of SEC service for all users having access to a Queue Manager portal and an assigned frame queue pair linking the portal to SEC. Once these conditions are met a bi-directional data path is enabled for service requests and responses between a user and SEC. In this use mode the Queue Manager provides access attributes for input, output, and control (context) data associated with the service request and response. The user's service request identifies the data including details about data buffer locations and how the data is to be processed.

Jobs can also be internally initiated by SEC's RTIC submodule. RTIC is typically configured at startup (and optionally reconfigured thereafter) by a trusted or privileged management entity, and then operates autonomously, periodically submitting specialized SEC descriptors to the job queue controller.

### 5.3.1 Job Ring interface

The Job Ring interface is a software job programming interface. For each job submitted to SEC, software must create a job descriptor that explicitly describes the data to be processed and the keys and context to be used for the processing (see Job descriptors). In most high-speed networking products the QI submits the majority of job descriptors executed by SEC (see Queue Manager Interface (QI)), but the Job Ring interface may be used to create security associations and perform other one-time cryptographic operations.

SEC implements 4 Job Ring interfaces. Each Job Ring interface provides an input ring for job descriptors and an output ring for results. This queuing mechanism allows software to schedule multiple SEC jobs for execution and then retrieve the results as convenient. The input rings and output rings are implemented as circular buffers (also called rings) that are located in system memory.

The entries in the input rings are pointers to job descriptors that are located elsewhere in memory (see Address pointers). Each entry in an output ring consists of a pointer to a job descriptor followed by a job termination status word (see Job termination status/error codes). They may also be followed by a word containing the number of bytes written by SEQ STORE and SEQ FIFO STORE commands during the descriptor's execution (see INCL_SEQ_OUT field in the Job Ring Configuration Register). The job descriptor pointers in the output rings allow software to correlate result status with the particular job descriptor that SEC executed to produce that result.

#### 5.3.1.1 Configuring and managing the input/output rings, overview

Software configures the input and output rings and then manages them jointly with SEC. The following table describes the uses of the input and output ring registers:

**Table 5-4. Input/output ring registers**

| Register | Description |
|---|---|
| Input/Output Ring Base Address Register | Describes the base address of the ring buffer, which must be a multiple of four bytes |
| Input/Output Ring Size Register | Describes the size of the ring buffer measured in the number of entries |
| Input Ring Jobs Added/Output Ring Jobs Removed Register | Tells SEC how many jobs software placed in the input ring or removed from the output ring |
| Input Ring Slots Available/Output Ring Slots Full Register | Tells software how many spaces are available to add jobs to the input ring or how many jobs are in the output ring ready for software processing |
| Input Ring Read Index | Points to the head of the queue within the input ring, that is, where SEC finds the next job descriptor to read from the job ring |
| Output Ring Write Index | Points to the tail of the queue within the output ring, that is, where SEC places the results of the next completed job descriptor for that job ring |
| Job Ring Configuration Register | Used to configure job ring interrupts, set endianness overrides, and select whether the optional sequence out length word appears in Output Ring entries |

Figure 5-3 shows an example input ring and output ring. The physical ring buffers are shown in the boxes on the right. The logical queues located within these ring buffers are shown in shaded boxes to the left. Each input ring entry consists of a pointer to a job descriptor. Each output ring entry consists of a pointer to a job descriptor followed by a 32-bit word indicating the job completion status.

In this example, jobs 10 through 15 are in the input ring waiting for SEC to process them. The results for jobs 4 through 8 are in the output ring, waiting for software to retrieve them. SEC has removed job 9 from the input ring, but has not yet written the results to the output ring. Old entries that have not yet been overwritten are shown in italics.

Note that job 7 completed ahead of job 6. In versions of SEC that implement more than one DECO, it is possible for jobs submitted through the same job ring to complete out of order. See Order of job completion.

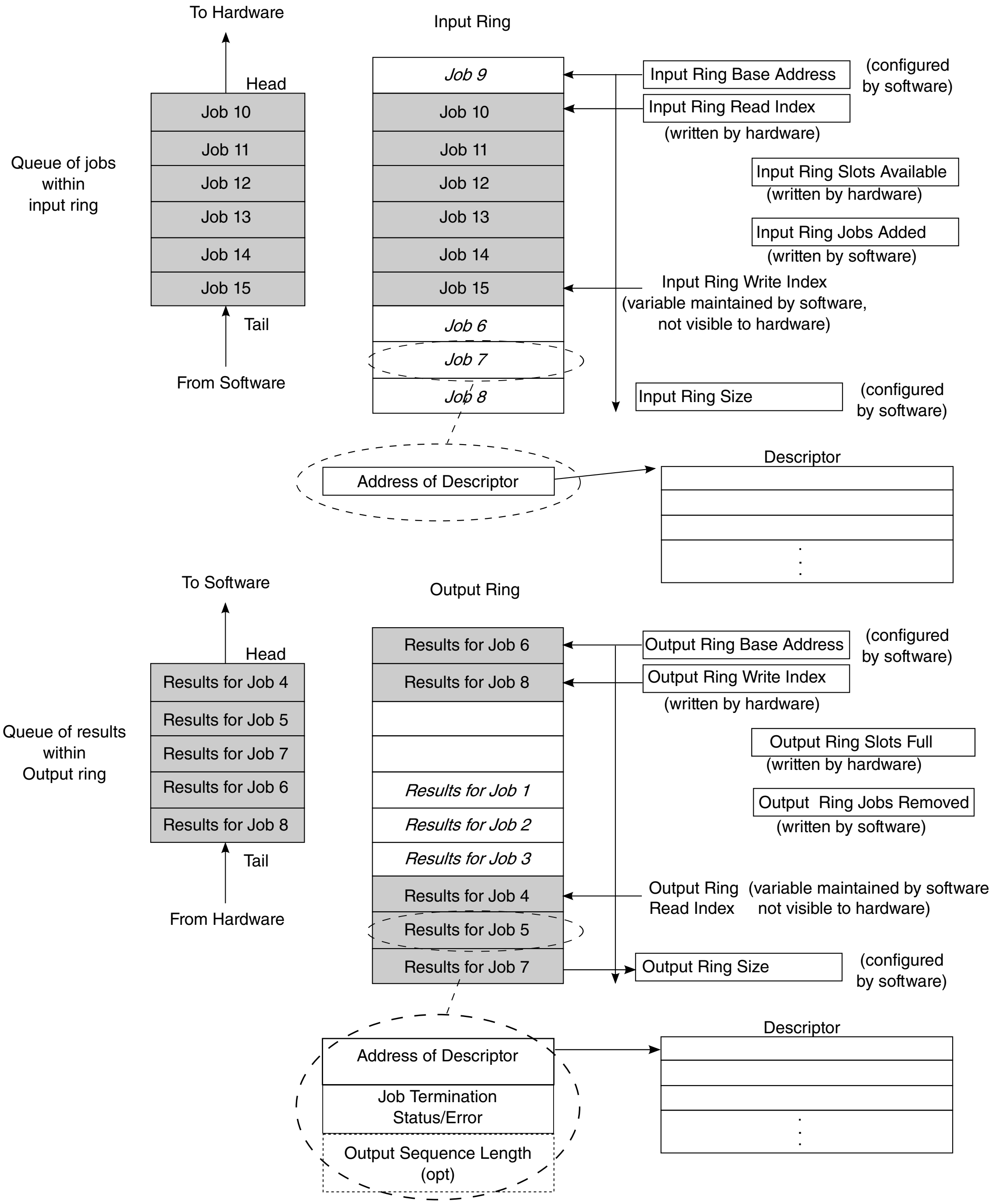


**Figure 5-3. Input and output ring example**

### 5.3.1.2 Managing the input rings

For the input ring, software is the producer, meaning that software

- Writes descriptor addresses into the job descriptor queue within the input ring.
- Writes the number of new jobs to the Input Ring Jobs Added Register

The address added to the input ring must point directly to the start of a job descriptor, not to a scatter/gather table. A job descriptor queue entry is determined by the size of a pointer, as specified in the PS field of the MCFGR Register (one word for 32-bit pointers, two words for 40-bit pointers). (See Address pointers.) Software maintains its own write pointer for the input ring, and SEC does not have direct access to that pointer.

For the input ring, SEC is the consumer, meaning that it increments the Input Ring Slots Available Register upon pulling descriptor addresses out of the queue. When software writes a new value to the Input Ring Jobs Added Register, SEC decrements the Input Ring Slots Available Register by the value that was written by software. SEC maintains a read index that it increments as it reads jobs from the input ring.

### 5.3.1.3 Managing the output rings

For the output ring, the roles are reversed from the input ring. SEC is the producer and software is the consumer.

- When SEC adds completed jobs to the output ring within the job ring, it increments the Output Ring Slots Full Register, which tells software how many results are available for software to retrieve. An interrupt may or may not be generated, depending upon the job ring configuration at the time (for more details, see Asserting job ring interrupts).
- When software removes jobs from the output ring for processing, it writes the number of jobs removed to the Output Ring Jobs Removed Register. SEC decrements the output ring slots full value by the new value that software wrote to the Output Ring Jobs Removed Register.

Note that each entry in the output ring consists of a job descriptor address and a job termination status word. See Job termination status/error codes for the format of this status word. Therefore, the size of an entry in the output ring is the size of a pointer plus one word for status, plus an optional word containing the output sequence length. SEC maintains an output ring write index that it increments as it places completed jobs and status into the output ring. Software can read this register to determine the current tail of the output ring.

Note that it is possible for a bus error to occur when the job queue controller is writing the completion status to the output ring. This results in an error code type 1 indication for that particular job ring. The correct response to any job ring error code 1 indication is to perform a job ring reset (See Job Ring Command Register RESET field), a software SEC reset or a power on reset. If a job ring reset is performed, it will clear all registers for that particular job ring except the REIR, IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers. The REIR registers should be manually reset after a job ring reset. The IRBAR, IRSR, ORBAR, ORSR, and JRCFGR registers can be reprogrammed or not, as appropriate, after a job ring reset.

#### 5.3.1.4 Controlling access to job rings

Access to a job ring can be restricted to a particular software entity because each job ring's registers are in a separate register address page. An OS or a hypervisor can enforce the restrictions by means of a memory management unit.

A process with permission to access a particular job ring's registers can:

- Schedule jobs for SEC by writing the address of a job descriptor or trusted descriptor at the tail of the queue within the input ring.
- Retrieve job completion status by reading the entry at the head of the queue within the corresponding output ring.

Each job ring can be configured so that SEC's DMA asserts different ICID values when executing jobs on behalf of that job ring. This allows slave devices or chip-level memory management units to make memory access control decisions based upon the job ring from which the job was initiated.

#### 5.3.1.5 Order of job completion

Job descriptors submitted through different job rings are not guaranteed to complete in the order they were submitted by software, even if they reference the same shared descriptor and use SERIAL or WAIT sharing. (See Shared descriptors for details about shared descriptors and sharing concepts).

As shown in the example in Figure 5-3, it is possible for results to be written into the output ring in a different order than the order in which the corresponding jobs appear in the input ring (see jobs 6 and 7, where 7 is a short job submitted after 6, which requires more processing time). Because jobs are assigned to DECOs as the DECOs become available, successive job descriptors generally execute in parallel in different DECOs. Therefore, the order of job completion is affected by the time required to process the data, the presence of a shared descriptor, and the sharing mode.

The only way to guarantee that jobs input on a single job ring complete in the order they were added to the input ring is to both:

- Have those jobs reference a shared descriptor
- Set the SERIAL sharing bits in the descriptor headers.

Note that the majority of the processing information can be included in the job descriptor with the shared descriptor enforcing serialized processing.

WAIT sharing can be used if all that is required is to guarantee that certain commands in one job are complete before another job is started. Types of sharing and impact on job completion ordering are further described in Specifying different types of shared descriptor sharing.

### 5.3.1.6 Initializing job rings

Minimal configuration for SEC operation using the job ring interface requires initializing at least one job ring by specifying the base addresses for the input ring and output ring and the size of these rings (see the Input Ring Base Address Register (IRBAR), the Output Ring Base Address Register (ORBAR), the Input Ring Size Register (IRSR), and the Output Ring Size Register (ORSR)). Most cases (with the possible exception of debugging with test data in use) also require specifying the ICID values associated with the job ring (see the JRaICID register). These values should be configured by a trusted SoC / ring management entity. The job Rings can also be configured for endianness and for whether to include the optional sequence out length word in the Output Ring entries (see Job Ring Configuration Register (JRCFGR) ).

### 5.3.1.7 Job Ring Registers

If the Job Ring is allocated to TrustZone SecureWorld, the Job Ring registers associated with this ring can be written only via a TrustZone secure bus transaction. Nonsecure writes to Job Ring registers owned by Trustzone SecureWorld will be ignored.

When virtualization is enabled (VIRT_EN=1 in the Security Configuration register), the Job Ring registers in pages 1...4 can be written only if the corresponding Job Ring has been "started", that is, the JRSTARTR[Start_JR] bit for that Job Ring is 1. Conversely, the Job Ring configuration registers in page 0 (for example, the JRaICID register) can be written only if the Start_JR bit for that Job Ring is 0. The Job Ring registers are reset when the Job Ring is stopped and virtualization is enabled, in order to prepare the Job Ring for a new owner. The input ring slots available, input ring read index, output ring slots full and output ring write index registers are read-only when virtualization is disabled. These registers are writable when virtualization is enabled.

For the job ring register descriptions, see IRBAR and the following register descriptions.

#### 5.3.1.8 Asserting job ring interrupts

Each job ring interface asserts an interrupt request on a separate interrupt request line to notify the driver software that job results are available from that job ring. Note that the software context switch overhead could have a severe performance impact if interrupts were asserted for every job completion. Therefore, SEC supports a configurable interface that allows the driver to specify how full the output ring can be before SEC generates a job completion interrupt request. To prevent any job from waiting too long for software completion processing, the driver can also specify a time out value. This value allows SEC to generate an interrupt if job results are available and too much time has elapsed since software last removed any completed jobs from the output ring. These values are programmed via fields described in the Job Ring Configuration Register (JRCFGR).

The job ring interrupt does not clear automatically when jobs are removed from the output ring. Software must clear the interrupts by writing to the Job Ring Interrupt Status Register. Note that one or more additional jobs can complete while software is clearing the interrupt. Depending on the interrupt coalescing settings, an additional interrupt may immediately be generated for these newly completed jobs.

### 5.3.2 Queue Manager Interface (QI)

QI is a high performance, flow-oriented job execution interface used in devices implementing the Data Path Acceleration Architecture (DPAA). Once initialized and enabled, QI performs the following functions:

- Request and receive (dequeue) frame descriptors from QMan for processing
- Acquire buffers from the Buffer Manager (BMan) for output data (optional)
- Prepare data for processing by assembling a job descriptor
- Pass the job descriptor to the job queue controller for job scheduling
- Receive job completion status from a descriptor controller (DECO)
- Release input frame buffers (optional)
- Send (enqueue) frame descriptors with processing results to QMan

In contrast to the job ring interface, which requires the dedication of a job ring to a relatively small group of select users, all users can be given access to a shared SEC by providing services via DPAA frame descriptors exchanged with SEC in queues. Frame queues are initialized with a pointer to a preheader typically followed by an optional but usually present shared descriptor that may supply session keys, context, and other processing instructions. Every frame descriptor enqueued to SEC is processed according to the information in the preheader and the shared descriptor. For more information, see Context_A field (preheader) and Shared descriptors.

From an application software perspective, mapping a security processing flow to a frame queue and linking the frame queue to the preheader/shared descriptor is an infrequent occurrence, corresponding to establishment or refresh of a security session, e.g. an Internet Key Exchange (IKE) establishing an IPsec tunnel. The enqueing of frame descriptors to the right frame queue, SEC's dequeue and processing of the data associated with those frame descriptors, and SEC's return of the frame descriptors to software corresponds to a data path.

Refer to the DPAA overview chapter of your SoC reference manual for a comprehensive explanation of frame queues and frame descriptors. Refer to the DPAA reference manual for your SoC to see a detailed description of Queue Manager and Buffer Manager. For an abbreviated discussion of these topics relevant to SEC, see Frame queues, frame descriptors, and buffers.

QI is usually configured and enabled during system initialization, but can be stopped, resumed, and (re-)configured later. The specific mandatory configuration steps are described in Initializing the Queue Manager Interface.

#### 5.3.2.1 Requesting and receiving frame descriptors from QMan

In response to each dequeue command, QMan sends QI one to three frame descriptors from a selected frame queue. The number of requested frame descriptors is determined by QI's configuration (see the Queue Interface Dequeue Configuration Register (QIDQC) for details). However, it is the QMan programming that controls the choice of frame queues and frame descriptors in response to the dequeue request, not the configuration of the QI.

The QI uses a QMan mechanism called subportals to ensure that QMan selects frame descriptors from different frame queues. Each dequeue command specifies a subportal ID. When issuing these commands QI uses as many subportal IDs as the number of job queue holding tanks. This allows SEC to process frames from more than one queue simultaneously, increasing opportunities for parallel processing. If jobs from a few frame queues were to fill up all available job slots QI might not be able to fill all the holding tanks in the job queue controller, possibly resulting in idle DECO resources and decreased throughput. To avoid this condition QI limits the number of outstanding dequeue requests for any subportal to one, and it does not issue a request for a subportal if the number of jobs from the subportal equals or exceeds the threshold defined by the SPFCNT field in the QIDQC register.

In response to each dequeue request, QMan sends QI one to three frame descriptors and summary information applying to all dequeued frames:

- Number of Frames dequeued

- Frame queue ID for enqueuing processing results (Context B field, bits 32-63 of the summary information)
- Memory address of the preheader (Context A field, bits 64-127 of the summary information)

### 5.3.2.2 Building job descriptors for QI jobs

QI uses the QMan dequeue summary information and the frame descriptors to build an internal job descriptor for each job. The summary information is primarily used to direct responses and provide frame processing control and context data structures. For most purposes, SEC hardware does not distinguish between job descriptors created by software and submitted to SEC through the job ring interface and job descriptors created by QI using dequeue information. The term Job Descriptor (JD) is used generically when describing processing instructions or embedded data utilized by SEC to process jobs.

Information for each job, including the input and output frame addresses, is stored in job buffers internal to QI. Job buffers are used to save all data needed to build a job descriptor, maintain processing status, track job order, and eventually return (enqueue) output to QMan.

On every cycle QI performs the following:

1. Searches its list of jobs waiting for transfer to the job queue controller
2. Determines each job's eligibility for transfer
3. Assigns selection priorities based on several criteria
4. Selects the next job, if any, for transfer

Priorities are assigned to eligible jobs to maximize sharing and throughput. QI builds the job descriptor for the selected job and notifies the job queue controller that the job is ready for transfer into an available holding tank.

### 5.3.2.3 Controlling QI access to frame queues and data

The use of frame queues in DPAA allows a large number of user software processes to share SEC. Users are prevented from directly accessing each other's private memory space through proper configuration of the CPU MMUs. When processing a job on behalf of a user process, SEC is prevented from accessing the private memory of other processes by means of the System Memory Management Unit (SMMU) and one or more ICID values assigned to each job. A general description of ICID usage in DPAA can be found in the DPAA overview chapter of your SoC reference manual.

The frame descriptor dequeued from QMan specifies an ICID, which in turn is used by QI to derive a SEQ and a non-SEQ ICID used to access input/output or control data. Depending on the ICID value and the QI ICID register configuration, SEQ and non-SEQ ICID values may be determined to be either the same or different. Control data includes the preheader and the optional but usually present shared descriptor. When QI reads the preheader, it uses the non-SEQ ICID whereas if the frame descriptor is a compound frame descriptor, QI reads and writes the compound frame list table using the SEQ ICID. The two ICIDs are passed with the job from QI to the job queue controller and DECO so that the appropriate ICID can be used by those processors for their memory accesses.

For information on how to configure ICIDs, passed to QI in the frame descriptor, see the QMan chapter in the DPAA reference manual. See the DECO ICID Status Register (DxISR) and the QI ICID Configuration Register (QIICR) for a description of how SEQ and non-SEQ ICID values are derived. SEQ and non-SEQ commands are explained in SEQ vs non-SEQ commands.

### 5.3.2.4 Tracking the completion order of QI jobs

QI uses ordered lists to track the dequeue order of jobs with the same preheader address. When job processing is done, QI uses the lists to enqueue jobs with the same preheader address in the same order they were dequeued.

When a DECO notifies the QI that a job has finished, QI checks the appropriate list to determine whether the job is the oldest job with that preheader address. If so, QI enqueues the job's frame descriptor. Otherwise the enqueue is delayed to maintain order. Under normal operating conditions (except system error or invalid queue IDs) QMan will not reject SEC enqueues.

See the Queue Manager section of the reference manual for a discussion of the hardwired settings of the SEC Direct Connect portal (DCP). QMan must be configured by software to select an error queue to store frames of SEC's rejected enqueues.

### 5.3.2.5 Initializing the Queue Manager Interface

Configuring and enabling QI requires initializing:

- Queue Interface ICID Configuration Registers
- Queue Interface Dequeue and Enqueue Configuration Registers (optional)
- Queue Interface Control Register (at a minimum enable dequeues)

Additional QI registers are accessible through the SEC register interface to assist with debugging.

Note that in addition to the SEC QI configuration, at least two frame queues must be initialized in QMan so that software can enqueue frame descriptors to SEC, and SEC can enqueue results back to software.

#### 5.3.2.6 Done/error notification for QI jobs

There are no SEC interrupts associated with QI. QI is utilized through QMan portals and queues. For all jobs handled by QI, completion and error status is indicated in the 32-bit STATUS/CMD field of the response frame descriptor SEC sends to QMan. The most-significant four bits of this field identify where the error was detected within SEC. For example, the value '5' indicates that QI detected the error and '4' indicates that DECO detected the error. The remaining bits provide more specific status details. See Table 5-3 for a complete list and formatting of error codes.

### 5.3.3 Register-based service interface

It is possible to use the register interface to perform entire cryptographic operations. For the purposes of debugging descriptors, it is possible to execute descriptors one descriptor at a time, or even one descriptor command at a time. [1] This method bypasses all job scheduling performed by the job queue controller. Software can perform CHA operations by writing and reading registers in the CCB directly, without using a Job Ring or the QI to run descriptors. When descriptors or commands are executed in this mode, software can examine the content of most DECO and CCB registers after each descriptor or descriptor command completes. This can assist with debugging hardware and descriptor programs.

To execute descriptors or commands in this fashion, software must request direct use of a DECO by writing into the DECO REQ register. But before requesting a DECO, software must specify the SDID and ICID values that will be used when executing descriptors under direct software control. When virtualization is disabled these values are specified via the DECO ICID Register for the selected DECO. When virtualization is enabled, the DECO Request Source register is used to select a particular Job Ring to act as the source for the ICID and SDID values. The DECO Request Source register must be written prior to writing the RQDn bit in the DECO REQ register.

To use the register-based job service interface, the DECO must be programmed in proper order so that a descriptor runs correctly. The steps are:

1. Specify the ICID and SDID values using the DECO Request Source register or appropriate DECO ICID Register

[1] Note that trusted descriptors cannot be executed via the register-based service interface.

2. Set the RQDn bit in the DECO Request Register. This RQDn bit must remain asserted during the entire time that software wants to access that DECO/CCB block directly. This indicates to the job queue controller that it should not assign any jobs to the requested DECO block. After the job queue controller sees the RQDn bit set to 1, it waits for the corresponding DECO block to complete any pending tasks.
3. Wait for the DENn bit in the DECO Request Register to be set to 1. The job queue controller sets the DENn bit to 1 when the DECO block becomes available. When this bit is set, software can use the DECO/CCB block by submitting descriptors by means of SEC's register interface.
4. Write at least the first burst of a descriptor into the descriptor buffer. If there is a shared descriptor, offset the descriptor into the descriptor buffer by the length of the shared descriptor.
5. Write the address of the descriptor into the DECO Descriptor Address Register so DECO knows where to find the descriptor. This is only required if the WHL bit (see the next step) is not set or if the descriptor attempts to do a STORE of type 41h or 45h to write back part, or all, of the descriptor to memory.
6. Write the Job Queue Control Register. If fewer than 4 words are in the first burst, the FOUR bit must be 0. If the entire descriptor has already been loaded, set the WHL bit. If the WHL bit in the DECO Job Queue Control Register is not set, DECO attempts to fetch the rest of the descriptor from memory regardless of whether portions of the descriptor beyond the first burst were already written to the descriptor buffer. SHR_FROM is not used in this format and will not be checked.
7. Wait until the DECO is done. To determine whether DECO is done, read the VALID and DECO_STATE fields in the DECOa DBG_DBG_REG register. While the job is running, VALID will be 1 and DECO state will change values as the descriptor is processed. If DECO_STATE is Dh, then an error occurred. Read other fields and registers to determine the cause of the error. Note that VALID will likely remain asserted in the event of an error. If DECO_STATE is 0h and VALID is 0, then the job finished normally.
8. Read registers of interest.
9. Done or start over.
10. When software is finished using the DECO/CCB block, it must clear the RQDn bit so that the DECO is available to the job queue controller for normal processing. The job queue controller then de-asserts the DENn bit, which resets the DECO and CCB.

Note that there are restrictions imposed when executing a descriptor under software control:

- The special cryptographic keys used to encrypt or decrypt Black Keys are not available, so Black Keys cannot be used.
- The master cryptographic key used to encrypt or decrypt Blobs is not available, so Blobs cannot be used.

- Sharing of Shared Descriptors is not permitted.
- Trusted Descriptors are not allowed.
- When virtualization is enabled, a Job Ring source must be selected in the DECO Request Source Register before executing any job under direct software control. All jobs running under direct software control will then utilize the ICID and SDID values for the Job Ring selected in the DECO Request Source Register. When virtualization is disabled, any job under direct software control will utilize the ICID and SDID values specified in the DECO ICID register, and the SRC field in the Job Queue Control Register must be programmed to indicate the job is running on behalf of one of the Job Rings. Jobs cannot be executed under direct software control if those jobs appear to be from other possible SEC sources, such as:
    - QI
    - RTIC

The normal use case for the register based service interface is to debug descriptors. When such a descriptor is run through the interface and the descriptor encounters an error, once analysis of the error is done, the only way to recover is to release the DECO. The user can recover by releasing the DECO or by writing a 1 to the STEP bit in the DECO Job Queue Control Register. The second method allows another descriptor to be loaded and run as described above.

## 5.4 Job scheduling

The job queue controller is the job scheduler within SEC. The default job scheduling algorithm operates as follows. The job queue controller pulls jobs to be sent to the holding tanks in round-robin fashion from the Job Rings, then from QI, and then from RTIC.

### 5.4.1 Job scheduling - default algorithm

Each time that the Job Ring's turn comes up in rotation and there is a job available in that Job Ring's input queue, a job is selected from that Job Ring. But because SEC buffers input ring entries for efficiency, several jobs may be scheduled from one Job Ring before a job is scheduled from the next Job Ring. Eventually all Job Rings will be serviced.

Each time that the QI's turn comes up in rotation and there is a QI job that can be processed, one QI job will be selected for transfer to JQ. QI selects that job from its list of jobs waiting for transfer using several criteria to determine eligibility and priority. QI builds its list of jobs eligible for transfer by dequeuing a number of jobs from one subportal and then switching to a different subportal for the next dequeue. This prevents starving any subportal.

This selection method favors jobs with the potential for sharing. The priority of a QI job will be reduced if the job uses CRID to specify that it requires a critical resource that cannot be used by all DECOs in parallel and all of those resources are already in use, unless there is already a job from this flow in a DECO (see Context_A field (preheader) ). A selection priority is calculated for each job waiting for transfer to the job queue controller, in decreasing order of priority:

1. Job is not the first job of a dequeue and another job from the same flow is currently in a DECO.
2. Job is first of a dequeue and another job from the same flow is currently in a DECO.
3. No job from the same flow is currently in a DECO and either CRID is not set or the critical resource is available.
4. No job from the same flow is currently in a DECO, CRID is set, and the critical resource is not available.

Note that RTIC requests at most one job at a time.

The following figure illustrates the algorithm for selecting a job for an available holding tank.

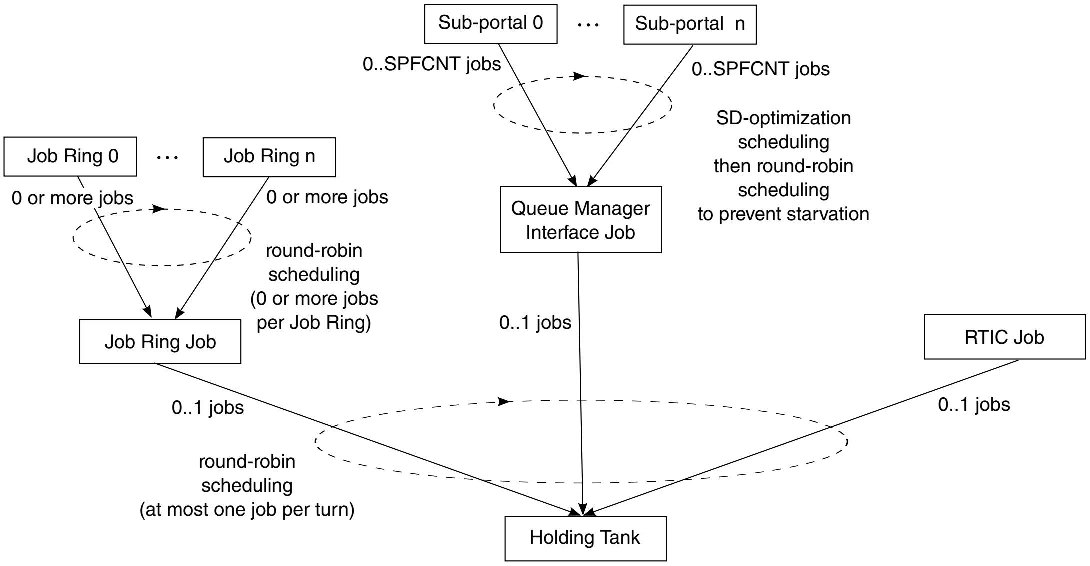

Sub-portal 0
...
Sub-portal n
0..SPFCNT jobs
0..SPFCNT jobs
SD-optimization scheduling then round-robin scheduling to prevent starvation
Job Ring 0
...
Job Ring n
0 or more jobs
0 or more jobs
round-robin scheduling (0 or more jobs per Job Ring)
Queue Manager Interface Job
Job Ring Job
0..1 jobs
RTIC Job
0..1 jobs
0..1 jobs
round-robin scheduling (at most one job per turn)
Holding Tank

**Figure 5-4. Selecting job for available holding tank**

The job queue controller prefetches some or all of the selected job descriptor and possibly the shared descriptor (if any, and it is not already in a DECO) and places them in a buffer referred to as a holding tank. After a job has been put in a holding tank, it is then eligible for dispatching to a DECO. If the job source is the QI the holding tank also fetches up to two bursts' worth of input frame data (up to the second burst boundary). SEC generally implements the same number of holding tanks as DECOs.

This prefetching of the job descriptor allows the job queue controller to take shared descriptors into consideration when allocating jobs to DECOs. SEC attempts to dispatch jobs to available DECOs as efficiently as possible. If (1) a job descriptor with a shared descriptor is currently executing in DECO n, (2) this descriptor can be shared, and (3) there is another job descriptor in a holding tank that references the same shared descriptor, the job descriptor in the holding tank is marked as pending for DECO n.

The following figure illustrates the SEC's dispatching algorithm, which favors reuse of already fetched descriptors but avoids starving Job Rings or QI queues. Note that when using the default scheduling method, the job source (QI, RTIC, Job Ring) is not considered when deciding which holding tank job should be assigned to the available DECO.

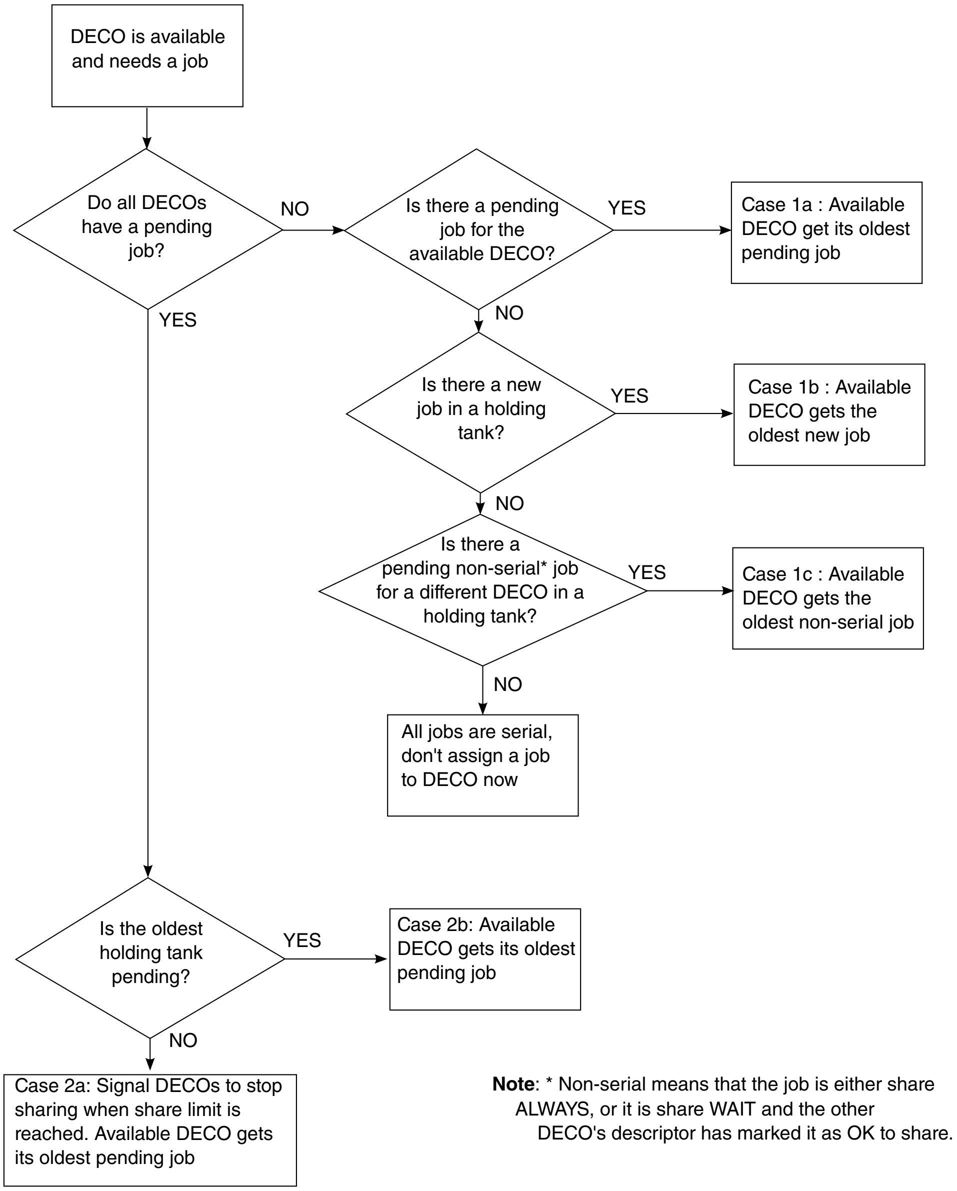

DECO is available and needs a job
Do all DECOs have a pending job?
NO
YES
Is there a pending job for the available DECO?
YES
NO
Case 1a : Available DECO get its oldest pending job
Is there a new job in a holding tank?
YES
NO
Case 1b : Available DECO gets the oldest new job
Is there a pending non-serial* job for a different DECO in a holding tank?
YES
NO
Case 1c : Available DECO gets the oldest non-serial job
All jobs are serial, don't assign a job to DECO now
Is the oldest holding tank pending?
YES
NO
Case 2b: Available DECO gets its oldest pending job
Case 2a: Signal DECOs to stop sharing when share limit is reached. Available DECO gets its oldest pending job
**Note**: * Non-serial means that the job is either share ALWAYS, or it is share WAIT and the other DECO's descriptor has marked it as OK to share.

**Figure 5-5. Job queue controller's job scheduling algorithm**

This approach gives preference to serial sharing whenever doing so does not cause a job to remain indefinitely in a holding tank. Serial sharing occurs in cases 1a, 2a, and 2b in the algorithm described above. Serial sharing is the most efficient since the information that the new job shares with the previous job is already in the DECO. Jobs are shared between DECOs (case 1c) only when the available DECO has no pending job and there are no non-pending jobs in the holding tanks. This is done because sharing between DECOs is less efficient than sharing serially, since the shared information must be copied to the available DECO.

### 5.4.2 Job scheduling - DECO-specific jobs

If there is more than one DECO it is possible to specify that a job should be run in a specific DECO by using the job header extension word. This can be used to support hardware testing. Once a DECO-specific job enters a holding tank, it remains there until the specified DECO becomes available, with the following exception. If the DECO-specific job contains a shared descriptor, specifies serial sharing, and the shared descriptor currently resides in a DECO other than the specified DECO, the DECO-specific job runs serially in the DECO that already contains the shared descriptor, resulting in a DECO-select error job termination code.

**NOTE**

DECO-specific jobs can create a deadlock in SEC when they are used as part of a flow. Therefore, it is strongly recommended that DECO-specific jobs should not be used in a flow.

## 5.5 Job execution hardware

The following modules in SEC execute cryptographic acceleration jobs:

- Descriptor controller/cryptographic control block
- Cryptographic hardware accelerators

### 5.5.1 Descriptor controller (DECO) and cryptographic control block (CCB)

The descriptor controller (DECO) is responsible for executing SEC job descriptors. After the job descriptor and any shared descriptor referenced by that job descriptor are loaded, DECO begins processing. Each DECO has a dedicated CCB (cryptographic control block) that it uses to access any cryptographic hardware accelerators (CHAs) needed to perform cryptographic functions.

When executing a descriptor, DECO activates the DMA controller to read the required inputs, and uses the CCB to dispatch the job to the appropriate CHAs. As data is produced by the CHAs, DECO activates the DMA to write the results and job completion status information out to the locations specified in the descriptor. When a descriptor finishes, either successfully or with errors, DECO informs the job source (Job Ring interface, QI or RTIC), which then takes appropriate action.

The CCB contains all the hardware necessary to control the various CHAs included in SEC. Every CCB has access to every type of CHA so that every DECO/CCB pair can perform all functions that can be performed by SEC. Note that there may be fewer instances of a CHA than there are CCBs. In such cases, a CCB may have to wait until the CHA it wants is available before proceeding. Arbitration for shared CHAs is automatically handled by SEC in all operating modes.

The hardware inside the DECO/CCB includes the input FIFO, output FIFO, information FIFO (NFIFO), mode registers, context registers, key registers, descriptor buffer, math registers, scatter/gather tables, alignment blocks and interconnects. DECO/CCB uses all of this hardware to process descriptors.

#### 5.5.1.1 Alignment blocks

SEC 's internal data pathways and cryptographic engines generally operate on 64-bit data, but the information that SEC obtains from memory need not be aligned to 64-bit boundaries. To concatenate and left-align information passed to certain destinations within SEC, SEC architecture includes three alignment blocks:

- Class 1 alignment block
- Class 2 alignment block
- DECO alignment block

Note that even if the data is aligned in memory, the alignment blocks may still need to align some portions of the data because a subset of the data may be passed to more than one destination, and the subset may need to be aligned separately for each destination.

The following figure illustrates the interconnections of one of the alignment blocks. All three alignment blocks have the inputs shown in the figure. The Class 1 alignment block contains a nibble shift register, which allows the Class 1 alignment block to handle data that needs to be shifted by a nibble. The only other difference between the alignment blocks is the consumer (Class 1 CHA, Class 2 CHA, or DECO).

The entry pulled from the NFIFO tells the NFIFO controller the data source that will be used with the alignment block, and whether the alignment block will be flushed when the data transfer is complete. The alignment blocks normally transfer eight bytes of data at a time to the consumer. When the amount of data needed by the consumer is not a multiple of eight bytes, a "flush" flag or "last" flag is required to transfer the last one to seven bytes from the alignment block to the consumer.

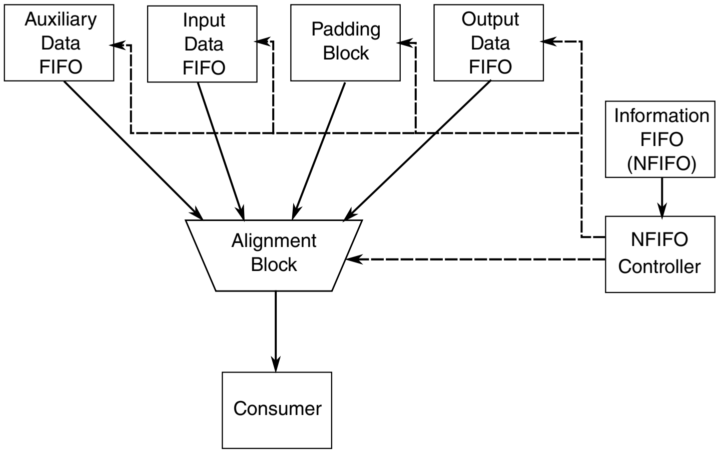


**Figure 5-6. Alignment block interconnections**

All data entering Class 1 CHAs first passes through the Class 1 alignment block, which ensures that the data presented to the Class 1 CHA is properly concatentated and left-aligned. Note that the Class 1 alignment block can also serve as the source for a MOVE command (see Table 7-33, SRC value 9h and AUXLS = 1).

Similarly, all data entering Class 2 CHAs first passes through the Class 2 alignment block, which ensures that the data presented to the Class 2 CHA is properly concatentated and left-aligned. The Class 2 alignment block can also serve as a source for the MOVE command (see Table 7-33, SRC value 9h and AUXLS = 0).

The third alignment block is the DECO alignment block, which can be used as a data source for a MATH command (see MATH and MATHI Commands, SRC0 and SRC1 fields) and as a data source for a MOVE command (see Table 7-33, SRC field).

Note that the only way to put data into an alignment block is with an info FIFO entry. Therefore, when using an alignment block as the data source for a MOVE command, the data source for the alignment block must have been specified with an info FIFO entry. This info FIFO entry may be automatically or manually generated. In order to use data stored in the input FIFO, that data must be passed through one of the alignment blocks. The only other way to take data out of the input FIFO is by resetting the input FIFO which also resets the alignment blocks.

## 5.5.2 Cryptographic hardware accelerators (CHAs) (overview)

SEC contains multiple cryptographic hardware accelerators (CHAs), each of which accelerates an encryption (Class 1) algorithm or message integrity (Class 2) algorithm.

- PKHA (public key hardware accelerator), see Public-key hardware accelerator (PKHA) functionality.
- DESA (DES accelerator), see Data encryption standard accelerator (DES) functionality.
- AESA (AES accelerator), see AES accelerator (AESA) functionality.
- MDHA (message digest hardware accelerator), see Message digest hardware accelerator (MDHA) functionality.
- RNG (random number generator), see Random-number generator (RNG) functionality.
- STHA f8 (SNOW 3G f8 hardware accelerator), see SNOW 3G f8 accelerator functionality.
- STHA f9 (SNOW 3G f9 hardware accelerator), see SNOW 3G f9 accelerator functionality.
- CRCA (cyclic redundancy check accelerator), see Cyclic-redundancy check accelerator (CRCA) functionality.
- KFHA f8/f9 (Kasumi hardware accelerator), see Kasumi f8 and f9 hardware accelerator(KFHA) functionality.
- ZUCE (ZUC encryption hardware accelerator), see ZUC encryption accelerator (ZUCE) functionality
- ZUCA (ZUC authentication hardware accelerator), see ZUC authentication accelerator (ZUCA) functionality

# Chapter 6
# Frame queues, frame descriptors, and buffers

Frame queues (FQs), frame descriptors (FDs), and (managed) frame buffers are concepts introduced in the DPAA overview chapter of your SoC reference manual and described in detail in the Queue Manager (QMan) and Buffer Manager (BMan) chapters of the DPAA reference manual.

As a component of DPAA, SEC interacts with QMan to dequeue and enqueue FDs, and with BMan to obtain and release managed buffers. These interactions and the processing required to turn FDs into SEC job descriptors are primarily handled by SEC's QMan Interface (QI), allowing the inner portions of SEC (the job queue and descriptor controllers) to be mostly ignorant of DPAA data structures and related processing requirements.

This section provides a functional overview of SEC's usage of FQs and FDs, including related interactions with QMan and BMan.

## 6.1 Frame queues

A pair of frame queues is used to submit jobs to SEC via QI and return results back to the service user. From SEC's perspective, a frame queue and a job ring are similar: each is a source of input and a destination for output. The major difference (besides the hardware/software interface) is the ability to efficiently share SEC services for multiple users and to support the optional use of BMan managed buffers.

A SEC job ring is a highly constrained hardware resource that can only be efficiently operated by a single owner, i.e., job requests from multiple users must either be coordinated by the job ring owner (e.g., an OS driver) while users (e.g., OS kernel threads) share common memory access rights (tied to the job ring configuration), or management software must grant full or temporary ownership of a job ring to individual users (e.g., a user process or thread) and memory access rights are (re-)configured by the management software on a per user basis.

The SEC QI on the other hand can receive frame descriptors and queue-specific user resource access rights (the basis for QI generated job descriptors) from millions of queues thus eliminating the need for control software to manage SEC service access for many users dynamically. Instead, control software configures queues (and associated user resource access rights) and assigns those queues to specific users thus enabling SEC to receive user-specific access rights as well as other processing parameters when one or more frame descriptors are de-queued from the user's queue for processing.

Besides access rights, software initializing SEC frame queues is required to define the Context_A and Context_B parameters in the Frame Queue Descriptor (FQD) which is utilized by QMan to enqueue, store, and dequeue frame descriptors destined to DPAA accelerators like SEC and return results back to a user. For SEC the Context_A parameter is used to define a pointer to the SEC preheader. The preheader address is passed to SEC during frame dequeue and points to the initial portion of a set of data structures utilized by SEC to process all frames arriving on the queue managed with this FQD. Context_B is used to convey the response/output frame queue ID utilized by the service user to receive SEC results.

### 6.1.1 Dequeue response

Each time SEC's QI dequeues work (in the form of 1-3 frame descriptors) from a frame queue, it also receives dequeue summary information from the QMan's Frame Queue Descriptor (FQD) in the response to a dequeue command issued by SEC. FQDs are initialized by management software during the creation of frame queues. The first word of the dequeue response of QMan, called the summary information, includes the following fields of interest to SEC:

- Number of Frames dequeued

  The number of dequeued frames is used by QI to determine for how many frames the dequeue summary information applies.

- Dequeue Context_A (address of preheader)

  Dequeue context A is used to convey a memory address to the SEC preheader. The memory content identified by the preheader provides information for SEC to determine whether and how to allocate output frames and the presence and size of an optional shared descriptor. For details of the preheader format see Context_A field (preheader).

- Dequeue Context_B (Response/output frame queue ID)

Dequeue context B is used to convey the frame queue ID to be used for enqueing the frame descriptor returning SEC processing results.

For a generic description of the details on all dequeue summary information fields see the QMan chapter in the reference manual of your SoC.

#### 6.1.1.1 Context_A field (preheader)

Management software populates this field in the FQD linked to SEC's Queue Manager (QMan) portal. QMan in turn delivers the Context_A field content in the dequeue summary information to SEC, which in turn interprets Context_A as an address to the SEC preheader.

The data stored in the preheader has the following uses:

- If an output frame is required and not already provided, the preheader provides QI with the information needed to create a frame for storing SEC output data.
- The preheader specifies whether a shared descriptor is defined for processing the associated frames.
  - If defined, the shared descriptor follows the preheader in memory (preheader address + 8B).
  - If not defined, an in-line job descriptor must be included at the beginning of the input frame or no processing will be performed and the job will be terminated with an error status code. See Using in-line descriptors for more details about in-line job descriptors.

The following figure and table show the preheader format.

**Table 6-1. Preheader format**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RSLS | EWS | RSV | | CRID | | | | Reserved | | | | | TBPSIZ | | |
| **47** | **46** | **45** | **44** | **43** | **42** | **41** | **40** | **39** | **38** | **37** | **36** | **35** | **34** | **33** | **32** |
| TBPID | | | | | | | | Reserved | | SDLEN | | | | | |
| **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** | **23** | **22** | **21** | **20** | **19** | **18** | **17** | **16** |
| DIFREL | Reserved | FSGT | LONG | OFFSET | | ABS | ADDBUF | POOLID | | | | | | | |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| POOL_BUF_SIZE | | | | | | | | | | | | | | | |

## Table 6-2. Preheader field descriptions

| Field | Description |
|---|---|
| 63<br>RSLS | Require SEQ ICIDs be the same<br>When sharing takes place, if this bit is set the two jobs must be using the same SEQ ICID. If this bit is set and the SEQ ICIDs don't match, an error is reported and the second descriptor (the descriptor that would have received the shared data) is not executed. |
| 62<br>EWS | Enable Write Safe<br>Enable write-safe for SEQ stores to the output frame for this job. |
| 61-60 | Reserved |
| 59-56<br>CRID | Critical resource ID<br>If a non-zero critical resource ID is programmed in this field, QI checks availability of the resource when selecting the next job for transfer to the job queue. If all instances of the critical resource are in use by jobs from other flows, the selection priority of jobs from this flow is lowered. |
| 55-51 | Reserved |
| 50-48<br>TBPSIZ | Table Buffer Pool Buffer Size. If not zero, this field specifies the size of buffers in TBPID.<br>000 alternate Buffer Pool disabled<br>001 alternate Buffer Pool buffers are 64B (4 entries)<br>010 alternate Buffer Pool buffers are 128B (8 entries)<br>011 alternate Buffer Pool buffers are 256B (16 entries)<br>100 alternate Buffer Pool buffers are 512B (32 entries)<br>101 alternate Buffer Pool buffers are 1024B (64 entries)<br>110 alternate Buffer Pool buffers are 2048B (128 entries)<br>111 alternate Buffer Pool buffers are 4096B (256 entries) |
| 47-40<br>TBPID | Table Buffer Pool ID. If TBPSIZ is non-zero, buffers from this buffer pool are to be used for creation of new Scatter-Gather Tables. The buffers referenced by this pool ID must be 64-byte aligned. |
| 39-38 | Reserved |
| 37-32<br>SDLEN | Length of shared descriptor, in 32-bit words, following the preheader.<br>If 0, there is no shared descriptor following the preheader. In that case, a job descriptor is included at the beginning of the input frame. |
| 31<br>DIFREL | Disable Input Frame Release. This causes QI to create a compound frame for enqueuing both the input and output frames. |
| 30 | Reserved |
| 29<br>FSGT | This bit is used to force QI to build the output frame with a scatter/gather table, even if a single buffer would suffice to hold the output data. Note that this bit is ignored if an output frame is provided to QI via a Compound Frame.<br>• If set, QI creates an output frame with a scatter/gather table.<br>• If cleared, QI uses a scatter/gather table only if multiple buffers are required. |
| 28<br>LONG | If set, QI uses the long buffer format for the output frame and associated frame descriptor. The long format frame descriptor provides 29 bits for frame length and no bits for offset.<br>If cleared, QI uses the short buffer format: 20 bits of length, 9 bits of offset. Note that if LONG is set, the OFFSET field is ignored.<br>**NOTE:** This bit is ignored if a compound frame provides the output frame is provided to QI. |

*Table continues on the next page...*

Table 6-2. Preheader field descriptions (continued)

| Field | Description |
|---|---|
| 27-26<br>OFFSET | This field is used to indicate if an offset should be placed in the first buffer of the output frame and, if so, how much offset. Note that this bit is ignored if a compound frame provides the output frame to SEC. It is up to software to ensure that sufficient space will be available in the output frame, including the space required for the offset. Software must also ensure that the offset is at least one byte less than POOL_BUFFER_SIZE. Note that if LONG is set, the OFFSET field is ignored.<br>00: No offset.<br>01: Add an offset equivalent to 1 burst (64 bytes).<br>10: Add an offset equivalent to 2 bursts.<br>11: Add an offset equivalent to 3 bursts. |
| 25<br>ABS | This bit specifies whether the number of buffers allocated is an absolute number or a relative number. Note that this bit is ignored if the Output Frame is provided to QI by means of a compound frame.<br>ABS = 0 means calculate the number of buffers required for the input data (the integer greater than or equal to Input Length/POOL SIZE), add ADDBUF (0 or 1) to this number, and obtain that number of buffers from pool POOL ID.<br>ABS =1 means obtain the number of buffers in ADDBUF (0 or 1) from the pool POOL ID. |
| 24<br>ADDBUF | Indicates whether to allocate an additional buffer to what is otherwise required by FSGT and ABS. Note that this bit is ignored if the output frame is provided to QI by means of a compound frame.<br>• If ABS is cleared, ADD BUF specifies whether to add a buffer to the number of buffers required to hold the input data.<br>• If ABS is set, ADD BUF specifies whether to allocate a buffer or not. |
| 23-16<br>POOLID | This is the ID number of the pool from which to obtain output buffers. Note that this bit is ignored if the output frame is provided to QI by means of a compound frame. |
| 15-0<br>POOL_BUFFE R_SIZE | This is the pool buffer size. In order to be able to represent all values from 1 to 64k in 16 bits, POOL_BUFFER_SIZE = 0 is taken to mean 64k. (This is possible because a size of 0 makes no sense.) Also note that the LSB is assumed to be 0 so that only even buffer sizes are supported: 2, 4, 8, etc.<br>**NOTE:** This bit is ignored if a compound frame provides the output frame to SEC. |

Most of the preheader fields are used to provide the Queue Manager Interface with instructions for building an output frame, if needed, and selecting the format for the associated frame descriptor. These fields are discussed further in Frame descriptors.

## 6.2 Frame descriptors

A frame is some number of bytes of space or information stored in memory and represents room for or the actual data of a single message, packet, or protocol data unit. This space or data can be located in one or more buffers in memory. Multi-buffer frames use one or more tables, called scatter/gather tables, to keep track of buffer order, addresses, and associated information.

A Frame Descriptor (FD) is a standardized DPAA data structure that defines a frame's starting address in memory, data length, and other frame attributes. The FD is a vehicle to transfer frames between the QMan and SEC's QI through use of a QMan hardware portal that provides frame dequeue and enqueue services.

Note that FDs should not be confused with the job and shared descriptors used by SEC to describe a sequence of operations to be performed by SEC.

The FD includes a format field that describes the format of the corresponding frame. There are two major frame formats: simple and compound.

- A simple frame FD describes a single frame that may define the input to or the output of SEC data processing. Note, the simple frame format is also used for FDs that do not convey a frame.
- A compound frame FD describes two frames, one for input and another for output, using a two-entry scatter/gather table.

When QI dequeues a job with a compound frame FD, it enqueues the processed job with the same compound frame FD. When QI obtains a simple frame FD, it may enqueue the processed job with a new simple frame FD for the output frame or with a new compound frame FD that describes both the input and output frames. The preheader defined in Context_A of the dequeue summary specifies how to use frame data buffers for result return.

## 6.2.1 Processing simple frame jobs

Simple frame subformats include short, long, single buffer, and multi-buffer.

- Short frames support a 20b data length and a 9b offset field.
- Long frames support a 29b data length and no offset.
- Single buffer frames contain an address that points directly to a single buffer that contains the data to be processed by SEC as part of a job.
- Multi-buffer frames contain an address that points to a scatter/gather table (SGT). The entries of the SGT may be marked as unused (by setting the address, length, and BPID fields to 0), may point to single buffers, or may point to an SGT extension.

The rules for SEC processing of jobs dequeued as simple frames are:

- Unless the Preheader ABS = 1 and ADDBUF = 0, the QI builds an output frame.
- If the Preheader DIFREL = 0, input frame buffers are released to BMan. If DIFREL=1, the QI builds a compound frame scatter/gather table to enqueue the input frame with the output frame.

- The Preheader tells the QI how to create the output frame descriptor. The preheader ABS and ADDBUF fields address this situation. The combined values of ABS and ADDBUF are interpreted as follows:
  - {ABS, ADDBUF} = 00b: Request the number of buffers required to hold the input data. If TBPSIZ=0, all buffers are acquired from POOLID. If TBPSIZ!=0, the output frame scatter/gather table, if needed, is acquired from TBPID and the data buffers from POOLID.
  - {ABS, ADDBUF} = 01b: Request the number of buffers required to hold the input data, plus one more buffer. If TBPSIZ=0, all buffers are acquired from POOLID. If TBPSIZ!=0, the output frame scatter/gather table, if needed, is acquired from TBPID and the data buffers from POOLID.
  - {ABS, ADDBUF} = 10b: Do not request buffers for output.
  - {ABS, ADDBUF} = 11b: Request one buffer for output from POOLID.
  - If the number of buffers prescribed by ABS and ADDBUF is one and FSGT is set, one more buffer is added for an output frame scatter/gather table. That table buffer is acquired from POOLID or TBPID, depending on TBPSIZ.
- If the Preheader DIFREL=1, TBPSIZ and TBPID tell the QI to acquire a buffer for a compound frame scatter/gather table. If TBPSIZ=0, the buffer is acquired from buffer pool POOLID. If TPBSIZ !=0, scatter/gather table buffers are acquired from buffer pool TBPID.
- If BMan is unable to provide all of the buffers requested for a job, SEC terminates the job with an error. See Frame descriptor error handling for more details.
- POOL_BUFFER_SIZE field tells SEC the size of the buffers in buffer pool POOLID. This allows SEC to determine when its output will fit in a single buffer, or if a multi-buffer frame is needed.
- TBPSIZ tells SEC the size of buffers in buffer pool TBPID.

### 6.2.2 Processing compound frame jobs

Although the software overhead of using compound frames is somewhat higher than using simple frames, compound frames preserve the input frame instead of requiring that SEC releases the input frame buffers. This is useful for multi-cast and retransmission scenarios in which the original data needs to be retained for some interval even after a successful encrypt/decrypt operation.

Compound frames are used to convey an input frame and, optionally, an output frame to SEC. If the output frame is included, SEC does not need to interact with BMan at all. If the output frame is not included, SEC may need to build the output frame, depending on preheader setting.

The FD for a compound frame points to a two-entry SGT, where the first entry provides address, length, and offset information for output data, and the second entry provides the same information for the input. Other than its explicit ordering of output/input, the compound frame's SGT has the same format as the SGT used by multi-buffer frames; each entry may point directly to a buffer or to a secondary SGT. (For more information on compound frames see the DPAA overview chapter in the reference manual.)

The rules for SEC processing of compound frame jobs are:

- SEC never releases input frame buffers back to BMan. Software may need to do this using the input frame description in the compound SGT
- In the compound frame's SGT, the following combinations are possible:
    - Output frame address = X, Input frame address = Y. The output is written to the frame supplied output frame is large enough to hold SEC's output.
    - Output frame address = Y, Input frame address = Y. The output overwrites the input data buffers. Software must ensure that the data to be read in has, in fact, been read in before it is overwritten. For example, the descriptor could output some sort of header before reading the input frame. As a result, the header could overwrite the input frame before the input frame is read. One easy way to avoid this is to use a nonzero offset for the input frame but have the output frame use a smaller offset. Note that software must also ensure that the total size of the supplied input buffer(s) is large enough to hold SEC's output.
    - Output frame = Unspecified, Input address = Y. A unspecified frame is indicated by an unused SGT entry (an entry in which the Address, Length, and BPID fields are all zero). SEC obtains output buffers from BMan as prescribed by the preheader. Although the length field of the output FD entry in the compound frame is 30 bits, QI limits the output frame length to fit the length field of a frame descriptor, 20 or 29 bits depending on the frame format. Likewise, QI limits the output frame offset to 9 bits, even though the SGT entry allows 13 bits.

### 6.2.3 Frame descriptor error handling

SEC can experience errors while accessing preheaders and compound frames, allocating new frames for output, processing input and output frame data, and generating response FDs, as well as updating compound frame SGTs. Allocation errors may be due to buffer pool depletion in BMan. As explained in earlier chapters, SEC reports errors using the FD STATUS/CMD field. For a full list of detected error conditions see Job termination status/error codes.

Some errors are detected during QI's preparation of a job for processing, including memory access and buffer pool depletion errors. When these errors occur, QI sets the Do Not Run (DNR) bit in the job descriptor HEADER command to instruct DECO not to execute shared descriptor commands. When DECO gets the job, it may only release input buffers (if RBS is set in the SEQ IN PTR command), and, if present, DECO will read the shared descriptor and check whether to propagate and update the DNR setting in the shared descriptor HEADER command if the PD bit is set in the shared descriptor. In either case, no additional job processing occurs and QI reports the detected error.

If SEC processing requires only a single output buffer and the buffer could not be acquired (BMan reports pool depletion), QI will respond by returning the input frame to the service user with a buffer pool depletion error status in the STATUS/CMD field of the FD.

When multiple output buffers are required (or if the service user requested the unconditional generation of an SGT), QI checks whether the buffers in the buffer pool assigned to provide SGT buffers are large enough to accommodate the required number of SGT entries. If not, no buffers will be acquired and QI will return the input frame to the service user with a table buffer too small error status in the STATUS/CMD field of the FD.

Further multi-buffer frame error handling depends on whether part of the additional frame data buffers were acquired and associated SGT entries were already written to memory when the error condition is detected. If no part of the SGT has been written to memory, all buffers acquired for output are released and the input frame will be returned to the service user with the detected error condition reported in the STATUS/CMD field of the FD. If part of the SGT entries have already been written, that partially generated SGT buffer and the associated data buffers will not be released, the last utilized SGT entry will be rewritten to set the F (final) bit, and the partially generated (and empty) frame will be passed to the service user with LENGTH set to 0 and a non-0 status reporting the detected error condition. Any buffers acquired from BMan that have not yet been added as entries to the partially generated SGT buffer are released back to BMan.

Note that all QI detected frame write errors indicate that some or all of the content of the service user received multi-buffer or compound frame SGT is at least partially invalid, associated buffer attributes (including the buffer address) need to be treated as suspect, and thus the information conveyed by such FDs is merely useful for debug and analysis.

### 6.2.4 Job descriptor construction from frame descriptor

QI builds an internal SEC Job Descriptor (JD) to process service requests conveyed via DPAA Frame Descriptors (FDs). The precise length of the generated JD depends on the utilized memory address size and the inclusion of an optional load command. If SEC is configured to use 32-bit addresses, the generated JD is 8 words long without the load command or 10 words long with it. If addresses are greater than 32 bits, the JD is 11 words long without the load command or 13 words long with it. Note that the sum total of words for the internally generated JD and an optional (but usually present) Shared Descriptor (SD) must not exceed 64 words, i.e., the maximum size of an SD utilized by QI depends on the size of the internally generated JD, but it is always safe to limit the SD length utilized with QI to 51 words or less.

The internally generated JD is constructed from the following DECO commands:

| |
|---|
| HEADER |
| Shared Descriptor Pointer (1 word for 32-bit addressing, 2 words otherwise) (See Address pointers.) |
| SEQ OUT PTR<br>Output Pointer (1 word for 32-bit addressing, 2 words otherwise)<br>Output Length |
| SEQ IN PTR<br>Input Pointer (1 word for 32-bit addressing, 2 words otherwise)<br>Input Length |
| LOAD immediate (optional)<br>Immediate data value (FD CMD field) to be written to DPOVRD Register (optional) |

If there is no SD (the Preheader defines SDLEN as 0), the START WORD field in the HEADER command is set to 2 or 3 to skip over the 1 or 2 words reserved for the SD pointer.

The STATUS/CMD field in the FD allows software to modify the processing of individual frames while retaining the performance advantages of enqueuing to a frame queue for flow-based processing. The three most significant bits of the STATUS/CMD field of the FD are interpreted as follows:

**Table 6-3. SEC Frame Descriptor Command (CMD) Field Options**

| 3 MS bits of CMD Field | Append Load Immediate Command | Set Non-Seq ICID equal to Seq ICID | Replace Job Descriptor |
|---|---|---|---|
| 000b | | | |
| 001b | | | • |
| 010b | | • | |
| 011b | | • | • |
| 10xb | • | | |
| 11xb | • | • | |

If the STATUS/CMD field has the value 1xxb, QI adds a LOAD Immediate command to the end of the internally generated JD, followed by the contents of the STATUS/CMD field. The full 32 bits of the STATUS/CMD field are loaded to the DECO Protocol

Override Register (DPOVRD). The effect of loading specific values into DPOVRD depends on the selected protocol and is described in detail in the protocol section of this document.

If the STATUS/CMD field has the value x1xb, QI sets the Non-SEQ ICID equal to the SEQ ICID for the job. See the DECO ICID Status Register (DxISR) and the Queue Interface ICID Configuration Register (QIICR) for more information on ICIDs assigned to each job. Controlling QI access to frame queues and data also discusses ICID usage by SEC.

If the STATUS/CMD field has the value 0x1b, QI sets the RJD bit in the SEQ IN PTR command, which tells DECO to replace the internally constructed JD with an RJD fetched from the beginning of the input frame. Note that if a DECO Protocol Override is required, the user must place the LOAD Immediate command in the RJD. In addition, the service user must assure that the sum total of words for the SD and the RJD does not exceed 64 words.

If there is no SD (SDLEN=0), QI sets the INL bit in the SEQ IN PTR command, which tells DECO to execute an inline descriptor fetched from the beginning of the input data stream immediately after the SEQ IN PTR command is executed, i.e., any set DECO Protocol Override option will have no effect because the associated LOAD Immediate following the SEQ IN PTR command will not be executed. If needed, the LOAD Immediate command must be placed in the inline descriptor.

# Chapter 7
# Descriptors and descriptor commands

Software's primary interaction with SEC is through the submission of descriptors. To make SEC flexible, the descriptor is a program that controls SEC's operation. It is therefore up to the user to provide meaningful descriptors for execution. Descriptors are submitted to SEC in order to process a job, where a job can specify a variety of functions supported by SEC, from initialization of a security parameter, to generation of a random number, to encryption or signing of data, or full security protocol encapsulation of a packet.

Descriptors consist of commands that are executed in sequence, although conditional and unconditional jumps are available to alter the sequence. The size of a single descriptor is limited to 64 32-bit words, but it is possible to jump from one descriptor to another so that, in effect, much larger descriptors can be created. Only the first of these descriptors has to be submitted by means of the job ring or the queue interface; the rest are automatically fetched and executed by SEC.

Job descriptors, trusted descriptors, and shared descriptors can be modified and written back to memory. This is usually done when the processing of a data block is dependent on the result of processing of the prior data block. Such dependencies exist for information such as sequence numbers, counter values, and cryptographic state. Write backs are performed using descriptor commands. Hardware does not make independent decisions regarding the fields that should be written back.

Note that to correctly use sharing flows (wait or serial) in SEC, if one job in the flow updates the PDB in memory, all jobs in that flow must update the PDB in memory even if the PDB did not change for that particular packet. If all jobs in the flow update the PDB, SEC will ensure that subsequent jobs do not read the PDB from memory until all updates from prior jobs are complete.

When a job is submitted via the queue interface, the interface automatically creates a job descriptor. This job descriptor is built based on information provided via the submission. See Job descriptor construction from frame descriptor.

## 7.1 Job descriptors

**NOTE**

The term 'job descriptor' refers to both control structures created by software and submitted to a job ring and to equivalent control structures created by QI based on frame descriptor information.

A job descriptor (JD) is a control structure that causes SEC to execute a single job. Given a pointer to a job descriptor, the job queue controller will fetch from that address to the next burst boundary. If this is less than the number of bytes required to load an entire, maximum size, Job HEADER command, then the job queue controller will fetch the remaining required bytes. If the address size is 32 bits, then the maximum Job HEADER size is 12 bytes. If the address size is greater than 32 bits, then the maximum Job HEADER size is 16 bytes.

**NOTE**

As stated above, the first read is at least to the next burst boundary even though the descriptor may not be that long. It is up to the user to ensure that reading beyond the end of the job descriptor to the burst boundary will not result in any memory access errors.

If a second read is required to read the entire Job HEADER command, the second read is done from the first burst boundary to the second burst boundary. Note that these reads are all done with a single request and that request may be split under some conditions. Once these first words of the job descriptor are received, the job queue controller makes a decision. If there is a DECO available into which this job may be placed, the job is placed into the DECO for execution. If no such DECO is available, or if the job can't be placed into the DECO due to sharing constraints, the job queue controller will fetch the rest of the job descriptor if the previous reads did not already accomplish this. In addition, if there is a shared descriptor, the job queue controller will also fetch the shared descriptor unless it can be shared and is already present for another job. Once these reads have completed, the job will be eligible for placement into a DECO for execution. By prefetching all of this material, the job queue controller saves the DECO from taking the time to do so, thereby signficantly improving performance. If the job queue controller is fetching a shared descriptor, it will also attempt to prefetch the input frame for QI jobs. In the case where the input frame is a single buffer, the job queue controller will prefetch up to 128 bytes from the input frame, but no more than the length of the input frame. In the case where the input frame is a scatter/gather table, the job queue controller will prefetch the first 4 entries of the table. If the input frame is scattered and the output frame is also scattered, the job queue controller will prefetch the first 4 entries of the output table as well. Once these reads have completed, the job will be eligible for placement into a

DECO. Software-generated job descriptors contain the lengths and pointers (see Address pointers) of the data to be operated on, and either directly embed security keys and context or explicitly point to these keys and context. Keys and context can also be referenced indirectly by pointing to a shared descriptor (SD) that either contains keys and context, or includes pointers to keys and context. A job descriptor can include a shared descriptor by reference, but a shared descriptor cannot include a job descriptor.

Job descriptors use the descriptor commands defined in Using descriptor commands. A job descriptor always begins with a HEADER command. A job descriptor without a shared descriptor typically includes:

- Commands that specify the inputs (such as keys, IV, or data) to a cryptographic operation and where to place them
- Commands that specify where to place the output(s) of the operation
- One or more OPERATION commands that specify the cryptographic work to be done

The job descriptor may also contain MATH commands that perform various calculations and conditional JUMP commands that branch based upon the results of those calculations.

If the job descriptor references a shared descriptor, the memory address pointer to the shared descriptor immediately follows the job descriptor HEADER. In this case the OPERATION command and certain inputs (such as the key) are normally specified in the shared descriptor. The job descriptor typically specifies the location of the memory buffers for the input and output data. In this case, the job HEADER command has the REO (Reverse Execution Order) bit set so that the job descriptor commands execute first (to specify the input and output data buffers), followed by the shared descriptor commands (to specify the operations to be performed on these data buffers). (see Command execution order)

Because the length of the job descriptor is contained in the job HEADER command, no special termination command is required. When execution reaches the command which extends to the end of the job descriptor, DECO knows that the execution of the job descriptor has completed. Note that this endpoint is marked and does not change unless a new descriptor is loaded. Therefore, even if new descriptor material is loaded over the original material via MOVE or LOAD commands, the endpoint will not change and DECO will end execution of the job descriptor there. An error will be generated if DECO detects that the endpoint is inside a command. (For example, an error will be generated if the endpoint is between the words of a 2-word command.)

## 7.2 Trusted descriptors

A trusted descriptor is a job descriptor (possibly including a shared descriptor) that is integrity-checked at run time and is executed only if the check passes. This provides a mechanism to ensure that particularly sensitive operations are performed only by descriptors that were created by trusted software. Trusted descriptors have the following privileges not available to ordinary job descriptors:

- Access to trusted descriptor-only black keys (See Black keys)
- Access to trusted descriptor-only blobs (See Blobs)

Trusted descriptors allow trusted software to extend these privileges to untrusted software in a carefully controlled fashion. The trusted software can generate trusted descriptors that access specific privileged data objects in specific ways on behalf of specific requestors and deny access to other data objects, access modes, or requesters. Note that each Trusted Descriptor is associated with a particular SDID, and will run only if it is executed with the same SDID as the job ring in which the Trusted Descriptor was created. (The signature over the Trusted Descriptor will not validate if the SDID is different.) The Trusted Descriptor can be run in the job ring in which it was created, or another job ring, or can be run from the QI, as long as the SDID is correct. The only exception to this is Trusted Descriptors created in a job ring owned by TrustZone SecureWorld. These "TrustZone Trusted Descriptors" can be run in any job ring or can be run from the QI and will assert the SecureWorld signal when accessing memory. Note that the address pointers used in TrustZone Trusted Descriptors use physical addresses rather than intermediate physical addresses.

Trusted descriptors must be created, and are usually run, via jobs submitted via a Job Ring. To run a trusted job via QI, one of three indirect methods must be used. The first method is to have the specified shared descriptor JUMP to the trusted descriptor. The second method is to submit an inline descriptor that is a trusted descriptor. Note that neither of the first two methods allows the trusted descriptor to have a shared descriptor. The third method is to use a replacment job descriptor or a control replacement job descritor, which can have a shared descriptor.

Any descriptor can jump to a trusted descriptor via any of the various means: CRJD, RJD, nonlocal JUMP, or inline descriptor. However, while a trusted descriptor may use any of those means to jump, the target of such a jump must be another trusted descriptor. Otherwise, an error will be generated.

**NOTE**

In order to use the derived key protocol (DKP) in a trusted descriptor, the input and output for the protocol must both be via the sequence pointers. That is, the option selected must be from SEQ IN PTR to SEQ OUT PTR. There are no restrictions for other protocols.

## 7.3 Shared descriptors

Because descriptors can hold a lot of information required to process a job for a particular flow, they can be large, particularly if efficiency is maximized by placing the keys and other information within the descriptor rather than referencing them with pointers. To save overhead, SEC supports a shared descriptor mechanism. A shared descriptor is fetched once and held internally while it is used by several different related jobs. The keys and context information can also be shared among multiple descriptors. This saves bandwidth and latency, particularly when black keys are in use.

A shared descriptor (SD) is constructed with the expectation that it will be used for multiple jobs. The general usage model is to have a shared descriptor for each security session (for example, unidirectional IPsec tunnel). Every time a job related to that security session is required, SEC obtains job-specific information about the data (length, pointer) from the job descriptor and obtains its session context from the shared descriptor. Shared descriptors can store session state and can include commands to update this session state as needed. Shared descriptors are well suited for complex operations, as the software overhead of creating the shared descriptor is amortized over many individual jobs.

In order to optimize performance when a job descriptor references a shared descriptor, use the following guidelines. The job descriptor should only contain commands specific to one job in the sequence of jobs for which the shared descriptor will be used. Such commands include where to find the input data and where to place the output data. In addition, occasional tasks such as executing an RJD or overriding the normal operation of the shared descriptor would also be found here. The shared descriptor should contain all of the generic, flow-specific, commands. That is, references to keys, context, state, operations, etc. This is exactly the type of job descriptor automatically created by the QI.

Job descriptors indicate the presence of an associated shared descriptor by setting the SHR bit in the job descriptor HEADER command. Software creates shared descriptors using the same command set as all other types of descriptors. A shared descriptor always starts with a shared descriptor HEADER.

The following restrictions are specific to shared descriptors:

- A shared descriptor cannot have its own shared descriptor.
- A shared descriptor can be, at most, sixty-two 32-bit words if pointers are 32 bits and, at most, sixty-one 32-bit words if pointers are larger than 32 bits. This limit is

imposed because the job descriptor and the shared descriptor must both fit into the 64-word descriptor buffer (see Figure 7-2), and the minimum job descriptor consists of a one-word job descriptor HEADER and a pointer to the shared descriptor (See Address pointers). Note that larger jobs can be created by JUMPing to another job descriptor.

- Some bits in the shared descriptor HEADER and the job descriptor HEADER commands differ.
- The creation of a trusted descriptor involves signing the entire job descriptor, including a referenced shared descriptor, if any. As a result, shared descriptors are signed as part of the job descriptor when creating trusted descriptors. Therefore the final signature is never part of a shared descriptor. Note that the REO bit cannot be set in a trusted descriptor.

The following figure illustrates two descriptors that reference the same shared descriptor.

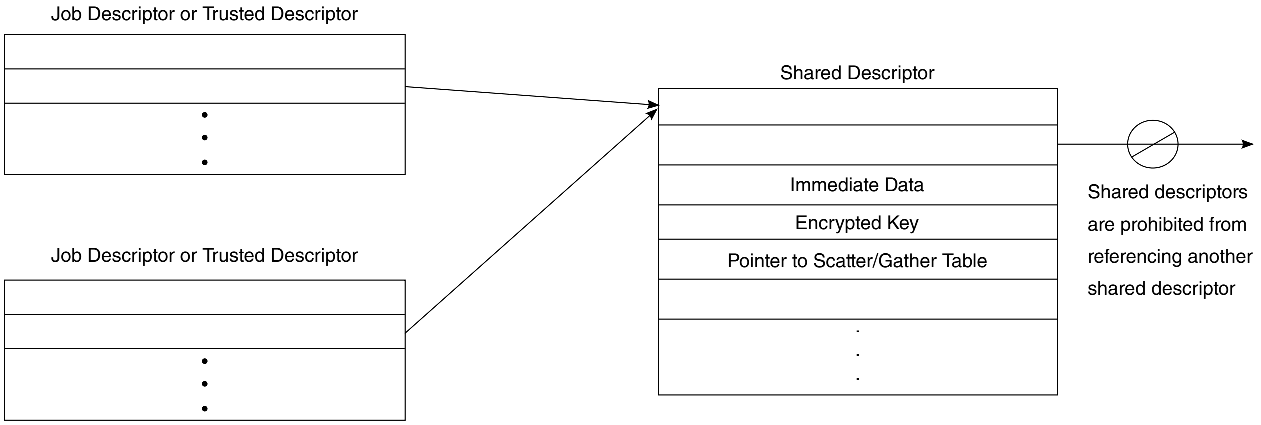

Job Descriptor or Trusted Descriptor
Job Descriptor or Trusted Descriptor
Shared Descriptor
Immediate Data
Encrypted Key
Pointer to Scatter/Gather Table
Shared descriptors are prohibited from referencing another shared descriptor

Figure 7-1. Two descriptors referencing the same shared descriptor

## 7.3.1 Executing shared descriptors in proper order

SEC provides mechanisms that can be used to ensure that jobs referencing the same shared descriptor execute in proper order. A shared descriptor may need to modify keys embedded within the descriptor, or particular fields of a protocol data block within the descriptor before a subsequent job uses the shared descriptor. Use the STORE command to update the shared descriptor. Note that to correctly use sharing flows (wait or serial), if one job in the flow updates the PDB in memory, all jobs in that flow must update the PDB in memory even if the PDB did not change for that particular packet. If all jobs in the flow update the PDB, SEC will ensure that subsequent jobs do not read the PDB from memory until all updates from prior jobs are complete.

### NOTE

If a NEVER share shared descriptor is modified during execution, and that modification is not written back to memory, the modification will NOT be seen by any other job which uses that shared descriptor. If a NEVER share shared descriptor is modified during execution, and if memory is updated with that change, subsequent jobs which reference that shared descriptor might have already fetched the original version or, if fetched during the update, might have a corrupted version of the shared descriptor. Therefore, it is up to the user to ensure that no jobs which use a NEVER share shared descriptor are in flight when the shared descriptor is updated. Clearly, NEVER share shared descriptors are not meant to be updated.

When a shared descriptor uses sequences, the sequence definitions should be in the job descriptor because the definitions can change from job descriptor to job descriptor. In such cases, set the REO bit in the job descriptor header. Note that setting the REO bit in the job descriptor header tells SEC to execute the job descriptor before the shared descriptor.

The sharing type may be changed to NEVER via a write to the DECO Control Register. Doing so prevents the descriptor from being shared from the DECO. The descriptor could be shared following a subsequent read from memory or from another DECO if that DECO has already gotten a copy of the descriptor. If the descriptor is being shared at the time the DECO Control Register is written to set the sharing type to NEVER, the descriptor will be shared.

SERIAL sharing can only take place by sharing the shared descriptor executing in a DECO with the next job to execute in that same DECO. For the same SERIAL shared descriptor to execute in a different DECO, it must be refetched from memory because it could not be shared with its prior execution.

A WAIT shared descriptor may be shared into the same, or a different, DECO. However, any instance of a WAIT shared descriptor may only be shared once. For example, if there are 3 jobs, X, Y, and Z, to be executed using the WAIT shared descriptor, once X has shared to Y, X can't share to Z. So, the sharing would be from X to Y, and then Y to Z. In other words, the WAIT shared descriptor may be shared many times, but each job descriptor can share it only once.

An ALWAYS shared descriptor may be shared as many times as necessary from the same job descriptor. This is because ALWAYS shared descriptors are stateless so that the order in which they are shared is not important. In the example above, X could share to both Y and Z.

### 7.3.2 Specifying different types of shared descriptor sharing

If two jobs are to be processed for the same data flow, they can share flow-specific data by referencing the same shared descriptor, which would be written to either reference or embed the flow-specific data. Sharing can be in parallel. i.e. two or more DECOs processing jobs using the same shared descriptor at the same time, or sharing can be sequential, i.e. a single DECO uses the same shared descriptor to process several jobs in a row without refetching the shared descriptor. "Self sharing" occurs when a descriptor is shared back into the same DECO. This can happen with "WAIT", "SERIAL", and "ALWAYS" sharing.

SEC distinguishes shared descriptors from each other by the address and ICID used to fetch the shared descriptor.

To share shared descriptors, the SHARE bits in the job descriptor header, and sometimes in the shared descriptor header itself, must be set. This lets SEC know under which circumstances the shared descriptors can be shared.

The following table shows the sharing possibilities supported by the HEADER command. The full details of the Shared Descriptor HEADER command can be found in HEADER command.

**Table 7-1. Interpretation of the SHARE fields**

| SHARE Name | Job Descriptor SHARE (binary) | Shared Descriptor SHARE (binary) | Description |
|---|---|---|---|
| NEVER | 000 | 00 | Never share the shared descriptor. Descriptors can execute in parallel, so no dependencies are allowed between them. Fetching the shared descriptor is repeated. |
| WAIT | 001 | 00 | Share the shared descriptor once set up has completed and processing has begun. Sharing can begin after a LOAD Command (or a PROTOCOL OPERATION) has set the OK to Share bit. Class 1 and Class 2 Key Registers are shared if valid. |
| SERIAL | 010 | 00 | Share once the descriptor has completed. The descriptor with which this should be shared will execute in the same DECO/CCB. Class 1 and Class 2 Key Registers are shared, if valid. Context may optionally be shared. |
| ALWAYS | 011 | 00 | Always share the shared descriptor, but keys are not shared. No dependencies can exist between the descriptors. |
| DEFER | 100 | 00: NEVER<br>01: WAIT<br>10: SERIAL<br>11: ALWAYS | Use the value of the SHARE bits in the shared descriptor to determine the type of sharing. |
| All other combinations are reserved | | | |

### 7.3.2.1 Error sharing

Shared descriptors can be marked as:

- NEVER
- WAIT
- SERIAL
- ALWAYS

When the job descriptor or shared descriptor is marked SERIAL or NEVER, no sharing can take place between DECOs. In the SERIAL case, only one DECO at a time is allowed to have a copy of the shared descriptor. In the NEVER case, each DECO receives a new version of the shared descriptor read from memory each time it runs a job referencing that shared descriptor. Note that the shared descriptor being marked as NEVER indicates that it is stateless (contains no information requiring update upon completion of a job). If a DECO reports an error while using the shared descriptor, there is no need to report that error to any other DECOs using an independent copy of the shared descriptor.

In cases where shared descriptor sharing occurs between DECOs, the first DECO to fetch the shared descriptor is the supplier DECO, and other DECOs using shared descriptor information from the supplier DECO's descriptor buffer are the consumer DECOs.

When the descriptor is shared between two jobs which run sequentially in the same DECO, no errors can be propagated.

In the ALWAYS case, errors do not propagate from supplier to consumers. If a supplier DECO has already started sharing the shared descriptor when an error occurs, the consumer DECO's job can complete normally regardless of the presence of an error in the supplier DECO.

In the WAIT case, an error in the supplier DECO can propagate to the consumer DECO while the shared descriptor and keys, if any, are being shared, causing both jobs to terminate with errors. The DECO Control Register can be written with value 0200h (that is, OFFSET = 02h and LENGTH = 00h) to enable error propagation, or 0300h (that is, OFFSET = 03h and LENGTH = 00h) to block error propagation (see value 06h, class 11 in Table 7-18). Using either of these values sets "OK to share" and tells the supplier DECO to propagate its shared descriptor and keys to the consumer DECO. Once the shared descriptor and keys (if any) have been copied to the consumer DECO, errors in the supplier DECO no longer affect the consumer DECO.

### 7.3.3 Changing shared descriptors

The best shared descriptors are independent, meaning that they do not need to be modified by software for each job descriptor with which they are used. (Note that this is a different topic than shared descriptors which update themselves.) Shared descriptors are more easily used the more generic they are. However, shared descriptors may have to be changed on occasion; for example, when there is a key change. Replacement job descriptors (see Using replacement job descriptors) can be used for such changes to avoid requiring software to make the change.

## 7.4 Using in-line descriptors

In the typical use case, the shared descriptor contains the main processing sequence. However, by setting the INL bit in a SEQ IN PTR command and providing appropriate address and length information, SEC is directed to an in-line descriptor, which is a job descriptor that software prepends to the data defined by an input sequence. (For more information about the SEQ IN PTR command, see SEQ vs non-SEQ commands and SEQ IN PTR command.)

Note that shared descriptors can point to in-line descriptors, but in-line descriptors cannot point to shared descriptors. This means that the in-line descriptor is loaded at the start of the descriptor buffer, overwriting as much shared descriptor, if one is present, and job descriptor, as needed. This means the shared descriptor will no longer be executable by this job and will no longer be shareable. Note that an in-line descriptor may be scattered by means of an SGT.

If SEC services are accessed via the Queue Manager, in-line descriptors are a method for instructing SEC to perform special processing on frames in a given flow. This is done by setting the shared descriptor length field to 0 in the preheader for the flow. In that case, QI will build the job descriptor for those frames with INL set in the SEQ IN PTR command.

Prepending the in-line descriptor to the input data can be accomplished in two ways:

- Exploiting empty space in the buffer provided by the upstream frame producer
- Creating the in-line descriptor in a new buffer, which is then placed at the head of the scatter/gather list for a multi-buffer frame descriptor.

Once the inline descriptor has been loaded, the detection of an error will result in a bit in the completion status being set which indicates that a non-local jump was taken. There is no indication of how many non-local jumps were made. For job ring jobs, the original job descriptor address is placed in the appropriate output ring.

Once an inline descriptor has been loaded, use of the STORE convenience sources for updating the job descriptor (41h and 45h) will result in an error.

In some cases, an inline descriptor will be used when a shared descriptor had been shared from a prior job. In such cases, it may be desirable to treat the job as if it had not been shared. This may be accomplished by writing to the CDS bit in the Clear Written Register.

## 7.5 Using replacement job descriptors

A replacement job descriptor (RJD) is an in-line descriptor that:

- Replaces the job descriptor that invoked the replacement descriptor. If SEC services are accessed via the Queue Manager, the internal job descriptor generated by the Queue Manager Interface (QI) is replaced.
- Does not replace the existing shared descriptor

To invoke the replacement job descriptor, execute a SEQ IN PTR with RJD = 1. This immediately executes the replacement job descriptor. Note that the replacement job descriptor must be at the start of the input sequence data at the time that this SEQ IN PTR command is executed.

The replacement job descriptor can modify the shared descriptor before allowing it to execute. This allows operations such as changing the keys and resetting the sequence number within a shared descriptor, such as for an IPSEC PROTOCOL OPERATION), without having to interrupt the flow of packets. However, because the shared descriptor has already been loaded, the length and address of the shared descriptor must not be modified. Note that when there is no shared descriptor, there is no difference between an in-line descriptor and a replacement job descriptor.

When using the replacement job descriptor capability, the current job descriptor can be replaced with any job descriptor. Other data, including an input frame, can follow the replacement job descriptor in the input sequence data. For example, an IPSec flow can modify the keys or sequence number and then immediately process the packet which follows the replacement job descriptor.

If there is a JUMP HALT command in the replacement job descriptor, the job terminates without executing the shared descriptor. Otherwise, if the job descriptor has the REO bit set (jobs from QI always do), once the replacement job descriptor has finished, execution continues with the shared descriptor so that data can be processed. If the shared descriptor will process data during this job, before beginning that processing make sure that all the updates made to the shared descriptor have completed both internally and externally (that is, the update to the descriptor buffer has completed and the update to the shared descriptor in memory has completed). This is discussed in the following two paragraphs.

The replacement job descriptor can insert new values in the shared descriptor with either the MOVE command or the LOAD command. The MOVE command's default behavior is to schedule the MOVE operation as soon as possible and then allow the next command to execute. As a result, the MOVE happens in parallel with subsequent commands. Be aware that the MOVE command can take multiple cycles to complete, and it is possible that shared descriptor commands may be executed before the MOVE completes. This could result in the intended updates not being used. If there is a chance that this may occur, use the WC bit in the MOVE command to ensure correct operation. See MOVE, MOVEB, MOVEDW, and MOVE_LEN commands for additional details about the MOVE command.

If using the LOAD command to modify the shared descriptor, the replacement job descriptor should use the JUMP command, waiting for the NIP (No Input Pending) bit to evaluate true before proceeding. Note that the replacement job descriptor can also be used to transfer data to other destinations, such as memory, context registers, or Math registers. It is the replacement job descriptor's responsibility to ensure that any and all of these transfers have completed before the shared descriptor uses the new data.

Replacement job descriptors can be trusted descriptors, and they must be trusted if the current descriptor is a trusted descriptor. Although trusted descriptors cannot be run directly through QI, trusted descriptors can be indirectly executed using a replacement job descriptor.

Once an RJD has been loaded, use of the STORE convenience sources for updating the job descriptor (41h and 45h) will result in an error.

The detection of an error will result in a bit in the completion status being set which indicates that a non-local jump was taken. There is no indication of how many non-local jumps were made. For job ring jobs, the original job descriptor address is placed in the appropriate output ring.

In some cases, an RJD descriptor will be used when a shared descriptor had been shared from a prior job. In such cases, it may be desirable to treat the job as if it had not been shared. This may be accomplished by writing to the CDS bit in the Clear Written Register.

Due to features of the AES encryption algorithm, special handling may be required when using a replacement job descriptor to update a key in a shared descriptor. AES encryption requires each block of data to be processed in a series of cryptographic rounds, and the AES key is successively modified at each round. When decrypting, the AES CHA must start with the fully modified form of the key (also called a decryption key or decap key) and reverse the modifications at each round, eventually ending up with the original encryption key. If the descriptor was shared, AES will have left the decryption key in the key register. However, when using a replacement job descriptor to update a key in a shared descriptor, the updated key is usually an encryption key (also called an encap key). To resolve this problem, use the LOAD command to clear the fact that the shared descriptor was, in fact, shared. That way, AES will expect the encryption key and will automatically generate the decryption key. This avoids having to generate the decryption key as part of the RJD.

## 7.6 Scatter/gather tables (SGTs)

When submitting jobs to SEC, software can create job descriptors or frame descriptors with address/length entries that point directly to data or indirectly to data by means of scatter/gather tables. An SGT consists of one or more SGT entries. The final entry in the SGT is marked by setting the F (Final) bit in an SGT entry.

Each of the SGT entries occupies four 32-bit words, as seen in Table 7-2. Note that an SGT entry with Length = 0 is legal. When this is the setting, no data will be read from or written to the buffer pointed to by the Address Pointer. If Length = 0, BPID = 0, and Address Pointer = 0, this is considered an unused SGT entry and no buffer release actions for such entries will take place.

An entry can point to another SGT that contains additional entries by setting the E (Extension) bit in the entry. When the E bit is set, SEC fetches SGT entries from the new SGT and ignores any remaining entries in the old SGT.

**NOTE**

An SGT with the E bit set in the first entry is considered malformed.

**NOTE**

SGT entries are read from memory four entries at a time. If the number of entries in the table is not a multiple of four, then the read of the entries will go beyond the end of the table. The user is responsible for ensuring that such reads beyond the end of the table do not result in memory access errors.

The following table shows the SGT entry format.

**Table 7-2. Scatter/gather table entry format**

| | | |
|---|---|---|
| word 0 | Reserved [24 bits] (must be 0) | Address Pointer [MS 8 bits] |
| word 1 | Address Pointer [LS 32 bits] | |

*Table continues on the next page...*

**Table 7-2. Scatter/gather table entry format (continued)**

<table>
<tr><td>word 2</td><td>E [1b]</td><td>F [1b]</td><td colspan="4">Length [30 bits]</td></tr>
<tr><td>word 3</td><td colspan="3">Reserved [8 bits]<br>(must be 0)</td><td>BPID [8 bits]</td><td>Reserved [3 bits]<br>(must be 0)</td><td>Offset [13 bits]</td></tr>
</table>

**Table 7-3. Scatter/gather table field descriptions**

| Field | Description |
|---|---|
| Offset | Offset (measured in bytes) into memory where significant data is to be found. The use of an offset permits reuse of a memory buffer without recalculating the address. |
| BPID | Buffer Pool ID. Indicates the identifier of the buffer pool that owns the buffer referred to by the address pointer |
| Length | Length of significant data in buffer, measured in bytes |
| F | Final Bit. If set to 1, this is the last entry in the scatter/gather table. |
| E | Extension bit. If set to 1, the address pointer points to a scatter/gather table entry instead of a memory buffer. In this case the Length and Offset fields are ignored, and this entry is regarded as the last entry of the current scatter/gather table even if the F bit is 0. The next table entry is taken from the scatter/gather table pointed to by the address pointer.<br>**NOTE:** It is an error to set the E bit if the SGT entry is unused (i.e. Length, BPID and Address Pointer all 0s). |
| Address Pointer | Pointer to memory buffer or discontinuous scatter/gather table entry, depending on the E bit. |
| Reserved | Field not currently defined. Leave these bits 0 to ensure forward compatibility. |

QI jobs use only 29 or 20 bits of the length field and 9 bits of the offset field. If the offset is 0, 29 bits of length can be used. These restrictions are made to ensure that a scatter/gather table entry can be coded in a frame descriptor format.

## 7.7 Using descriptor commands

Descriptors contain one or more commands that tell SEC what operations to perform, as well as what data on which to operate. Commands can also be used to enforce data type separation. For example, specifying that input data be treated as a cryptographic key forces SEC to treat it exclusively as a key and prevents the key from being written back out into memory in unencrypted form.

SEC permits a great deal of flexibility in composing descriptors, but it is highly recommended that descriptors be modeled after the examples provided in the reference software. Some sequences of commands or combinations of command options may produce unexpected results.

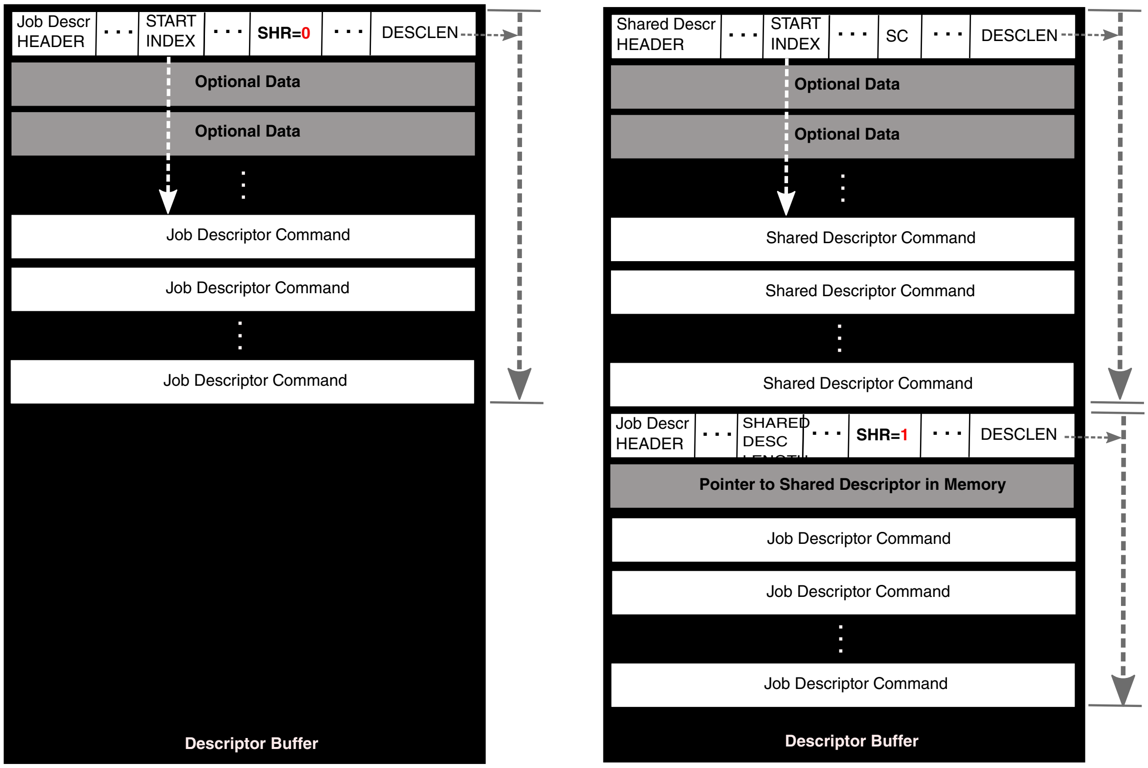
### 7.7.1 Command execution order

**NOTE**

In the following discussion, the term "Job Descriptor" should be taken to include both job descriptors and trusted descriptors.

Before a job descriptor begins execution, the portion of the job descriptor contained in the holding tank is loaded into the descriptor buffer. This includes the job descriptor's HEADER command (see HEADER command), which is the first command executed. Once the remainder of the job descriptor has finished loading, the next command to execute depends upon three fields in the HEADER: SHR, REO, and START INDEX.

The following figure shows the layouts for job descriptors depending on whether SHR = 0 or 1.


Job Descr HEADER
. . .
START INDEX
. . .
SHR=0
. . .
DESCLEN
Optional Data
Optional Data
Job Descriptor Command
Job Descriptor Command
Job Descriptor Command
Descriptor Buffer
Shared Descr HEADER
. . .
START INDEX
. . .
SC
. . .
DESCLEN
Optional Data
Optional Data
Shared Descriptor Command
Shared Descriptor Command
Shared Descriptor Command
Job Descr HEADER
. . .
SHARED DESC LENGTH
. . .
SHR=1
. . .
DESCLEN
Pointer to Shared Descriptor in Memory
Job Descriptor Command
Job Descriptor Command
Job Descriptor Command
Descriptor Buffer

#### 7.7.1.1 Executing commands when SHR = 0

When SHR = 0, the job descriptor does not reference a shared descriptor. Therefore, the HEADER's START INDEX field specifies the position of the next command that will execute within the job descriptor. If the START INDEX value is 0, the next command to execute is the command immediately following the HEADER command. Any other value causes a jump to the position indicated by the START INDEX field. Note that within a protocol job descriptor, the START INDEX value is used to skip over the PDB, if any. Before the job descriptor continues execution, the remainder of the job descriptor is fetched from memory and loaded into the DECO's descriptor buffer. The left half of Figure 7-2 shows the layout of a job descriptor that does not reference a shared descriptor.

Commands execute in the order in which they appear in the descriptor buffer until one of the following is executed:

- The last command in the job descriptor
- A JUMP command when the JUMP is taken (see JUMP (HALT) command)
- An in-line descriptor (see Using in-line descriptors)
- A replacement job descriptor (see Using replacement job descriptors)

When JUMP commands are executed, the behavior is as follows:

- If DECO executes an unconditional halt type of JUMP or a conditional halt type whose tested condition evaluates to true, execution of the job descriptor terminates.
- If DECO executes a JUMP whose type is conditional halt, local conditional jump, non-local conditional jump, conditional subroutine call, or conditional subroutine return and the tested condition evaluates to false, execution continues with the command following the JUMP.
- If the JUMP type is local or non-local jump or conditional subroutine call and the tested condition evaluates as true, the command indicated by the LOCAL OFFSET field (for local jumps) or by the Pointer Field (for non-local jumps) is the next command to execute.
  - If the jump is local, the target of the JUMP should be within the current job descriptor. If the target is beyond the end of the current job descriptor, it is up to the programmer to ensure there is executable code at the target and that the descriptor will be able to terminate properly. One common method for proper termination is to use a JUMP HALT command.

- If the jump is non-local, the target of the JUMP must be the start of a job descriptor.
- If the JUMP type is conditional subroutine return and the tested condition evaluates as true, the next command to execute is the command following the most recently executed conditional subroutine call.

### 7.7.1.2 Executing commands when SHR = 1

As described in Executing commands when SHR = 0, the portion of the job descriptor (including the HEADER command) contained in the holding tank is loaded into the descriptor buffer. When SHR = 1, the job descriptor references a shared descriptor (see Shared descriptors).

In this case, instead of a START INDEX field, the job descriptor HEADER contains a SHR DESCR LENGTH field. This field specifies the length of the shared descriptor, which allows DECO to leave enough space for the shared descriptor when the job descriptor is loaded into the descriptor buffer. The right side of Figure 7-2 shows the layout of a job descriptor that references a shared descriptor.

A pointer to the shared descriptor's location in memory appears in the word (or two words) immediately following the job descriptor HEADER. Note that the job descriptor HEADER may occupy two words in addition to the shared descriptor address (see EXT field in HEADER command). The pointer, together with the ICID that was used when fetching the descriptor, is used to determine if the shared descriptor is already resident in the same or another DECO and is therefore a candidate for sharing. If the shared descriptor is not resident or cannot be shared, the shared descriptor is fetched from memory using the pointer as the starting address. If the shared descriptor is available from another DECO, processing cannot continue until the shared descriptor (and the keys, if the keys are to be copied) has been copied from the other DECO. Processing cannot continue until the entire shared descriptor is present. The START INDEX field within the shared descriptor's HEADER specifies the position of the next command that will execute within the shared descriptor once the shared descriptor begins execution. If either the shared or job descriptor contain a PROTOCOL OPERATION command, the START INDEX value in the shared descriptor is used to skip over the PDB, if any. Note that when a shared descriptor is present, the PDB is always in the shared descriptor even if the PROTOCOL OPERATION is in the job descriptor.

#### 7.7.1.3 Executing commands when REO = 0

If the job descriptor references a shared descriptor, the REO bit in the HEADER command determines the next command to be executed. [1] When REO = 0, DECO executes the shared descriptor before the remainder of the job descriptor, as illustrated on the left of Figure 7-3. After the job descriptor HEADER executes, the HEADER command within the shared descriptor (2.0 in the diagram) is the next command to execute. The commands within the shared descriptor then execute. Once the shared descriptor starts executing, any job descriptor HEADER command will be treated as a no-op until a new job descriptor is loaded.

The shared descriptor ceases executing when any of the following occurs:

- an in-line job descriptor is executed
- a replacement job descriptor is executed
- a JUMP HALT command is executed
- a non-local JUMP is executed
- or the shared descriptor "falls through" to the job descriptor

Since the shared descriptor immediately precedes the job descriptor in the descriptor buffer (see right side of Figure 7-2), unless the last command of the shared descriptor causes a jump, the shared descriptor may complete by simply "falling through" to the job descriptor. Once the shared descriptor completes, DECO executes the job descriptor, starting with the job descriptor HEADER, which will be treated as a no-op. Execution will continue with the next command of the job descriptor. Execution will end following execution of the last command in the job descriptor unless the last command is a taken JUMP.

Once the shared descriptor HEADER command has been executed, any further shared descriptor HEADER commands will be used as absolute, unconditional, jump commands if the START INDEX field is nonzero. The START INDEX field will be used to determine the target. Note that, unlike the JUMP command, the START INDEX is the value of the target index, not a relative index. No other fields in the shared descriptor HEADER command will cause an action to take place although error conditions may be triggered. If a subsequent execution of a shared descriptor HEADER command is done where the START INDEX is zero, then the shared descriptor HEADER command will be treated as a no-op.

It is important to note the difference in how subsequent job descriptor HEADER commands are handled when REO=0 and REO=1.

[1] Note that the REO bit cannot be set in a trusted descriptor.

### 7.7.1.4 Executing commands when REO = 1

When REO is 1, DECO executes the job descriptor before the shared descriptor, as illustrated in the diagram on the right of Figure 7-3. In this case, the job descriptor command (if any) that immediately follows the shared descriptor pointer (1.1 in the diagram), or the extended HEADER word if it is present, executes immediately after the job descriptor HEADER. After the job descriptor completes, DECO then executes the shared descriptor commands, starting with the shared descriptor HEADER (2.0 in the diagram).

Execution of a subsequent job descriptor HEADER, other than one reached via a non-local JUMP, an RJD or inline descriptor, will terminate execution normally. Upon execution of the command which ended the job descriptor, no matter how many times this occurs, with the exception of taken JUMPs, execution will continue with the command at the start of the descriptor buffer. After the first execution of the shared descriptor HEADER, subsequent executions can be used as absolute, unconditional, jumps in the same manner as subsequent shared descriptor HEADER commands are used when REO=0.

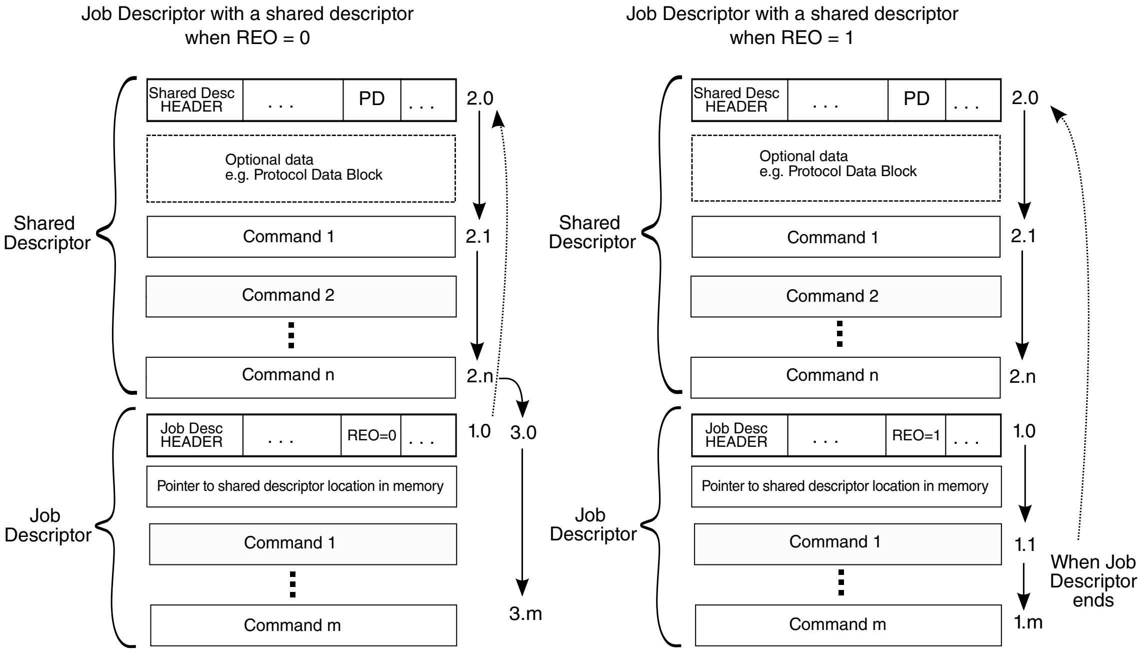


**Figure 7-3. Order of command execution if a shared descriptor is referenced**

### 7.7.1.5 Executing additional HEADER commands

A job descriptor must start with a job descriptor header, and a shared descriptor must start with a shared descriptor header. These are typically the only HEADER commands within a descriptor, but it is possible for the descriptor to have additional HEADER commands.

No error is generated if a job descriptor or shared descriptor executes additional shared descriptor HEADER commands. These are essentially no-ops, with one exception. If START INDEX is non-zero, the Shared Descriptor HEADER command causes a jump to that position within the descriptor buffer. That is, the Shared Descriptor HEADER command executes as if it is an unconditional JUMP to an absolute index. Note that this is different from the JUMP command, which uses relative addressing. The first shared descriptor HEADER command is the one that is treated as real. All subsequent shared descriptor HEADER commands executed, including the first one if executed again, are no-ops (other than an absolute jump if the START INDEX is nonzero).

**NOTE**

It is an error to execute a shared descriptor HEADER command in a job descriptor when there is no shared descriptor (when SHR= 0).

If a job descriptor does not reference a shared descriptor, any additional job descriptor HEADER commands that it executes (for example, by jumping back to the beginning of the job descriptor) are treated as jumps to an absolute address within the descriptor buffer. If a job descriptor does reference a shared descriptor, any additional job descriptor HEADER commands that it executes are treated as no-ops. Executing a job descriptor HEADER command within a shared descriptor terminates the shared descriptor if the shared descriptor runs after the job descriptor runs (that is, REO = 1), but the job descriptor header acts as a no-op if the shared descriptor runs before the job descriptor runs (that is, REO = 0).

### 7.7.1.6 Jumping to another job descriptor

Note that either a job descriptor or a shared descriptor can execute a non-local JUMP to a job descriptor. In these cases, the current job descriptor or shared descriptor terminates, and the new job descriptor is fetched into the descriptor buffer and executes. Note that this new job descriptor is not permitted to reference a shared descriptor, but can also execute a non-local JUMP to another job descriptor. This mechanism allows the construction of jobs that are larger than the descriptor buffer. Once the entire chain of job descriptors terminates, a single job termination status word (see Job termination status/error codes) is returned. The return status is as if the original job descriptor had completed. That is, for job ring jobs, the original job descriptor address is placed in the appropriate output ring. If an error is detected following a jump to another descriptor, a bit in the status indicates that a non-local jump was taken. Note that there is no indication of how many non-local jumps were made.

## 7.7.2 Command properties

Three properties determine how SEC handles each command:

- Blocking
- Load/store checkpoint
- Done checkpoint

### 7.7.2.1 Blocking commands

A blocking command must complete before the next command can begin. Note that the completion is from the standpoint of the DECO. If the command requires a read and the DECO has scheduled the read, the next command can begin even if the read has not completed.

Many commands are blocking commands. The notable exceptions are commands that perform LOADs, STOREs, MOVEs and OPERATION algorithm commands. (That is, not PROTOCOL or PKHA OPERATIONS, which are blocking.) Note that setting the WC bit in any of the MOVE commands cause the command to become blocking.

### 7.7.2.2 Load/store checkpoint

If a command is a load/store checkpoint, it must wait for certain prior LOADs and/or STOREs to complete before it can start. This property ensures that LOADs, STOREs, and other commands occur in proper order.

### 7.7.2.3 Done checkpoint

If a command is a done checkpoint, it must wait until all current cryptographic activity associated with the descriptor is done. The CHAs signal done once their computation has completed. Note that this is different from the descriptor being done, since not all loads and stores may have completed. It merely indicates that the CHAs in use have completed their current tasks. Note that done checkpoints can be for only Class 1, or only Class 2, or both Class 1 and Class2.

### 7.7.3 Command types

The following is a list of commands that are supported in SEC along with their blocking and checkpoint properties.

Table 7-4. List of command types

| Command name | CTYPE | Blocking | Load/store checkpoint | Done checkpoint | See section/page |
|---|---|---|---|---|---|
| KEY<br>(& SEQ KEY) | 00000<br>(00001) | Yes, if not immediate or if encrypted | Load/Store, if not immediate or if encrypted | Yes | KEY commands |
| LOAD<br>(& SEQ LOAD) | 00010<br>(00011) | No | Load, for some destinations | No | LOAD commands |
| FIFO LOAD<br>(& SEQ FIFO LOAD) | 00100<br>(00101) | No | Load, if immediate and another FIFO LOAD is pending or if not immediate and an immediate FIFO LOAD or MOVE to the input FIFO is pending | No | FIFO LOAD command |
| STORE<br>(& SEQ STORE) | 01010<br>(01011) | No | if storing the checksum or if storing a scatter/gather table and that table is still being loaded | If from a Context Register the corresponding CHA must be done | STORE command |
| FIFO STORE<br>(& SEQ FIFO STORE) | 01100<br>(01101) | No | Load checkpoint if encrypting | Yes if encrypting | FIFO STORE command |
| MOVE<br>(& MOVE_LEN)<br>(& MOVEB)<br>(& MOVEW) | 01111<br>(01110) | Yes, if WC set | Load or Store depending on type of MOVE. It is a checkpoint if the CCB DMA is being used for a prior MOVE or for moving IMM data for KEY, LOAD, or FIFO LOAD commands. | If from a Context Register the corresponding CHA must be done | MOVE, MOVEB, MOVEDW, and MOVE_LEN commands |
| OPERATION<br>(ALGORITHM OPERATION)<br>(PROTOCOL OPERATION)<br>(PKHA OPERATION) | 10000 | Yes, if PKHA or protocol | For PKHA | No | ALGORITHM OPERATION command<br>PROTOCOL OPERATION commands<br>PKHA OPERATION command |
| SIGNATURE | 10010 | Yes, when verifying or re-signing | Yes if recomputing signature following execution; no pending reads or writes. | No | SIGNATURE command |

*Table continues on the next page...*

Table 7-4. List of command types (continued)

| Command name | CTYPE | Blocking | Load/store checkpoint | Done checkpoint | See section/page |
|---|---|---|---|---|---|
| JUMP | 10100 | Yes | Checkpoint based upon condition bits | If Class bit or bits are set | JUMP (HALT) command |
| MATH and MATHI | 10101<br>11101 | Yes | Will wait for data if SRC1 is Input or Output Data FIFO and data is not yet available | No | MATH and MATHI Commands |
| Job descriptor HEADER<br>Shared Descriptor HEADER | 10110<br>10111 | Yes | N/A | N/A | HEADER command |
| ECPARAM | 11100 | No | Load if an immediate FIFO LOAD or MOVE to the input FIFO is pending | No | see ECPARAM command |
| SEQ IN PTR | 11110 | Yes | Yes if there is a pending gather table read. For QI jobs, then also yes if buffer releasing remains to be done for a prior SEQ IN PTR command. | No | SEQ IN PTR command |
| SEQ OUT PTR | 11111 | Yes | Yes if there is a pending scatter table read. | No | SEQ OUT PTR command |

## 7.7.4 SEQ vs non-SEQ commands

SEC can process networking protocol packets that consist of separate fields, such as headers, sequence numbers, AADs, payloads, and ICVs. (A complete discussion of network security protocol packet formats is beyond the scope of this document. However, examples using the protocols that SEC supports can be found in Protocol acceleration.) To help process such packets efficiently, SEC provides sequence (SEQ) versions of the following descriptor commands:

- KEY
- LOAD
- STORE
- FIFO LOAD
- FIFO STORE

SEQ and non-SEQ versions of descriptor commands have nearly identical functions, with the major distinction being that the SEQ versions do not require pointers because SEC uses sequence addresses that were defined by previously executed SEQ IN PTR or SEQ OUT PTR commands. Another difference is that SEQ commands (with the exception of SEQ STORE) do not have immediate data modes.

For jobs submitted by means of the job ring interface, each job ring can be configured via the INCL_SEQ_OUT field in each Job Ring Configuration Register to output an additional word in each entry of the Output Ring that indicates the length of the output sequence (that is, the number of bytes output via SEQ commands).

### 7.7.4.1 Creating a sequence

Sequences are generally associated with shared descriptors (see Shared descriptors) which support a one-time definition of a set of commands to be performed on each packet in a flow. The address and length of the input and output packets are usually specified in a job descriptor (see Job descriptors) that references a shared descriptor containing SEQ-version commands to indicate how to process the data. The shared descriptor is analogous to a subroutine, and the job descriptor is analogous to a software program supplying arguments and then calling that subroutine.

The job descriptor uses the following commands to provide information about the data to be processed by the sequence:

- A SEQ IN PTR command to specify the length and address of the data to be processed (see SEQ IN PTR command).
- A SEQ OUT PTR command to specify the length and address of the buffer for the output data (see SEQ OUT PTR command).

The SEQ IN PTR and SEQ OUT PTR commands each have an SGF field which, when set to 1, allows sequence input and/or output areas to be defined by means of scatter/gather tables.

The SEQ IN PTR command is used to create an input sequence. The SEQ OUT PTR is used to create an output sequence. Once the input and/or output sequence pointers have been set, subsequent SEQ commands indicate how to process the packet. The length of the sequence may be extended by issuing additional SEQ IN PTR and SEQ OUT PTR commands with the PRE bit set (see Table 7-97 and Table 7-99) or by using MATH or MATHI commands to add length directly. DECO tracks how far into the output sequence DECO has progressed, and this information is used if a rewind is needed so that a second pass can be made over the sequence (see RTO field and SOP field in the SEQ IN PTR command and REW field in the SEQ OUT PTR command).

An input sequence ends when any of the following occurs:

- All specified input data is consumed (unless a rewind is then done).
- A new input sequence is started.
- An error occurs.

An output sequence ends when any of the following occurs:

- All specified output space is consumed (unless a rewind is then done).
- A new output sequence is started.
- An error occurs.

There can be at most one scatter/gather table active for input and at most one scatter/gather table active for output in the DECO at any time. Note that non-sequential commands can be executed within the same descriptor while a sequence is running. However, an input gather table can be in use by either an input sequence or by non-SEQ KEY, LOAD, or FIFO LOAD commands, but not both. Likewise, an output scatter table can be in use by either an output sequence or by non-SEQ STORE or FIFO STORE commands, but not both.

To accelerate performance, SEC caches gather table and scatter table entries in registers ( see the Gather Table Register (DxGTR) and Scatter Table Register (DxSTR)).

**NOTE**

If a scatter/gather table is being used for an input or output sequence, and a non-SEQ command references a second scatter/gather table for input or output data, entries from the second scatter/gather table overwrite the entries from the initial scatter/gather table. This can result in the input/output sequence referencing the wrong data. (The opposite case is not a problem. For example, a non-SEQ LOAD command which references a scatter/gather table followed by a SEQ IN PTR which references a scatter/gather table won't be an issue since the LOAD has to complete the use of its scatter/gather table before the SEQ IN PTR command can execute.)

**NOTE**

Hardware does not flag overwriting the scatter/gather table as an error. The descriptor programmer must ensure this does not happen.

### 7.7.4.2 Using sequences for fixed and variable length data

Some SEQ commands act on fixed length data (for example, keys, IVs, or packet header fields) whereas other SEQ commands act on data that changes length from packet to packet, such as packet payload. The VLF bit found in all SEQ commands indicates whether the data associated with the SEQ command is a constant length or whether the length is to be found in the corresponding variable length registers (VSIL and VSOL).

Note that the VSIL and VSOL are not accessible through the register bus, but rather they are read from or written to by means of the MATH or MATHI Command (see Table 7-93). This allows commands within the descriptor to calculate variable lengths. Given a total packet length (from an internally or externally generated job descriptor), the descriptor can calculate the variable length portion of a job and load it into the Variable Length Registers to be referenced by subsequent SEQ commands (by setting the VLF bit).

### 7.7.4.3 Transferring meta data

When processing data, SEC typically uses the DMA to read input data and write output data. Because SEC is primarily intended to accelerate cryptographic operations, the output data is normally different from the input data. However, it is possible to use SEC's external DMA to transfer data from an input buffer to an output buffer without modifying the data (that is, the identity transformation, also called null encryption), typically to either:

- Benefit from SEC's scatter/gather capabilities
- Transfer meta data in conjunction with cryptographic processing

The latter case is often useful because the meta data may describe the type, the source, the destination, the classification, the priority and/or the amount of the cryptographic data. If this meta data appears ahead of the cryptographically processed data, it is called 'leading meta data'. If it appears after, it is called 'trailing meta data'.

Three different tasks must be scheduled in order to transfer meta data from input to output without modification:

- Data must be read. Most often, the data is read into the input data FIFO. FIFO LOAD and SEQ FIFO LOAD are the most common methods for getting data into the input FIFO.
- Data must be stored. Most often, the data is stored from the output FIFO. FIFO STORE and SEQ FIFO STORE are the most common methods for storing data from the output FIFO.
- If the data is brought into the input data FIFO, it must be moved, via one of the MOVE commands, to the output FIFO.

While it is possible to transfer meta data without going through the input FIFO and output FIFO, such transfer methods are discouraged as timing can be complex. Furthermore, meta data is most often used with sequences so that multiple pointers don't have to be specified.

It is possible to accomplish two or three of the above tasks using a single command. This command is the SEQ FIFO STORE command with the meta data Output Data Type, 3Eh. Depending on how the auxiliary bits are set, this type of SEQ FIFO STORE will adjust the various lengths and obtain the data either from the input frame or from the input data FIFO.

The above procdures work for leading meta data. To handle trailing meta data for a sequence, start by subtracting the length of the meta data from the Sequence Input Length Register. Then, process the input frame. Once the processing is complete, add the length of the meta data back to the Sequence Input Length Register and handle the meta data. (Note that if using SEQ FIFO STORE with meta data Output Data Type, you don't need to add the length back into the register.)

#### 7.7.4.4 Rewinding a sequence

Note that it is possible to rewind a sequence to make an additional pass over the input and output data (see RTO field and SOP field in SEQ IN PTR command and REW field in SEQ OUT PTR command). A rewind can fill in data that was skipped over in a previous pass. For example, a rewind may be necessary if a field contains a hash value that is computed over data that appears later in the output data.

The following built-in protocol operations perform rewinds:

- TLS Decapsulation rewinds the input frame.
- IPSec Decapsulation rewinds the output frame.
- DCRC Encapsulation and Decapsulation rewind the output frame.

Note that owing to hardware limitations, the amount of leading meta data is limited to $2^{15}$ bytes when using any of these built-in protocols.

An error will be generated if the SEQ IN PTR command is used to rewind the input frame after buffers have been released.

### 7.7.5 Information FIFO entries

Each CCB has an iNformation FIFO (NFIFO). The NFIFO holds entries that describe the corresponding data to obtain from the input data FIFO, the output data FIFO, the auxiliary data FIFO, or the padding module. (The padding module provides a means for generating different types of padding and random numbers.) The data is obtained from each source in the order in which the NFIFO entries are loaded. Note that a single entry is able to describe the same (in-snooping) or different (out-snooping) sources for the class 1 and class 2 alignment blocks.

Typically, a command which loads data to the input data FIFO or pulls data through one of the alignment blocks will result in SEC automatically generating the proper NFIFO entries to handle that data. However, that functionality can be overridden to allow the descriptor to directly specify the NFIFO entries. Entries can be placed into the NFIFO via a LOAD Immediate command with a DST value of 7Ah or 70h through 75h.

### 7.7.6 Output FIFO Operation

There are three ways to get data into the output FIFO: a LOAD IMM to the output FIFO, a move command where the destination is the output FIFO, and CHAs pushing their results into the output FIFO. It is up to the descriptor writer to ensure that there are no collisions of data from these sources. If such a collision does occur, an error will be generated.

The output FIFO does not track valid bytes. Therefore, it is up to the descriptor writer to know which bytes in the output FIFO are valid. For example, if you push 3 bytes into the output FIFO followed by 5 more bytes, these 8 bytes are not contiguous. The first three bytes are in one dword and the other 5 bytes are in a second dword. However, the output FIFO always stores 8 bytes per push. When a LOAD IMM to the output FIFO is done, the specified number of bytes are left aligned and any other bytes are written as provided. That is, if the immediate data is to be one byte, 55h, but the 4-byte value provided is 55443322h, then all four bytes are written to the output FIFO along with 4 more bytes of 0. Moves to the output FIFO will have similar results. However, if a CHA is pushing 3 bytes into the output FIFO, those bytes will be left aligned and the other 5 bytes will be 0.

The output FIFO provides data through two access points. The first is for the external DMA and the second is shared by three consumers: the CCB DMA, DECO access via the MATH command, and the NFIFO. The two access points have separate indices into the output FIFO so each can track separately allowing consumption of data at different rates. The following list illustrates how these indices work.

- If the current NFIFO entry is not pulling data from the output FIFO, then whenever the external DMA pops an entry off the output FIFO, the two indices increment.
- If the CCB DMA pops an entry off the output FIFO, both indices will increment.
- If the DECO pops an entry off the output FIFO via the MATH command, both indices will increment.
- If the current NFIFO entry is pulling data from the output FIFO, then the two indices will track separately if the NFIFO entry is not STYPE=01 and AST=1. This is a critical point to understand: since the indices are tracking separately, if one of the consumers, either the NFIFO or the external DMA, falls far enough behind the other, the output FIFO can fill and operations will stall until the lagging consumer catches up. If the NFIFO is consuming data but there are no FIFO STOREs to advance the

external DMA pointer, then the NFIFO can only consume as much data as the output FIFO can hold before a hang will result.

- If the current NFIFO entry has STYPE=01 and AST=1, the indices will both increment when the NFIFO pops entries from the output FIFO. This scenario is useful when all of the data being pushed into the output FIFO is to be consumed via one of the alignment blocks.

There are two ways to alter the behavior of the output FIFO via descriptor control. The first is the means to set the index shared by the CCB DMA, DECO, and the NFIFO to have the same value as the index used for the external DMA. This is done via a LOAD IMM to the DECO control register. The second method is to reset the output FIFO, which clears the data in the FIFO and resets both indices. The reset can also be done via a LOAD IMM to the DECO control register. Another means of resetting the output FIFO is via a LOAD IMM to the Clear Written register.

Another way to alter access to the output FIFO is via the ofifo offset. This value is tracked by DECO as a means of remembering where the last access left off. For example, if a SEQ FIFO STORE of 3 bytes is done, what happens to the other 5 bytes in the output FIFO entry? Both the NFIFO entry and the FIFO STORE commands allow the descriptor to have these remaining bytes retained or discarded. If the OC bit in the NFIFO entry is set when the NFIFO is pulling data from the output FIFO, the remaining bytes are retained. A subsequent access via DECO, the CCB DMA, or the NFIFO will be able to obtain this data. However, the descriptor writer will be responsible for shifting the data as needed to get to the remaining bytes if the access is done via the DECO or the CCB DMA since only the NFIFO will be tracking where it left off.

If the CONT bit in the FIFO STORE command is set, the remaining bytes are also retained when the the external DMA reads the specified number of bytes. In this case, it is DECO which tracks how many remaining bytes there are so that the subsequent FIFO STORE command will start where the prior one left off. The move commands will also use the ofifo offset, so that it can also start with the remaining data. However, the CCB DMA will always pop entries from which it takes data so that it is not possible for the CCB DMA to leave any trailing bytes in the output FIFO. Please see the section for the move commands for important information on how this works.

In order to provide greater control of access to the output FIFO, the value of the ofifo offset can be changed via a LOAD IMM to the DECO control register. This feature can be useful in several scenrios:

- If there are 7, or fewer, bytes of interest in the current output FIFO entry, and this data needs to be stored to memory and used within DECO but snooping is not convenient, the descriptor could do a FIFO STORE with the CONT bit set, and then do a move from the output FIFO to another destination in DECO or the CCB. (If the original ofifo offset was nonzero, the sum of the original ofifo offset and the number of bytes of interest must be less than 8.

- In order to do a single FIFO STORE of data that was sent to the output FIFO via separate methods, e.g. a LOAD IMM to the output FIFO followed by data from a CHA, the data must be contiguous in the output FIFO. For example, a LOAD IMM to the output FIFO of 3 bytes followed by data from the CHA would have a 5-byte gap between the loaded data and the CHA data. This could be solved by shifting the load data 5 bytes to the right and then doing a LOAD IMM of 8 bytes with those 3 bytes of interest right aligned. But now there are 5 "garbage" bytes at the start of the output FIFO data. These can be skipped over by setting the ofifo offset to 5. Now, a single FIFO STORE can be done with a length of the number of bytes pushed by the CHA plus 3.
- The ofifo offset can also be set to a smaller value than it currently contains. This can allow the same data to be stored twice, although the limit is back to the start of the current entry.

### 7.7.7 Output Checksum logic

Normally, all data written out through a SEQ STORE or SEQ FIFO STORE command has a checksum computed upon it.

The checksum computed is a sixteen-bit ones complement modular sum checksum, compatible with UDP and TCP. RFC 793 uses this description: "The checksum field is the 16 bit one's complement of the one's complement sum of all 16-bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16-bit word for checksum purposes."

By default, all bytes written through a SEQ STORE or SEQ FIFO STORE command are included in the computation. The SEQ FIFO STORE command has special output data types (not available to FIFO STORE) to enable (output data type 31h) or disable (output data type 30h) the checksum logic. The first time the enable is invoked, the contents of the DECO checksum register are cleared so that the checksum appears to have started with the enable. An enabling or disabling SEQ FIFO STORE may be used with a zero length to simply "turn on" or "turn off" the checksum computation without actually writing any contents of the output FIFO to memory.

If the data is stored via multiple SEQ STORE and/or SEQ FIFO STORE commands, each store command will result in a data segment of a specified number of bytes being added to the checksum. If, for one such segment, the last byte ends at an odd boundary, the next data to be checked will start on the odd boundary. In other words, the checksum is computed as if all of the segments to be checked are concatenated. This is true even if there are checksum-disabled bytes between two checksum-enabled segments. If the final length of the data is odd, then the checksum is computed as if a zero byte was appended.

The contents of the checksum computation register can be written at any time by performing a STORE or SEQ STORE of the DCHKSM register.

The checksum computation logic is present in part to enable UDP-encapsulated-ESP as part of IPsec ESP Tunnel encapsulation. See section Outer IP Header handling with UDP-encapsulated-ESP for details on ESP Tunnel support for UDP-encapsulated-ESP. When ESP Tunnel performs UDP-encapsulated-ESP, the checksum logic controls are overridden in order to perform the proper UDP checksum computation. The checksum result can be written out using any of the methods above, but that value will duplicate the UDP checksum included in the UDP header. The IPsec ESP tunnel protocol, if NAT and NUC are both set, enable and disable the checksum logic as needed. Otherwise, the IPsec ESP tunnel protocol, and all other protocols including IPsec ESP Transport (and legacy tunnel) neither enable nor disable the checksum logic.

**NOTE**

In some instances, a protocol may write out more data past the end of the output frame. In the case of IPsec ESP Transport Decapsulation and IPSEC ESP Tunnel Decapsulation, this is because the protocol doesn't determine the end of the decapsulated payload until decrypting the Pad Length byte found in the ESP trailer. For all instances when this occurs, the computation of the checksum will include all the bytes actually written out to memory, even if the output frame length is subsequently adjusted to hide them.

### 7.7.8 Cryptographic class

SEC divides cryptographic algorithms into two different classes for the purpose of selecting CHAs. Some key and data movement commands must have a CLASS value associated with them so they are delivered to the proper CHA.

The following table shows the class for each cryptographic algorithm.

**Table 7-5. Classes of the cryptographic algorithms**

| Class | Algorithm |
|---|---|
| Class 1 | AES (all modes)[1], DES, 3DES, Kasumi, SNOW f8, ZUCE, Public Key, RNG |
| Class 2 | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256, SNOW f9, CRC, ZUCA AES authentication modes, MD5 |

1. Note that AES optimization modes (such as, modes that combine encryption and authentication) require that the authentication key be present in the Class 2 Key Register, and the encryption/decryption key be present in the Class 1 Key Register.

**NOTE**

A descriptor that requests both a Class 1 CHA and a Class 2 CHA must request the Class 2 CHA first. Otherwise, in versions of SEC that implement more than one DECO a deadlock situation could occur as follows:

- Descriptor *x* executing in DECO *x* acquires CHA 1 and then requests CHA 2.
- Descriptor *y* in DECO *y* acquires CHA 2 and then requests CHA 1.
- Descriptor x and descriptor y wait until both CHAs are available, but neither will release the CHA that it currently has.

If descriptors always acquire CHAs in the same order, this deadlock situation is avoided. The required order is Class 2 first, then Class 1. An error is generated if a Class 2 CHA is selected after a Class 1 CHA.

Note that software written for versions of SEC that implement only one DECO must still follow this practice to ensure that the software is portable to versions of SEC that implement two or more DECOs.

When specifying classes in commands, a two-bit field is used to specify class as follows:

**Table 7-6. Class field**

| Class Value | Meaning for LOAD and STORE Commands | Meaning for Other Commands |
|---|---|---|
| 00 | CCB class independent | None, sequence data skipped for SEQ FIFO LOAD command. |
| 01 | CCB Class 1 | Class 1 |
| 10 | CCB Class 2 | Class 2 |
| 11 | DECO | Both Class 1 and Class 2 |

## 7.7.9 Address pointers

Many of the descriptor commands and several data structures used by SEC include Pointer fields. If PS = 0 in the Master Configuration Register, all address pointers used by SEC will be 32-bits in length. If PS = 1 in the Master Configuration Register, all pointers used by SEC will be 40 bits in length, laid out as shown in Table 7-7 when the endianness is in the default configuration. The default endianness configuration can be overridden for each individual Job Ring via the PEO bit in the Job Ring Configuration Register (JRCFGR). The default endianness configuration can be overridden for RTIC via the REPO bit in the RTIC Endian Register (REND). When the endianness is overridden, the two portions of the 40-bit address appear in the opposite order from that shown below.

When operating in little-endian mode, it is important to remember that the least significant word of two-word addresses appears prior to the most significant word in all locations. If the descriptor needs to manipulate such addresses, the MOVEW command can be used to move the address to a Math Register such that it arrives as an 8-byte integer ready for manipulation. The MOVEW command can then be used to move the address back Note also that the same endianess controls which govern the order of the address words also govern whether 8-byte immediate values for the MATH command are word swapped when the LENGTH field is specified as 9.

**Table 7-7. Format of 40-bit pointers**

| | | |
|---|---|---|
| Dword at address X | 24 bits ignored | most-significant 8 bits of 40-bit address |
| Dword at address X +4 | least-significant 32 bits of 40-bit address | |

## 7.8 HEADER command

Every descriptor begins with a HEADER command, which provides basic information about the descriptor itself, such as length, ability of DECOs to share the descriptor, and whether errors are propagated when the descriptor is shared.

Job descriptor and shared descriptor HEADER commands share a base format, but some fields are specific to descriptor type. The formats of the Job Descriptor Header Command and Shared Descriptor Header Command are shown in the diagrams below, and the fields of both are described in detail below each format diagram.

**Table 7-8. Job descriptor header command format**

| 31-27 | 26 | 25 | 24 | 23 | 22 | 21-16 |
|---|---|---|---|---|---|---|
| CTYPE = 10110 | EXT | RSd | DNR | ONE | Reserved | START INDEX / SHR DESCR LENGTH |
| **15** \| **14-13** \| **12** \| **11** | **10-8** | | **7** | **6-0** | | |

*Table continues on the next page...*

### Table 7-8. Job descriptor header command format (continued)

| ZRO | TDES | SHR | REO | SHARE | Reser ved | DESCLEN |
|---|---|---|---|---|---|---|
| *Optional additional words of HEADER command:* | | | | | | |
| If SHR = 1, a one or two word pointer to a shared descriptor is located immediately after the HEADER. | | | | | | |
| If EXT = 1, a job descriptor header extension is located immediately after the HEADER or, if SHR = 1, immediately after the shared descriptor pointer. (see Job descriptor header extension format, below) | | | | | | |

### Table 7-9. Job descriptor header field descriptions

| Job descriptor header fields | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>10110 Job descriptor header |
| 26<br>EXT | Extended Job Descriptor HEADER<br>If EXT=0 : There is no extended HEADER word.<br>If EXT=1 : The HEADER command contains an extended HEADER word, as illustrated in the format diagram below. Note that if there is no shared descriptor a HEADER error (13h) is generated if EXT=1 and START INDEX=1. |
| 25<br>RSI | Requires SEQ ICID to be the same. This bit is used to ensure that ICID-based access control cannot be bypassed by sharing shared descriptors.If RSd=0 the ICIDs of two job descriptors *do not* have to match in order for them to share the same shared descriptor. If RSd=0 the ICIDs of two job descriptors **do** have to match in order for them to share the same shared descriptor. |
| 24<br>DNR | Do Not Run<br>If DNR=0 : Normal execution of the job descriptor.<br>If DNR=1 : Do Not Run the job descriptor. There was a problem upstream so this descriptor should not be executed. This allows the job to be passed through the hardware and software pipeline to a point where the problem might be corrected by software and the job resubmitted.<br>**NOTE:** If this bit is found in a job descriptor header, SEC still fetches any associated shared descriptor. If the shared descriptor header's PD bit is set and the DNR bit is not set, SEC updates the shared descriptor header's DNR bit. As a result, future job descriptors that use this shared descriptor do not run. Once software clears the DNR bit in the shared descriptor, any new job descriptors that use this shared descriptor run normally. In addition, when the DNR bit is set and the input frame buffers were to be released, DECO will release them. |
| 23<br>ONE | One<br>The ONE bit is always 1. This bit is used in combination with the ZRO bit to verify that the endianness of the header is correct. This is necessary because SEC is used in chips with both big-endian and little-endian processors. |
| 22 | Reserved |
| 21-16<br>START INDEX/ SHR DESCR LENGTH | Start Index or Shared Descriptor Length<br>If SHR = 0, this is the START INDEX field<br>Start Index specifies the position of the word in the descriptor buffer where execution of the job descriptor should continue following execution of the job descriptor HEADER. That is, DECO should jump to the specified word to continue processing. Note that if there is a HEADER extension word (EXT=1) the START INDEX must not be 1, else a 13h (header) error will result.<br>If SHR = 1, this is SHR DESCR LENGTH field |

*Table continues on the next page...*

## Table 7-9. Job descriptor header field descriptions (continued)

| Job descriptor header fields | Description |
|---|---|
| | The Shared Descriptor Length specifies the length of the Shared Descriptor (in 32-bit words). |
| 15<br>ZRO | Zero<br>The ZRO bit is always 0. This bit is used in combination with the ONE bit to verify that the endianness of the header is correct. This is necessary as SEC is used in NXP product lines with both big-endian and little-endian processors. |
| 14-13<br>TDES | Trusted Descriptor<br>If TDES=00b : This is a normal job descriptor, that is, not a trusted descriptor. However, if the AMTD bit is set in the JRaICID register this descriptor can be run as a trusted descriptor by setting the FTD bit in the extended header word. In that case, no SIGNATURE command is required and no signature will be generated or verified. Note that if FTD=1 in the extended header word then TDES must be 00b. An error will be generated if this is not the case.<br>If TDES=01b : This is a TrustZone SecureWorld trusted descriptor - a special trusted descriptor created by TrustZone SecureWorld.<br>If TDES=10b : This is a TrustZone non-SecureWorld Trusted descriptor, that is, a Trusted Descriptor created by TrustZone non-SecureWorld.<br>If TDES=11b : This is a candidate trusted descriptor, that is, a descriptor that will be made into a trusted descriptor by appending a signature to it. Note that an error will be generated if AMTD=0 in the job ring's ICID register. The DESCLEN field must account for the eight 32-bit words of signature which will be added.<br>See the discussion Trusted descriptors for an explanation of trusted descriptors. |
| 12<br>SHR | Shared Descriptor (SHR) flag<br>If SHR=0 : This job descriptor does not have a shared descriptor and so does not include a shared descriptor pointer.<br>If SHR=1 : This descriptor has a shared descriptor that is pointed to by the next word or words.<br>SHR controls how START INDEX / SHR DESCR LENGTH is used. |
| 11<br>REO | Reverse Execution Order (REO). Note that this bit is ignored if SHR = 0 (that is, no shared descriptor).<br>If REO=0 : The shared descriptor is executed prior to the remainder of the job descriptor.<br>If REO=1 : The job descriptor will be executed prior to the shared descriptor. Setting REO=1 in a trusted descriptor results in an error. |
| 10-8<br>SHARE | Share State (SHARE)<br>This defines if, and when, the shared descriptor of this descriptor can be shared with another job descriptor. (See Table 7-1.) Also see Specifying different types of shared descriptor sharing for further information. |
| 7 | Reserved |
| 6-0<br>DESCLEN | Descriptor Length<br>This field represents the total length in 4-byte words of the descriptor. A descriptor length of 0 is undefined. The header word is included in the length. Note that the size of the descriptor buffer is 64 words, so that is the maximum size of a single job descriptor with no shared descriptor. |

## Table 7-10. Job descriptor header extension format

<table>
<tr><td colspan="10">31-16</td></tr>
<tr><td colspan="10">Reserved</td></tr>
<tr><td>15-9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

*Table continues on the next page...*

## Table 7-10. Job descriptor header extension format (continued)

| | | | | |
|---|---|---|---|---|
| Reserved | FTD | DSELECTVALID | Reserved | DECO_SELECT |

## Table 7-11. Job descriptor header extension field descriptions

| Field | Description |
|---|---|
| 31-9 | Reserved. |
| 8<br>FTD | Fake Trusted Descriptor. Treat the current descriptor as a trusted descriptor but do not check the signature. If the descriptor is run in a job ring owned by TrustZone SecureWorld, the descriptor will be treated as a TrustZone SecureWorld Trusted Descriptor, otherwise the descriptor will be treated as a TrustZone non-SecureWorld Trusted Descriptor. Note that an error will be generated if FTD=1 and the source job ring's JRaICID register AMTD=0 (that is, the extended header says to run the descriptor as a trusted descriptor, but the job ring is not allowed to make trusted descriptors). In order to use FTD, the TDES field in the first word of the header command must be 00b. An error will be generated if this is not the case.<br>If FTD=1, no SIGNATURE commands are required. If any SIGNATURE skip commands are present they will be treated as no-ops. If a final SIGNATURE command is present, it will be treated as the end of the descriptor. |
| 7<br>DSELECTVALID | DECO_SELECT field is valid.<br>If DSELECTVALID=0 : Any DECO can run the job. The DECO_SELECT field is ignored.<br>If DSELECTVALID=1 : The job must be run in the DECO specified in the DECO_SELECT field. If the number specifies an unimplemented DECO, DECO error 026h will be generated. |
| 6-4 | Reserved |
| 3-0<br>DECO_SELECT | DECO Select<br>If DSELECTVALID = 1, the job is run in the DECO specified in the DECO_SELECT field. If the number specifies an unimplemented DECO, an error will be generated, however ***only the bits that are actually implemented in the DECO_SELECT field are considered***. Note that the size of the DECO_SELECT field depends on the number of DECOs that are implemented in SEC:<br>• 15-8 DECOs, DECO Select field is 4 bits<br>• 7-4 DECOs, DECO Select field is 3 bits<br>• 3-2 DECOs, DECO Select field is 2 bits<br>• 2-1 DECOs, DECO Select field is 1 bit<br>Note that for programming consistency, a one-DECO version of SEC has a one-bit DECO_SELECT field. The unimplemented bits are reserved. They should be written as 0s for compatibility across different versions of SEC.<br>**NOTE:** DECO-specific jobs have the possibility to create a deadlock in SEC when they are used as part of a flow. Therefore, it is strongly recommended that DECO-specific jobs either not be part of a flow or all the jobs in the flow be assigned to the same DECO. |

## Table 7-12. Shared descriptor header format

| 31-27 | | | | | 26 | 25 | 24 | 23 | 22 | 21-16 |
|---|---|---|---|---|---|---|---|---|---|---|
| CTYPE = 10111 | | | | | Reserved | RIF | DNR | ONE | Reserved | START INDEX |
| **15** | **14** | **13** | **12** | **11** | **10** | **9-8** | | **7-6** | | **5-0** |
| ZRO | Reserved | CIF | SC | PD | Reserved | SHARE | | Reserved | | DESCLEN |

## Table 7-13. Shared descriptor header field descriptions

| Shared descriptor header fields | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>10111 Shared descriptor |
| 26 | Reserved |
| 25<br>RIF | Read Input Frame<br>As soon as possible, DECO should read the entire input frame as defined in a SEQ IN PTR command in the job descriptor. The length of the input frame is placed in the VSIL Register. The data is read into the input data FIFO. This is the equivalent of a SEQ FIFO LOAD of the entire input frame without an NFIFO entry or any data size registers being written.<br>This bit is intended to allow DECO to issue reads from the SEQ IN PTR as soon as possible, thereby reducing processing latency. However, there are contraindications to its use:<br>• If the descriptor contains any LOAD or KEY command that is not immediate.<br>• If the descriptor contains a KEY command that loads an encrypted key. The Derived Key PROTOCOL OPERATION command is not included in this restriction.<br>• If the descriptor contains a SEQ IN PTR with RTO (Restore Input Sequence) set<br>• If the descriptor contains a PROTOCOL OPERATION comamnd specifying SSL / TLS Decapsulation<br>• If the descriptor contains a PROTOCOL OPERATION command specifying IPsec ESP Tunnel mode encapsulation *and* PDB Options field OIHI=10 (specifying inclusion of an Outer IP Header referenced by the PDB).<br>• If the descriptor contains a PROTOCOL OPERATION command specifying LTE Control Plane encapsulation or LTE Data Plane encapsulation for RN with either SNOW or ZUC for the confidentiality algorithm *and* AES for the integrity algorithm.<br>• If the descriptor contains a PROTOCOL Operation command specifying either Blob encapsulation or Blob decapsulation.<br>There are restrictions with the use of RIF with Public Key Cryptography operations and PRFs: RIF may be used, but only if all the input FIFO is drained by other descriptor commands before the PROTOCOL COMMAND is encountered. |
| 24<br>DNR | Do Not Run<br>0 Normal execution<br>1 Do Not Run. There was a problem upstream so this descriptor should not be executed.<br>**NOTE:** If this bit is found in a job descriptor header, SEC still fetches any associated shared descriptor. If the shared descriptor header's PD bit is set and the DNR bit is not set, SEC updates the shared descriptor header's DNR bit. As a result, future job descriptors that use this shared descriptor do not run. Once software clears the DNR bit in the shared descriptor, any new job descriptors that use this shared descriptor run normally. In addition, when the DNR bit is set and the input frame buffers were to be released, DECO will release them. |
| 23<br>ONE | One<br>The ONE bit is always 1. This bit is used in combination with the ZRO bit to verify that the endianness of the header is correct. This is necessary because SEC is used in chips with both big-endian and little-endian processors. |
| 22 | Reserved |
| 21-16<br>START INDEX | This is the START INDEX field, which specifies the index of the word in the descriptor buffer where execution of the shared descriptor should start. This allows protocol or other information to be jumped over. |
| 15<br>ZRO | Zero |

*Table continues on the next page...*

Table 7-13. Shared descriptor header field descriptions (continued)

| Shared descriptor header fields | Description |
|---|---|
| | The ZRO bit is always 0. This bit is used in combination with the ONE bit to verify that the endianness of the header is correct. This is necessary as SEC is used in NXP product lines with both big-endian and little-endian processors. |
| 14 | Reserved |
| 13<br>CIF | Clear Input FIFO (CIF)<br>If set, the input FIFO and the NFIFO entries are reset between self-shared descriptors. That is, these are reset if the next job to be run within the same DECO has the same shared descriptor as the previous job run in that same DECO. (The Input FIFO and NFIFO are always reset between descriptors that don't share the same shared descriptor.) |
| 12<br>SC | Save Context (SC)<br>After this descriptor completes, if Serial-Sharing is selected, and if sharing of the shared descriptor occurs within the same DECO (self-sharing):<br>0 The context registers are cleared.<br>1 The context registers are maintained and used by the subsequent descriptor.<br>Save Context is intended to allow multiple subsequent shared descriptors to maintain cryptographic context when a cryptographic operation is split between multiple jobs. |
| 11<br>PD | Propagate DNR (PD)<br>If the job descriptor's DNR bit is set and this bit is set, set the DNR bit of the shared descriptor header if it is not already set. |
| 10 | Reserved. |
| 9-8<br>SHARE | Share State (SHARE)<br>This defines if, and when, the shared descriptor of this descriptor can be shared with another job descriptor. (See Table 7-1.) Also see Specifying different types of shared descriptor sharing" for further information. |
| 7-6 | Reserved |
| 5-0<br>DESCLEN | Descriptor Length<br>This field represents the total length in 4-byte words of the shared descriptor. A shared descriptor length of 0 is undefined. The header word is included in the length. Note that the size of the shared descriptor buffer is 64 words, so the maximum size of a shared descriptor is 62 words (assuming that the pointer size is 32 bits and the job descriptor consists of only the Job HEADER command and the pointer to the shared descriptor). |

If the SHR bit in a job descriptor header command is set, a pointer to the shared descriptor immediately follows the header.

## 7.9 KEY commands

NOTE

In the following discussion, the term 'KEY command' refers to both the SEQ and non-SEQ forms of the command.

KEY commands are used to load keys into one of SEC's key registers: Class 1 or Class 2 Key Register or PKHA E-Memory. The SEQ KEY command is identical to the KEY command except that no address is specified and the VLF bit replaces the SGT bit and the AIDF bit replaces the IMM bit.

If the key to be loaded into a key register is encrypted, SEC can be told to automatically decrypt it as it is loaded into the key register. The MOVE command, LOAD command and KEY command can all be used to load a Red Key into a key register, but only the KEY command can be used to load a Black Key. Note that Black Keys can be loaded only into PKHA E-Memory or key registers because only KEY commands decrypt Black Keys, and these registers are the only possible destinations for KEY commands.

If the KEY command is loading a Black Key, the Class 2 key must be loaded prior to the Class 1 key as the loading of encrypted keys has side effects on the Class 1 Key Register.

If ENC is set (that is, a Black Key is being loaded), the KEY command has significant side effects, including clearing the following:

- Input Data FIFO
- Output Data FIFO
- Class 1 Key Register
- Class 1 Data Size Registers
- Class 1 Mode Register
- Class 1 Context (if EKT is also set)

As a result, the only commands that should precede loading a Black Key are:

- JUMP
- SEQ IN PTR
- SEQ OUT PTR
- LOADs to registers not mentioned above.
- MOVEs to or from registers not mentioned above.

**NOTE**

The KEY command is blocking under the following circumstances:

1. Decrypting a black key.
2. Loading a red key that is NOT immediate.
3. CHAs are not done.
4. The data must pass through the input FIFO and there are info FIFO entries in the way.
5. The data must be read into the data FIFO and there is other data in the input data FIFO that is in the way. (This does not apply to SEQ KEY AIDF.)

6. The CCB DMA is required but is busy.
7. The hardware which schedules external reads is required but is busy.

**Table 7-14. KEY command format**

<table>
<tr><th colspan="3">31–27</th><th colspan="2">26–25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19–18</th><th>17–16</th></tr>
<tr><td colspan="3">CTYPE = 00000 or 00001</td><td colspan="2">CLASS</td><td>SGF or VLF</td><td>IMM or AIDF</td><td>ENC</td><td>NWB</td><td>EKT</td><td>Reserved</td><td>KDEST</td></tr>
<tr><th>15</th><th>14</th><th colspan="2">13–10</th><th colspan="8">9–0</th></tr>
<tr><td>TK</td><td>PTS</td><td colspan="2">Reserved</td><td colspan="8">LENGTH</td></tr>
<tr><td colspan="12"><i>Additional words of KEY command:</i></td></tr>
<tr><td colspan="12">Pointer (one or two words) or Value (if immediate, one or more words)</td></tr>
</table>

**Table 7-15. KEY command field descriptions**

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command Type<br>If CTYPE=00000b : KEY command<br>If CTYPE=00001b : SEQ KEY command |
| 26-25<br>CLASS | Class. This defines whether this key is for a Class 1 or Class 2 algorithm.<br>If CLASS=00b : Reserved<br>If CLASS=01b : Class 1 Key<br>If CLASS=10b : Class 2 Key<br>If CLASS=11b : Reserved<br>**NOTE:** Class must be set to Class 1 if the key DEST field is set to 01b. Class must be set to Class 2 if the key DEST field is set to 11b. |
| 24<br>SGF or VLF | Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF)<br>If CTYPE = 00000b (KEY), this bit is the Scatter/Gather table Flag (SGF).<br>If SGF=0: Pointer points to actual data.<br>If SGF=1: Pointer points to a scatter/gather table.<br>**NOTE:** It is an error for this bit to be set if the IMM bit is set.<br>If CTYPE = 00001b (SEQ KEY), this bit is the Variable Length Flag (VLF).<br>If VLF=0: The number of bytes of data to be loaded into the key register is specified by the LENGTH field.<br>If VLF=1: The number of bytes of data to be loaded into the key register is specified by the value in the VSIL register. |
| 23<br>IMM or AIDF | Immediate Flag (IMM) or Already in Input Data FIFO (AIDF)<br>If CTYPE = 00000 (KEY), this bit is the IMM flag.<br>If IMM=0 : The key value is found at the location pointed to by the pointer in the next word (32-bit pointers) or next dword (> 32-bit pointers).<br>If IMM=1 : The key value follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word. |

*Table continues on the next page...*

## Table 7-15. KEY command field descriptions (continued)

| Field | Description |
|---|---|
| | **NOTE:** PKHA E-Register values can be very large and may not fit within the descriptor buffer.<br>If the SGF bit is set, It is an error for the IMM bit to be set.<br>If CTYPE = 00001b (SEQ KEY), this bit is the AIDF flag.<br>If AIDF=0 : Read the Input Data Sequence data and load it into the specified destination.<br>If AIDF=1 : Do not read the Input Data Sequence data since it is already in the Input Data FIFO, but load the data in the Input Data FIFO into the specified destination. It is an error for ENC and AIDF to both be 1. |
| 22<br>ENC | Key is encrypted<br>If ENC = 0 : The key is assumed to be in plaintext and is loaded into the destination register without decryption.<br>If ENC = 1 : SEC automatically decrypts the key (using the JDKEK, or if this is a trusted descriptor, using the TDKEK if TK = 1) before putting it in the key register. Decrypting a key requires using the AESA, the input and output data FIFOs, Class 1 Mode register, Key Size, context, and key registers. Therefore, Class 2 Black Keys (if any) must be loaded prior to loading the Class 1 register, and Class 1 Black Keys must be loaded prior to loading any of the resources noted above. |
| 21<br>NWB | No Write Back<br>If NWB=0 : Note that it is not usually possible to write a key back out to memory as plaintext, but if NWB=0 the key can be written out as a Black Key by using the FIFO STORE command.<br>If NWB=1 : Prevents the key that is loaded into the key register from being written back out to memory.<br>NWB applies to all key locations: Class 1 Key Register, Class 2 Key Register, and PKHA E Memory. Setting this bit sets the key register's NWB flag. The No Write Back setting lasts until the end of the descriptor (or sequence of shared descriptors) or until the corresponding key register or CHA is cleared/reset. |
| 20<br>EKT | Encrypted Key Type<br>The EKT bit determines which decryption mode is used when a Black Key (ENC = 1) is loaded.<br>If EKT=0 : The Black Key is decrypted using AES-ECB.<br>If EKT=1 : The Black Key is decrypted using AES-CCM.<br>A Black Key encrypted with AES-ECB must be decrypted with AES_ECB, and a Black Key encrypted with AES-CCM must be decrypted with AES_CCM. If the wrong mode is selected it is possible that no error will be issued but the value loaded into the key register will be incorrect.<br>Note that an error status is generated if EKT=1 and ENC=0. |
| 19-18 | Reserved |
| 17-16<br>KDEST | Key Destination<br>If KDEST=00b : The key is loaded into the Key Register (either Class 1 or Class 2 as specified in the CLASS field)<br>If KDEST=01b : The key is loaded into the PKHA E-Memory. This key destination requires CLASS = 01b (Class 1 key).<br>If KDEST=11b : The key is regarded as an MDHA Split Key, and is loaded into the Class 2 Key Register. An MDHA split key is the concatenation of the IPAD material followed by the OPAD material. Split keys offer higher performance for HMACs. Note that MDHA IPAD/OPAD values are considered keys and are decrypted to the Class 2 Key Register. This key destination requires CLASS = 10b (Class 2 key).<br>All other values are reserved. |
| 15<br>TK | Trusted Key<br>This bit is used only by trusted descriptors. If not a trusted descriptor, setting TK = 1 and ENC = 1 is an error. If the ENC bit is not set, this bit is ignored.<br>If TK=0 : Use the Job Descriptor Key Encryption Key (JDKEK) to decrypt the key that is to be loaded into a key register. |

*Table continues on the next page...*

**Table 7-15. KEY command field descriptions (continued)**

| Field | Description |
|---|---|
| | If TK=1 : A trusted descriptor wants to use the Trusted Descriptor Key Encryption Key (TDKEK) to decrypt the key that is to be loaded into a key register. |
| 14<br>PTS | Plaintext Store<br>If PTS=0: The key loaded cannot later be stored in plaintext form.<br>If PTS=1: The key loaded can later be stored in plaintext form using a FIFO STORE or SEQ FIFO STORE command. Note the following restrictions:<br>The Class 2 Key register can be stored in plaintext form if a split key was loaded into it with a KEY command with PTS=1 or if a key is loaded into the Class 2 Key register with a KEY command with PTS=1 and the MDHA is run in INIT mode to create a split key.<br>An error is generated<br>• if PTS=1 & ENC=1.<br>• if PTS=1 & NWB=1.<br>• if PTS=1 & KDEST=01b (PKHA E-Memory).<br>• if a non-split key is stored from the Class 2 Key register after the Class 2 Key register was loaded with a KEY command with PTS=1.<br>• if the Class 1 Key register is stored after the Class 1 Key register was loaded using a KEY command with PTS=1. |
| 13-10 | Reserved |
| 9-0<br>LENGTH | Key Length<br>This field defines the length of the key in bytes. If the key is encrypted, this is the decrypted length of the key material only. The built-in key decryption operation produces output whose length is as specified in the LENGTH field. ECB encrypted keys are padded to 16-byte boundaries, so the KEY command reads enough input to read the entire encrypted key. CCM-encrypted keys have a 6-byte nonce, a 6-byte MAC, and padding of up to 7 bytes. The length is checked to ensure it is not too large for the specified destination. |
| | *Additional words of KEY command:* |
| POINTER | If IMM = 0, this field is a pointer to the key to be loaded.<br>If IMM = 1, this field is not present.<br>**NOTE:** This field is not present for SEQ KEY Commands. |

## 7.10 LOAD commands

**NOTE**

In the following discussion, the term 'LOAD command' refers to both the SEQ and non-SEQ forms of the command.

LOAD commands are used to load values into registers, either directly from the descriptor (a LOAD IMMEDIATE command contains constant data within the command) or from a memory location addressed by a pointer within the command. The SEQ LOAD command is identical to the LOAD command except that no address is specified, the VLF bit replaces the SGF bit, and the immediate bit cannot be set. See SEQ vs non-SEQ commands. (Note that while SEQ KEY and SEQ FIFO LOAD have an AIDF bit, SEQ LOAD does not.)

When reading from an external address, the LOAD command, whether SEQ or non-SEQ, uses hardware within DECO to schedule DMA transactions. This command will block until that hardware is available. For LOAD IMM, if the DMA hardware is required but is in use, the command will block until the DMA hardware becomes available. (The command may block for other reasons as well, as documented in a following table.) Once the command is handed off to the responsible hardware, descriptor execution will continue with the next command. Therefore, the requested data may not be present for some time. It is up to the descriptor writer to ensure that the data arrives prior to attempting to use it. Paying attention to the blocking nature just discussed is critical in order to avoid hanging descriptors.

The definitions of the OFFSET and LENGTH fields in the LOAD command can depend on the CLASS and destination (DST) fields. The first table below shows the LOAD command fields, the second table defines the fields, and the third table defines the legal destinations and how each destination affects the other fields.

Table 7-16. LOAD command format

<table>
<tr><th>31–27</th><th>26-25</th><th>24</th><th>23</th><th>22–16</th></tr>
<tr><td>CTYPE = 00010 or 00011</td><td>CLASS</td><td>SGF or VLF</td><td>IMM</td><td>DST</td></tr>
<tr><th colspan="3">15–8</th><th colspan="2">7–0</th></tr>
<tr><td colspan="3">OFFSET</td><td colspan="2">LENGTH</td></tr>
<tr><th colspan="5"><i>Additional words of LOAD command:</i></th></tr>
<tr><td colspan="5">Pointer (one or two words, see Address pointers) or Value (if immediate, one or more words)</td></tr>
</table>

Table 7-17. LOAD command field descriptions

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command Type<br>If CTYPE=00010b : LOAD command<br>If CTYPE=00011b : SEQ LOAD command |
| 26-25<br>CLASS | Class. The algorithm class of the data to be loaded.<br>If CLASS=00b : Load class-independent objects in CCB.<br>If CLASS=01b : Load Class 1 objects in CCB.<br>If CLASS=10b : Load Class 2 objects in CCB.<br>If CLASS=11b : Load objects in DECO. |
| 24<br>SGF or VLF | Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF) flag. Meaning depends on CTYPE.<br>If CTYPE = 00010 (LOAD), this bit is the Scatter/Gather table Flag (SGF).<br>If SGF=0 : The pointer points to actual data.<br>If SGF=1 : The pointer points to a scatter/gather table.<br>**NOTE:** If the IMM bit is set, it is an error for this bit to be set. |

*Table continues on the next page...*

### Table 7-17. LOAD command field descriptions (continued)

| Field | Description |
|---|---|
| | If CTYPE = 00011 (SEQ LOAD), this bit is the Variable Length Flag (VLF).<br>If VLF=0 : The LENGTH field indicates the length of the data.<br>If VLF=1 : The length of the data is variable. SEC uses the length in the Variable Sequence In Length register rather than the value in the LENGTH field. However, an error will be generated if the values in the VSIL register and OFFSET field are not a valid combination as indicated in table Table 7-18. |
| 23<br>IMM | Immediate Flag<br>If CTYPE = 00010 (LOAD)<br>If IMM=0, the data to be loaded is found at the location pointed to by the address pointer.<br>If IMM=1, the data to be loaded follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word.<br>**NOTE:** If the SGF bit is set, it is an error for this bit to be set.<br>If CTYPE = 00011 (SEQ LOAD)<br>IMM must be set to 0. Setting IMM to 1 generates an automatic error. |
| 22-16<br>DST | The DST value defines the destination register, such as CONTEXT, ICV, or IV. See Table 7-18 for a list of supported destinations. |
| 15-8<br>OFFSET | OFFSET defines the start point for writing within the destination. The destination DST determines whether the length is specified in bytes or words. See Table 7-18 for details. |
| 7-0<br>LENGTH | Length of the data. The value in the DST field determines whether the length is specified in bytes or words. See Table 7-18 for details. |
| *Additional words of LOAD command:* | |
| 31-0<br>POINTER/ VALUE | Address pointer if IMM = 0 or the immediate value if IMM = 1. Note that the immediate value occupies as many words as required to fit the number of bytes specified in the LENGTH field. Data is left aligned.<br>**NOTE:** This field is present only for LOAD Commands (that is, not for SEQ LOAD Commands). |

SEC can accomplish the data transfer associated with a LOAD immediate command in two different ways:

- Using a direct (non-DMA) path to the register, referred to as a direct immediate load
- Using SEC's internal-transfer DMA

SEC automatically selects the appropriate transfer mechanism as follows:

- SEC selects the direct immediate load data path (the first bullet above) if the restrictions are met because this is the fastest of the transfer mechanisms (see following paragraph).
- If the data length or offset restrictions are not met, SEC automatically selects the internal-transfer DMA data path (the second bullet above).

The direct immediate load is the most efficient of the transfer mechanisms, but it has the following restrictions:

- It can transfer only 4 or 8 bytes, unless the destination is the input Data FIFO, the Auxiliary Data FIFO, or the output Data FIFO, in which case the length must be no more than 8 bytes.
- The sum of the data length and the offset cannot be larger than 8, meaning the legal combinations of length and offset are either
  - 4 bytes with an offset of 0 or 4
  - 8 bytes with an offset of 0
  - 4 bytes with any multiple of a 4-byte offset if the destination is a context register
  - 8 bytes with any multiple of an 8-byte offset if the destination is a context register

As shown in Table 7-18, some registers can be loaded only with a LOAD IMM command. These registers always use the direct immediate load data path. Other registers can be loaded using either the LOAD or LOAD IMM form of the command.

As shown in Table 7-18, some LOAD destinations are control data registers and other destinations are message data registers. Data loaded into or stored from control data registers is regarded as word-oriented data, whereas data loaded into or stored from message data registers is regarded as byte strings.

To facilitate operation in chips with different endianness configurations, the following data-swapping operations can be configured:

- byte-swapping
- half-word swapping
- word-swapping
- double-word swapping

and these swapping operations for control data registers can be configured independently from the swapping for message data registers.

The same swapping operations can be configured independently for:

- each job ring (see the Job Ring Configuration Register (JRCFGR))
- the Queue Manager Interface (see the Queue Interface Control Register (QICTL))

**NOTE**

For those destinations that allow immediate loads with a nonzero offset, the combination of offset=0 length=4 is equivalent to the combination of offset=4 length=4. This has been done to maintain backward compatibility. Both of these combinations will load the right-most word of the destination. Therefore, in order to load the left-most word of the destination, the combination must be offset=0 length=8. This is ONLY the case when IMM=1 and, therefore, does not affect the SEQ LOAD command. When IMM=0, offset=0 length=4 will load the left-most word of the destination while offset=4 length=4 will load the right-most word. This behavior does not affect any other commands.

### Table 7-18. LOAD command DST, LENGTH, and OFFSET field values

| DST value (hex) | Class (binary) | Control data/ message data | Legal values in LENGTH/ OFFSET fields | Must use IMM? | Tag | Internal register | Comment |
|---|---|---|---|---|---|---|---|
| 01 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | C1KSR | Class 1 Key Size Register | The key size registers are normally written with the KEY command. Once KEY SIZE is written, the user cannot modify the key or key size until the key is cleared. |
| | 10 | | | | C2KSR | Class 2 Key Size Register | |
| 02 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | C1DSR | Class 1 Data Size Register | Writes to the data size registers block if there are any outstanding context loads since a write to a data size register indicates that the corresponding context is in place and ready. |
| | 10 | | | | C2DSR | Class 2 Data Size Register | |
| 03 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | C1ICVS | Class 1 ICV Size Register | |
| | 10 | | | | C2ICVS | Class 2 ICV Size Register | |
| 04 | 11 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | ISR | DECO ICID Status Register | |
| 05 | 11 | Control | See below | Yes | DCTRL2 | DECO Control Register 2 | See notes below. |
| | | | The DECO Control Register 2 is used to control the operation of DECO by means of a 1-word command that uses the LOAD command fields that normally represent OFFSET and LENGTH. This LOAD must be IMMEDIATE, which means that this DEST cannot be used with SEQ LOAD. The OFFSET and LENGTH fields are redefined as follows:<br>LENGTH[5]: If data for this job was prefetched by the JQ, invalidate the prefetched data so that subsequent reads of that data come from memory rather than the prefetch buffers. This is useful when the descriptor needs to modify the data and then rewinds the input frame.<br>LENGTH[4]: Transfer the value in the output frame tracking length register to the Variable Sequence Output Length Register.<br>LENGTH[0]: Set bit 5 of the PM EVENT Bus entry for this DECO for one clock.<br>All other bits of OFFSET and LENGTH are reserved and must be 0. | | | | |
| 06 | 00 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | CCTRL | CHA Control Register | |
| | 11 | Control | See below | Yes | DCTRL | DECO Control Register | See notes below. |

*Table continues on the next page...*

## Table 7-18. LOAD command DST, LENGTH, and OFFSET field values (continued)

| DST value<br>(hex) | Class<br>(binary) | Control data/ message data | Legal values in LENGTH/ OFFSET<br>fields | Must use IMM? | Tag | Internal register | Comment |
|---|---|---|---|---|---|---|---|
| | | | The DECO Control Register is used to control the operation of DECO by means of a 1-word command that uses the LOAD command fields that normally represent OFFSET and LENGTH. This LOAD must be IMMEDIATE, which means that this DEST cannot be used with SEQ LOAD. The OFFSET and LENGTH fields are redefined as follows:<br>OFFSET[7:6]: ICID Select for non-sequence operations (KEY, LOAD, FIFO STORE)<br>00 no change<br>01 SEQ ICID<br>10 Non-SEQ ICID (default)<br>11 trusted ICID (error if descriptor not trusted)<br>OFFSET[5:4]: ICID Select for sequence operations (SEQ KEY, SEQ LOAD, SEQ FIFO STORE)<br>00 no change<br>01 SEQ ICID (default)<br>10 non-SEQ ICID<br>11 trusted ICID (error if descriptor not trusted)<br>OFFSET[3]: Disable Automatic NFIFO Entries (If disable and enable are both set, disable dominates)<br>OFFSET[2]: Enable Automatic NFIFO Entries<br>OFFSET[1:0]: Change Share Type<br>00 no change<br>01 NEVER share<br>10 OK to share, do propagate errors<br>11 OK to share, don't propagate errors<br>LENGTH[7]: Turn On Output Sequence Length Counting (turned off by doing sequence output pointer rewind)<br>LENGTH[6]: Reset CHA pointer in Output Data FIFO<br>LENGTH[5]: Reset Output Data FIFO<br>LENGTH[4]: Process the Output Data FIFO Offset Field (automatically stalls if write burster is busy)<br>LENGTH[3]: Reserved<br>LENGTH[2:0]: Output Data FIFO Offset | | | | |
| 07 | 00 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | ICTRL | IRQ Control Register | - |

*Table continues on the next page...*

## Table 7-18. LOAD command DST, LENGTH, and OFFSET field values (continued)

| DST value (hex) | Class (binary) | Control data/ message data | Legal values in LENGTH/ OFFSET fields | Must use IMM? | Tag | Internal register | Comment |
|---|---|---|---|---|---|---|---|
| | 11 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | DPOVRD | DECO Protocol Override Register | If bit 31 = 1 the value loaded into DPOVRD overrides the default values in some protocol PDB fields. See individual protocol sections for usage details.<br>If bit 31 = 0, DPOVRD is not used as an override by the built-in protocols.<br>The other bits are defined on a protocol by protocol basis.<br>This register may be used as a source or destination by MATH and MATHI commands. |
| 08 | 00 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | CLRW | Clear Written Register | LOAD to the Clear Written Register will block if there are outstanding loads to a Context Register and the Class 1 Key Register is to be cleared. |
| | 11 | Control | 0-64/<br>0-7 bytes | No | MATH0 W | DECO Math Register 0 (Words) | 1, 2 |
| 09 | 11 | Control | 0-56/<br>0-7 bytes | No | MATH1 W | DECO Math Register 1 (Words) | 1, 2 |
| 0A | 11 | Control | 0-48/<br>0-7 bytes | No | MATH2 W | DECO Math Register 2 (Words) | 1, 2 |
| | 00 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | CISEL | CHA Instance Select Register | - |
| 0B | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | AADSZ | AAD Size Register | - |
| | 11 | Control | 0-40/<br>0-7 bytes | No | MATH3 W | DECO Math Register 3 (Words) | 1, 2 |
| 0C | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | C1IVSZ | Class 1 IV Size Register | - |
| 0C | 11 | Control | 0-32/<br>0-7 bytes | No | MATH4 W | DECO Math Register 4 (Words) | 1, 2 |
| 0D | 11 | Control | 0-24/<br>0-7 bytes | No | MATH5 W | DECO Math Register 5 (Words) | 1, 2 |

*Table continues on the next page...*

## Table 7-18. LOAD command DST, LENGTH, and OFFSET field values (continued)

| DST value (hex) | Class (binary) | Control data/ message data | Legal values in LENGTH/ OFFSET fields | Must use IMM? | Tag | Internal register | Comment |
|---|---|---|---|---|---|---|---|
| 0E | 11 | Control | 0-16/<br>0-7 bytes | No | MATH6 W | DECO Math Register 6 (Words) | 1, 2 |
| 0F | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | ALTDS1 | Alternate Data Size Class 1 Register<br>(aliased to the Class 1 Data Size Register) | The ALTDS1 destination can be used only with a LOAD Immediate command. Writes to the ALTDS1 block if there are any outstanding context loads since a write to a data size register indicates that the corresponding context is in place and ready. The alternate address for the Class 1 Data Size register limits handling of the data type "Special Authentication Data" as authentication/protection to the performance counters. |
| 0F | 11 | Control | 0-8/<br>0-7 bytes | No | MATH7 W | DECO Math Register 7 (Words) | 1, 2 |
| 10 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | PKASZ | PKHA A Size Register | This holds the size of the data in the PKHA A RAM.<br>The descriptor writer must ensure that any bits written to this register above the width of the register are 0. |
| 11 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | PKBSZ | PKHA B Size Register | This holds the size of the data in the PKHA B RAM.<br>The descriptor writer must ensure that any bits written to this register above the width of the register are 0. |
| 12 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | PKNSZ | PKHA N Size Register | This holds the size of the data in the PKHA N RAM.<br>The descriptor writer must ensure that any bits written to this register above the width of the register are 0. |
| 13 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | PKESZ | PKHA E Size Register | This holds the size of the data in the PKHA E RAM.<br>The descriptor writer must ensure that any bits written to this register above the width of the register are 0. |

*Table continues on the next page...*

## Table 7-18. LOAD command DST, LENGTH, and OFFSET field values (continued)

| DST value<br>(hex) | Class<br>(binary) | Control data/ message data | Legal values in LENGTH/ OFFSET<br>fields | Must use IMM? | Tag | Internal register | Comment |
|---|---|---|---|---|---|---|---|
| 20 | 01 | Message | 0-128/<br>0-128 bytes | No | CTX1 | Class 1 Context Register | A LOAD IMM to a context register blocks if there are any outstanding external loads to either context register.<br>A non_IMM LOAD to a context register blocks if the CCB DMA is writing to either context register. |
|  | 10 | Message | 0-128/<br>0-128 bytes | No | CTX2 | Class 2 Context Register |  |
| 30 | 11 | Control | 0-64/<br>0 bytes | No | MATH0DW | DECO Math Register 0 (Double Word) | 1, 3 |
| 31 | 11 | Control | 0-56/<br>0 bytes | No | MATH1DW | DECO Math Register 1 (Double Word) | 1, 3 |
| 32 | 11 | Control | 0-48/<br>0 bytes | No | MATH2DW | DECO Math Register 2 (Double Word) | 1, 3 |
| 33 | 11 | Control | 0-40/<br>0 bytes | No | MATH3DW | DECO Math Register 3 (Double Word) | 1, 3 |
| 34 | 11 | Control | 0-32/<br>0 bytes | No | MATH4DW | DECO Math Register 4 (Double Word) | 1, 3 |
| 35 | 11 | Control | 0-24/<br>0 bytes | No | MATH5DW | DECO Math Register 5 (Double Word) | 1, 3 |
| 36 | 11 | Control | 0-16/<br>0 bytes | No | MATH6DW | DECO Math Register 6 (Double Word) | 1, 3 |
| 37 | 11 | Control | 0-8/<br>0 bytes | No | MATH7DW | DECO Math Register 7 (Double Word) | 1, 3 |
| 38 | 11 | Control | 0-64/<br>0-7 bytes | No | MATH0B | DECO Math Register 0 (Bytes) | 1, 4 |
| 39 | 11 | Control | 0-56/<br>0-7 bytes | No | MATH1B | DECO Math Register 1 (Bytes) | 1, 4 |
| 3A | 11 | Control | 0-48/<br>0-7 bytes | No | MATH2B | DECO Math Register 2 (Bytes) | 1, 4 |
| 3B | 11 | Control | 0-40/<br>0-7 bytes | No | MATH3B | DECO Math Register 3 (Bytes) | 1, 4 |
| 3C | 11 | Control | 0-32/<br>0-7 bytes | No | MATH4B | DECO Math Register 4 (Bytes) | 1, 4 |
| 3D | 11 | Control | 0-24/<br>0-7 bytes | No | MATH5B | DECO Math Register 5 (Bytes) | 1, 4 |

*Table continues on the next page...*

## Table 7-18. LOAD command DST, LENGTH, and OFFSET field values (continued)

| DST value<br>(hex) | Class<br>(binary) | Control data/ message data | Legal values in LENGTH/ OFFSET<br>fields | Must use IMM? | Tag | Internal register | Comment |
|---|---|---|---|---|---|---|---|
| 3E | 11 | Control | 0-16/<br>0-7 bytes | No | MATH6B | DECO Math Register 6 (Bytes) | 1, 4 |
| 3F | 11 | Control | 0-8/<br>0-7 bytes | No | MATH7B | DECO Math Register 7 (Bytes) | 1, 4 |
| 40 | 01 | Message | 0-96/0-95 bytes | No | KEY1 | Class 1 Key Register | The key registers are normally written by the KEY Command, but can be written by a LOAD Command using this DST value. In this case the KEY SIZE register must be written by a separate command after the KEY register has been loaded.<br>A LOAD IMM to a key register blocks if there are any outstanding external loads to either key register.<br>A non_IMM LOAD to a key register blocks if the CCB DMA is writing to either key register. |
|  | 10 | Message | 0-128/<br>0-127 bytes | No | KEY2 | Class 2 Key Register |  |
|  | 11 | Control | 1-64/<br>1-63 | No | DESC<br>BUF | DECO descriptor buffer | See comments below. |
|  |  |  | For LOADs into the Descriptor Buffer the values in the LENGTH and OFFSET field are specified in 4-byte words.<br>An error is generated if the sum of the LENGTH and OFFSET fields is greater than 64. The OFFSET is used to specify the starting word of the destination within the descriptor buffer. Note that the OFFSET is relative to the start of the descriptor buffer. For SEQ LOAD, the data written into the descriptor buffer is read from the current location pointed to by the input sequence pointer (there is no offset with respect to the source address). |  |  |  |  |
| 70 | 00 | Control | 4 or 8/<br>0 bytes | Yes | NFSL | NFIFO and size register(s) | Using the Immediate data, write an NFIFO entry and load the Size register from the DL or PL field in that NFIFO entry. Also see note below. |
| - | - | - | This creates an NFIFO entry from 4 or 8 bytes of IMM data and also writes to one or more size registers. The entry's DL or PL field is filled in with the same value loaded into the size register(s). The table below titled "Which Size Registers are loaded" indicates which size registers are loaded. |  |  |  |  |
| 71 | 00 | Control | 0-7<br>0 bytes | Yes | NFSM | NFIFO and size register(s) | Using the Immediate data write an NFIFO entry and load the size register using for PL or DL the least-significant 32-bits of the MATH register selected by means of the three ls bits of the LENGTH field. Also see note above. |

*Table continues on the next page...*

## Table 7-18. LOAD command DST, LENGTH, and OFFSET field values (continued)

| DST value (hex) | Class (binary) | Control data/ message data | Legal values in LENGTH/ OFFSET fields | Must use IMM? | Tag | Internal register | Comment |
|---|---|---|---|---|---|---|---|
| 72 | 00 | Control | 4 or 8/<br>0 bytes | Yes | NFL | NFIFO | Using the Immediate data write an NFIFO entry. This is equivalent to DST value 7Ah. Also see note below. |
| - | - | - | This creates an NFIFO entry from 4 or 8 bytes of IMM data. No size registers are written. | | | | |
| 73 | 00 | Control | 0-7<br>0 bytes | Yes | NFM | NFIFO | Using the Immediate data write an NFIFO entry filling in the DL or PL field from the least-significant 32-bits of the MATH register selected via the three ls bits of the LENGTH field. Also see note above. |
| 74 | 00 | Control | 4 or 8/<br>0 bytes | Yes | SL | Size register(s) | With the Immediate data in NFIFO entry format, load the size register(s) from the DL or PL field in the Immediate data but do not write an entry into the NFIFO. The table below titled "Which Size Registers are loaded" indicates which size registers are loaded. |
| 75 | 00 | Control | 0-7<br>0 bytes | Yes | SM | Size register(s) | Load the size register(s) from the value in the MATH register that is selected by means of the three ls bits of the LENGTH field. No NFIFO entry is loaded. The table below titled "Which Size Registers are loaded" indicates which size registers are loaded. |
| **NOTE:** For DST values 70h, 71h, 74h, and 75h, the particular size registers that are loaded depend on the CHAs that are selected and the DTYPE field of the entry that is written into the NFIFO. The table below titled "Which Size Registers are loaded" indicates which size registers are loaded.<br>**NOTE:** For DST values 70h, 72h, 74h, and 7Ah, the direct destination depends on the value in the LENGTH field. If the LENGTH is 4, then the direct destination is the NFIFO. However, if the LENGTH field is 8, then the direct destination is special control hardware which breaks up large lengths so that the maximum length permitted in an NFIFO entry is not exceeded. This hardware pushes as many entries into the NFIFO as necessary. The special control hardware will stall if the NFIFO is full, resuming when space becomes available.<br>**NOTE:** For DST values 70h-75h and 7Ah, the LOAD will block if the NFIFO is full. In addition, further access to the NFIFO will block if the hardware which breaks up large entries is in use. | | | | | | | |
| 76 | 00 | Message | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | IDFNS | Input Data FIFO Nibble Shift Register | See notes below. |
| | | | Inserts the rightmost 4 bits of the immediate value into the input to the Class 1 Alignment Block, which causes the remainder of the input data to be shifted by one nibble. This nibble alignment continues until the L1 bit or F1 bit in an NFIFO entry is encountered. Thereafter, input to the Class 1 Alignment Block will not be nibble shifted unless the IDFNSR is written | | | | |

*Table continues on the next page...*

## Table 7-18. LOAD command DST, LENGTH, and OFFSET field values (continued)

| DST value (hex) | Class (binary) | Control data/ message data | Legal values in LENGTH/ OFFSET fields | Must use IMM? | Tag | Internal register | Comment |
|---|---|---|---|---|---|---|---|
| | | | again. Any nibble remaining in the shift register will remain there once the last or flush is seen. This means that if there are N bytes of data, to get the last nibble out requires NFIFO entries totaling N+1 bytes. | | | | |
| 77 | 00 | Message | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | Yes | ODFNS | Output Data FIFO Nibble Shift Register | See notes below. |
| | | | Inserts the rightmost 4 bits of the immediate value into the output from a Class 1 CHA, so subsequent data from that Class 1 CHA is now shifted one nibble. Data from other sources (such as MOVE Command or LOAD IMM to the Output Data FIFO) will not be concatenated correctly. This nibble alignment continues until the CHA Done signal is asserted. Thereafter, output from a second operation, even from the same CHA, is not nibble shifted unless the ODFNSR is written again. Any valid nibble remaining is always pushed into the output FIFO following the assertion of CHA Done. | | | | |
| 78 | 00 | Message | 1-8/0 bytes | Yes | AUXDATA | Auxiliary Data FIFO | See notes below. |
| | | | This DST value can be used to provide data to the Auxiliary Data FIFO. Each LOAD IMM command can load 1-8 left-aligned bytes. Byte swapping is done automatically if the endianness settings require it. The LOAD IMM to AUXDATA will stall if there is no room left in the AUXDATA buffer. The AUXDATA path should be treated in the same way as the input DATA FIFO. If NFIFO entries are not used properly, execution will hang if more LOADs are done to the AUXDATA buffer but room can't be created by draining that buffer via NFIFO entries and the corresponding alignment block being drained by CHAs or the MOVE command. Note that data can also be supplied to the Auxiliary Data FIFO by using a MOVE command. | | | | |
| 7A | 00 | Control | 4/0 bytes<br>8/0 bytes<br>** | Yes | NFIFO | NFIFO | See notes below. |
| | | | The NFIFO is normally written by means of the FIFO LOAD command, but the NFIFO can be written using this DST value with a LOAD Immediate command.<br>** If LENGTH = 8, the LOAD command is interpreted as follows:<br>word 1: LOAD IMM, LENGTH = 8, DST = 7Ah<br>word 2: bits [31:12] contain the NFIFO entry if not padding type, else bits [31:10] contain the NFIFO entry<br>word 3: Extended Length (DECO creates as many NFIFO entries as needed to satisfy the Extended Length - see the notes above) | | | | |
| 7C | 00 | Message | 1-8/0 bytes | Yes | IFIFO | Input Data FIFO | See notes below |
| | | | The input data FIFO is normally written by means of the FIFO LOAD command, but the Input Data FIFO can be written using this DST value with a LOAD Immediate Command. The data must be left-aligned and byte swapping will be done if the endianess settings require it. This LOAD will block if there is no more room in the input data FIFO or if there is other data heading to the input data FIFO. Care should be take since this block could turn into a hang if the LOAD is unable to proceed. | | | | |

*Table continues on the next page...*

**Table 7-18. LOAD command DST, LENGTH, and OFFSET field values (continued)**

| DST value (hex) | Class (binary) | Control data/ message data | Legal values in LENGTH/ OFFSET fields | Must use IMM? | Tag | Internal register | Comment |
|---|---|---|---|---|---|---|---|
| 7E | 00 | Message | 1-8/0 bytes | Yes | OFIFO | Output Data FIFO | Must use a LOAD Immediate command. The data must be left-aligned and byte swapping will be done if the endianess settings require it. This LOAD will block if there is no more room in the output data FIFO. It is up to the descriptor writer to ensure that this LOAD will not collide with a move to the output FIFO or a push to the output FIFO by one of the C1 CHAs. Care should be take since this block could turn into a hang if the LOAD is unable to proceed. |
| All combinations of value and class that do not appear in this table are reserved | | | | | | | |

1. May be affected by protocols. Note that using the LOAD command to place values in the math registers does not update the MATH status bits (see MNV, MN, MC and MZ). Because the math registers are in contiguous addresses, it is possible to load more than one math register simultaneously. A LOAD IMM to a math register blocks if there are any outstanding external loads to any math register. A non_IMM LOAD to a math register blocks if the CCB DMA is writing to any math register.
2. When this destination is used, the data loaded into the Math register will be treated as words.
3. When this destination is used, the data loaded into the Math register will be treated as double words. Offset must be 0. Word swapping will be handled the same as address pointers. It is recommended that only full double words be loaded.
4. When this destination is used, the data loaded into the Math register will be treated as bytes.

The following table details which size registers are written when loading NFIFO entries when one, or more, size registers are also written. (DST values 70h, 71h, 74h, and 75h.) An entry of "None/Reserved" means that no size register will be written and NXP reserves the right to assign that DTYPE to some size register in the future. Therefore, such DTYPEs should not be used as they could break compatibility.

**Table 7-19. Which Size Registers Are Loaded**

| DTYPE (hex) | If PKHA selected | If Class 1 CHA selected, but not PKHA | If Class 2 CHA selected |
|---|---|---|---|
| 0 | PKHA A Size | None/Reserved | Class 2 Data Size |
| 1 | PKHA A Size | AAD Size and Class 1 Data Size | Class 2 Data Size |
| 2 | PKHA A Size | IV Size and Class 1 Data Size | Class 2 Data Size |
| 3 | PKHA A Size | AAD Size and Class 1 Data Size<br>(but data only counts as Auth data) | Class 2 Data Size |
| 4 | PKHA B Size | None/Reserved | If PKHA selected, Class 2 Data Size else None/Reserved |

*Table continues on the next page...*

Table 7-19. Which Size Registers Are Loaded (continued)

| DTYPE<br>(hex) | If PKHA selected | If Class 1 CHA selected, but not PKHA | If Class 2 CHA selected |
|---|---|---|---|
| 5 | PKHA B Size | None/Reserved | If PKHA selected, Class 2 Data Size else None/Reserved |
| 6 | PKHA B Size | None/Reserved | If PKHA selected, Class 2 Data Size else None/Reserved |
| 7 | PKHA B Size | None/Reserved | If PKHA selected, Class 2 Data Size else None/Reserved |
| 8 | PKHA N Size | None/Reserved | If PKHA selected, Class 2 Data Size else None/Reserved |
| 9 | PKHA E Size | None/Reserved | If PKHA selected, Class 2 Data Size else None/Reserved |
| A | None/Reserved | ICV Size | If PKHA selected: Class 2 ICV Size<br>If PKHA not selected: if both Class 1 and Class 2 CHAs selected, Class 2 Data Size else Class 2 ICV Size |
| B | None/Reserved | None/Reserved | None/Reserved |
| C | PKHA A Size | None/Reserved | If PKHA selected, Class 2 Data Size else None/Reserved |
| D | PKHA B Size | None/Reserved | If PKHA selected, Class 2 Data Size else None/Reserved |
| E | None/Reserved | None/Reserved | None/Reserved |
| F | None/Reserved | Class 1 Data Size | Class 2 Data Size |

## 7.11 FIFO LOAD command

**NOTE**

In the following discussion, the term 'FIFO LOAD command' refers to both the SEQ and non-SEQ forms of the command.

FIFO LOAD commands are used to load message data, PKHA data (other than for the E Memory), IV, AAD, ICV, and bit-length message data into the input data FIFO. The SEQ FIFO LOAD command is identical to the FIFO LOAD command except that no address is specified, the command contains an AIDF bit in place of the IMM bit, and the VLF bit instead of the SGF bit. See SEQ vs non-SEQ commands.

As the only destination is the input data FIFO, this command does not include a DST field. The FIFO INPUT DATA TYPE is used to indicate what type of data is being loaded and whether the length is specified in bits or bytes. The length of data other than message data is measured in bytes. The length of message data can be specified in either bits or bytes (see Table 7-23). If automatic info FIFO entries are enabled, the FIFO LOAD command writes the appropriate size register(s) and the required info FIFO entry for the specified input data type. This command will block for a variety of reasons:

1. FIFO LOAD IMM will block if the input FIFO is full.
2. FIFO LOAD IMM will block if the DMA is required to move the data but the DMA is busy.
3. If a DMA transaction is required, the FIFO LOAD command will block if the hardware which schedules DMA transactions is in use.
4. If there are external reads destined for the input data FIFO, FIFO LOAD IMM will block until that data arrives.
5. The FIFO LOAD command uses the same logic as the LOAD command does when loading NFIFO entries with LENGTH=8. If an NFIFO entry is required and this logic is busy, the command will block.
6. A SEQ FIFO LOAD SKIP will block if there is a buffer release pending. Once the pending release completes, the command will proceed.

### Table 7-20. FIFO LOAD command format

<table>
<tr><th>31–27</th><th>26–25</th><th>24</th><th>23</th><th>22</th><th>21–16</th></tr>
<tr><td>CTYPE = 00100 or 00101</td><td>CLASS</td><td>SGF or VLF</td><td>IMM or AIDF</td><td>EXT</td><td>INPUT DATA TYPE</td></tr>
<tr><th colspan="6">15–0</th></tr>
<tr><td colspan="6">LENGTH</td></tr>
<tr><td colspan="6"><i>Additional words of FIFO LOAD command:</i></td></tr>
<tr><td colspan="6">Pointer (one or two words, see Address pointers) or Value (if immediate, one or more words)</td></tr>
<tr><td colspan="6"></td></tr>
<tr><td colspan="6">EXT LENGTH (two words, present if EXT=1)</td></tr>
</table>

### Table 7-21. FIFO LOAD command field descriptions

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>If CTYPE=00100b : FIFO LOAD command<br>If CTYPE=00101b : SEQ FIFO LOAD command |
| 26-25<br>CLASS | Class. Cryptographic algorithm class.<br>If CLASS=00b : Used for SEQ FIFO LOAD only. Skips the specified length in memory without scheduling any read transactions and no data is actually read. However, Scatter/Gather Table entries will be read as needed. FIFO INPUT DATA TYPE field is ignored. No info FIFO entry is generated.<br>If CLASS=01b : Load FIFO with data for a Class 1 CHA.<br>If CLASS=10b : Load FIFO with data for a Class 2 CHA.<br>If CLASS=11b : Load FIFO with data for both Class 1 and Class 2 CHAs (both In Snoop and Out Snoop; the INPUT DATA TYPE will distinguish between them). |

*Table continues on the next page...*

## Table 7-21. FIFO LOAD command field descriptions (continued)

| Field | Description |
|---|---|
| | **NOTE:** The CLASS field must be non-zero for FIFO LOAD commands because the 00b case indicates skipping, which is illegal for FIFO LOAD. This is true even when automatic information FIFO entries are disabled. |
| 24<br>SGF or VLF | Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF).<br>If CTYPE = 00100b (FIFO LOAD), this bit is the Scatter/Gather table Flag (SGF).<br>If SGF=0, the pointer points to actual data.<br>If SGF=1, the pointer points to a Scatter/Gather Table.<br>**NOTE:** If the IMM bit is set, it is an error for this bit to be set.<br>If CTYPE = 00101b (SEQ FIFO LOAD), this bit is the Variable Length Flag (VLF).<br>If VLF=0, the LENGTH field indicates the length of the data.<br>If VLF=1, the length is variable. SEC uses the length in the Variable Sequence In Length register and ignores the LENGTH field.<br>**NOTE:** It is an error to set VLF = 1 when the EXT bit = 1. |
| 23<br>IMM or AIDF | Immediate Flag(IMM) or Already in Input Data FIFO (AIDF)<br>If CTYPE = 00100 (FIFO LOAD), this bit is the Immediate Flag (IMM).<br>If IMM=0, the data begins at the location pointed to by the Pointer field.<br>If IMM=1, the data follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word.<br>**NOTE:** It is an error if this bit is set when SGF = 1 or EXT = 1.<br>If CTYPE = 00101 (SEQ FIFO LOAD), this is the Already in Data FIFO (AIDF) bit.<br>If AIDF is 0, SEC will read the input sequence data from memory.<br>If AIDF is 1, SEC will not read the input sequence data (because it is already in the Input Data FIFO). This form is convenient since the NFIFO and Data Size Registers will be loaded automatically if Automatic Info FIFO Entries is enabled. As a result, a 1-word command can replace two 2-word commands. |
| 22<br>EXT | Extended Length<br>If EXT=0 : Input data length is solely determined by the 16-bit LENGTH field,<br>If EXT=1 : Input data length is determined by the 32-bit EXTENDED LENGTH. If the INPUT DATA TYPE indicates a bit length, then the EXTENDED LENGTH field contains the number of full bytes and the right 3 bits of the LENGTH field, if nonzero, indicate the number of valid bits in the last byte. See Bit length data.<br>**NOTE:** It is an error if this bit is set when IMM is also set. |
| 21-16<br>INPUT DATA TYPE | FIFO input data type<br>See Table 7-23 for a description of the supported types. When automatic information FIFO entries are disabled, (SEQ) FIFO LOAD Commands ignore the FIFO INPUT DATA TYPE field. |
| 15-0<br>LENGTH | Length of data<br>If EXT = 0 : LENGTH = number of bytes of input data, or for bit length message data, the number of bits of input data.<br>If EXT = 1 : The EXT LENGTH field indicates the number of full bytes of data. The LENGTH field is ignored (unless if FIFO INPUT DATA TYPE = bit length, in which case the least-significant 3 bits of the LENGTH field indicate the number of valid bits in an additional byte of data).<br>See Bit length data. |
| *Additional words of FIFO LOAD command:* | |

*Table continues on the next page...*

**Table 7-21. FIFO LOAD command field descriptions (continued)**

| Field | Description |
|---|---|
| POINTER | If IMM = 0, this field is a pointer to the data to be loaded.<br>If IMM = 1, this field is not present.<br>**NOTE:** This field is not present for SEQ FIFO LOAD Commands. |
| | |
| 31-0<br>EXT LENGTH | For EXT = 0, this field not present.<br>For EXT = 1, EXTENDED LENGTH specifies number of full bytes of data to load. For bit-length data, the least-significant 3 bits of the LENGTH field indicate the number of valid bits in an additional byte of data. See Bit length data. |

## 7.11.1 Bit length data

If the INPUT DATA TYPE indicates that the input data type being loaded is bit-length message data, the LENGTH field is defined as a bit count, as shown in the "Number of Bits" row in the following figure. This can also be interpreted as a "Number of Full Bytes field" in bits positions 15-3, and a "Number of Additional Valid Bits" field in bit positions 2-0. These additional valid bits are in the next byte after the number of full bytes, starting with the bit on the left. For example, if the LENGTH field is 0101h, SEC loads 33 bytes, with only the leftmost bit of the 33rd byte valid. Note that the entire 33rd byte is read and it is up to the consumer of that last byte to know that only the specified number of bits in the last byte are valid.

The Number of Additional Valid Bits is placed in the NUMBITS field of the Class 1 and/or Class 2 Data Size Register. The NUMBITS field is not visible to any functional logic in SEC other than a subset of the CHAs. (The NUMBITS field may be read via SkyBlue or a store of the Data Size Registers.) The CHAs which receive the NUMBITS field are:

- KFHA
- SNOW F8 and SNOW F9
- ZUCA and ZUCE
- AESA, which will error if it sees a nonzero NUMBITS field.

The following CHAs do not receive the NUMBITS field:

- PKHA
- DES
- CRCA
- MDHA
- RNG

It is possible to use a nonzero NUMBITS field with a CHA which does not receive the NUMBITS field. To do this, add 1 more to the proper Data Size Register. However, note that the remaining bits in the last byte will be whatever values they were at the source of that byte. That is, the remaining bits are not masked to 0.

It is not possible for SEC to automatically concatenate two separate bit fields. For example, if an NFIFO entry for 3 bits is followed by an entry for 5 bits, these entries will NOT result in a one-byte entry. To achieve such concatenation, use the shift operations in the MATH command.

When using automatic NFIFO entries with the FIFO LOAD command to specify bit lengths, C1 must always have Flush or Last set and C2 must always have Last set. Failure to set the Last and Flush bits as stated will result in an error. When manually generating NFIFO entries, no error will be generated if the Flush or Last bits are not set as suggested. However, as stated above, each incomplete byte's remaining bits will have whatever values they had at the source.

If the FIFO LOAD command is used to generate the NFIFO entry, the number of bytes specified in the LENGTH field for the NFIFO entry will be the number of full bytes if the NUMBITS field is zero, and the number of full bytes plus one if the NUMBITS field is nonzero. This ensures that the CHA will consume the last, partial, byte. If the descriptor writer is manually generating the NFIFO entries, care must be taken to handle the length properly.

**Table 7-22. Specifying data with residual bit length**

| 15–0 | |
|---|---|
| Number of Bits | |
| ***Alternate interpretation:*** | |
| **15–3** | **2–0** |
| Number of Full Bytes | Number of Additional Valid Bits |

If the input data's bit length is equal to or greater than $2^{16}$, set the EXT bit and use the EXTENDED LENGTH field to specify the number of full bytes. The upper 13 bits of LENGTH must be zero, with the rightmost 3 bits specifying the number of additional valid bits as before.

## 7.11.2 FIFO LOAD input data type

Table 7-23 contains an enumeration of the various built-in input data FIFO data types. This field is ignored if neither of the Class bits are set. Only message data can have a bit length (as opposed to a byte length), and such message data must always have Flush or Last set.

Note that the NFIFO source type is not needed, as it is always inferred from the FIFO LOAD input data type. The length for the NFIFO entry is the amount of data being placed in the data FIFO. (One is added to the length in the case of bit-length data if the number of bits in the last byte is nonzero.) Also, the Last and Flush bits are always sent as 0 except with the last byte of data, in which case the values shown in the table are sent.

Note also that data should not be left in the Input Data FIFO with the expectation that it will be shared with a subsequent shared descriptor executing in the same DECO. This may cause data to be lost if the next shared job executes in a different DECO.

With the exception of IV and AAD, the FIFO LOAD command does not do any padding. This is because all algorithmic padding requires a pad length or a special last byte, which means that at least one byte of padding is required. Therefore, the padding can be sent using a padding NFIFO entry.

**Table 7-23. FIFO LOAD input data type field**

| FIFO Input Type Field Bit # | | | | | | Meaning |
|---|---|---|---|---|---|---|
| 21 | 20 | 19 | 18 | 17 | 16 | |
| 00b | | PKHA | | | | PKHA Register Load<br>All values not specified below are reserved. This data is always flushed. An error is asserted if the length is larger than fits in the PKHA RAM.<br>**NOTE:** Loading quadrants of a given PKHA register with different-sized values may cause invalid data to be loaded into the quadrants. To avoid this issue, make sure that all quadrants of a given register have the same size values by left-filling short values with zero. If it is necessary to load different-sized values in quadrants of the same register, insert a JUMP command between quadrant loads (which will wait for automatic information FIFO entries to be processed): jump jsl = 1, type = 0, cond = nifp, local offset = 1.<br>**NOTE:** The PKHA E RAM can not be loaded via the FIFO LOAD command using automatic NFIFO entries. Use the KEY command or get the data into the input data FIFO without an automatic NFIFO entry and then manually create an NFIFO entry and write the PKHA E Size register. |
| | | 0 | 0 | 0 | 0 | PKHA A0 |
| | | 0 | 0 | 0 | 1 | PKHA A1 |
| | | 0 | 0 | 1 | 0 | PKHA A2 |
| | | 0 | 0 | 1 | 1 | PKHA A3 |
| | | 0 | 1 | 0 | 0 | PKHA B0 |
| | | 0 | 1 | 0 | 1 | PKHA B1 |
| | | 0 | 1 | 1 | 0 | PKHA B2 |

*Table continues on the next page...*

**Table 7-23. FIFO LOAD input data type field (continued)**

| FIFO Input Type Field Bit # | | | | | | Meaning |
|---|---|---|---|---|---|---|
| 21 | 20 | 19 | 18 | 17 | 16 | |
| | | 0 | 1 | 1 | 1 | PKHA B3 |
| | | 1 | 0 | 0 | 0 | PKHA N |
| | | 1 | 1 | 0 | 0 | PKHA A |
| | | 1 | 1 | 0 | 1 | PKHA B |
| 00b | | 1 | 1 | 1 | 1 | Place the data into the input Data FIFO but do not generate an NFIFO entry and do not write any size registers, even if automatic NFIFO entries are enabled. |
| 010b | | | LC2 | LC1 | FC1 | Message Data |
| 011b | | | LC2 | LC1 | FC1 | Message Data for Class 1 out-snooped to Class 2 |
| 100b | | | LC2 | LC1 | FC1 | IV. If Last or Flush for Class 1 is set, Class 1 is padded to 16-byte boundary with 0. No padding is done for Class2. No padding is done for Class 1 if the data naturally ends on a 16-byte boundary. |
| 101b | | | 1 | LC1 | FC1 | Bit-length message data. Last of Class 2 treated as set even if not set in the command; either Flush or Last for Class 1 must be set |
| 110b | | | LC2 | LC1 | FC1 | AAD. If Last or Flush for Class 1 is set. Class 1 is padded to a 16-byte boundary with 0. No padding is done for Class 2. It is an error if Class 2 is specified and Class 1 is not. |
| 111b | | | LC2 | LC1 | FC1 | ICV |
| If CLASS 1 is asserted, LC1 means the data defined in the current NFIFO entry is the last data. When LC1 = 1, the last byte of the specified length is made available from the Class 1 Alignment Block even if that last byte does not complete an 8-byte entry. If CLASS 1 is negated, LC1 is ignored.<br>If CLASS 2 is asserted, LC2 means the data defined in the current NFIFO entry is the last data. When LC2 = 1, the last byte of the specified length is made available from the Class 2 Alignment Block even if that last byte does not complete an 8-byte entry. If CLASS 2 is negated, LC2 is ignored<br>If CLASS 1 is asserted, FC1 means the data defined in the current NFIFO entry is the last data of this type. When FC1 = 1, the last byte of the specified length is made available from the Class 1 Alignment Block even if that last byte does not complete an 8-byte entry. If CLASS 1 is negated, FC1 is ignored.<br>Note that the difference between LC1 and FC1 is only important when the data is going to a CHA. If the data is to be consumed by the CCB DMA, then FC1 should be used as LC1 may confuse a CHA. Similarly, LC2 should not be used if the data will be consumed by the CCB DMA. In such cases, it is better to use a manual NFIFO entry with the FC2 bit set. | | | | | | |
| All values not specified are reserved. | | | | | | |

## 7.12 ECPARAM command

The ECPARAM command is used to load one parameter from a set of built-in elliptic curve parameters into one of the PKHA registers. This command will block until any data already in transit to the Input Data FIFO has been received.

The DEST value 1111b is used to leave the selected parameter in the Input Data FIFO. No NFIFO entry is generated. If automatic NFIFO entries have been disabled, the ECPARAM command will leave the parameter in the Input Data FIFO and no NFIFO entry is generated. Note that when no NFIFO entry is generated, no PKHA size register is written. Furthermore, the descriptor writer is responsible for the NFIFO entry to get this parameter to its final destination.[3, 4] In all other legal cases, the ECPARAM command will result in loading the parameter into the selected PKHA register by automatically creating the NFIFO entry and automatically writing the correct PKHA size register with the correct value.

### Table 7-24. ECPARAM command, format

| 31–27 | | 26-20 | | 19–16 |
|---|---|---|---|---|
| CTYPE = 11100 | | 0100000 | | DEST |
| **15** | **14-8** | | **7–4** | **3–0** |
| Reserved | DOMAIN | | Reserved | PARAMETER |

### Table 7-25. ECPARAM command, field descriptions

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>IF CTYPE=11100b : ECPARAM command |
| 26-20 | 0100000b. This particular 7-bit value is mandatory. Any other value will generate an error. |
| 19-16<br>DEST | Destination. This field specifies which PKHA register to load:<br>0000b - PKHA A0<br>0001b - PKHA A1<br>0010b - PKHA A2<br>0011b - PKHA A3<br>0100b - PKHA B0<br>0101b - PKHA B1<br>0110b - PKHA B2<br>0111b - PKHA B3<br>1000b - PKHA N<br>1100b - PKHA A<br>1101b - PKHA B<br>1111b - Input Data FIFO--This parameter passes through the Input Data FIFO. It is up to the descriptor writer to ensure that any data already in the Input Data FIFO already has corresponding NFIFO entries.<br>All other values are reserved. |
| 15 | Reserved |
| 14-8 | Elliptic Curve Domain. This field selects one of the built-in elliptic curve domains: |

*Table continues on the next page...*

[3] This parameter passes through the Input Data FIFO. It is up to the descriptor writer to ensure that any data already in the Input Data FIFO already has corresponding NFIFO entries.

[4] If parameters of different sizes are to be loaded into different quadrants of the same PKHA register, then it is up to the user to ensure that the first parameter is completely loaded before executing the second ECPARAM command. This is because the same size register is used and can't be changed for the second parameter until the first has been successfully loaded.

## Table 7-25. ECPARAM command, field descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>DOMAIN</td><td>
<table>
<tr><th>DOMAIN</th><th>ECC Domain Selected</th></tr>
<tr><td>0h</td><td>P-192</td></tr>
<tr><td>1h</td><td>P-224</td></tr>
<tr><td>2h</td><td>P-256</td></tr>
<tr><td>3h</td><td>P-384</td></tr>
<tr><td>4h</td><td>P-521</td></tr>
<tr><td>5h</td><td>brainpoolP160r1</td></tr>
<tr><td>6h</td><td>brainpoolP160t1</td></tr>
<tr><td>7h</td><td>brainpoolP192r1</td></tr>
<tr><td>8h</td><td>brainpoolP192t1</td></tr>
<tr><td>9h</td><td>brainpoolP224r1</td></tr>
<tr><td>Ah</td><td>brainpoolP224t1</td></tr>
<tr><td>Bh</td><td>brainpoolP256r1</td></tr>
<tr><td>Ch</td><td>brainpoolP256t1</td></tr>
<tr><td>Dh</td><td>brainpoolP320r1</td></tr>
<tr><td>Eh</td><td>brainpoolP320t1</td></tr>
<tr><td>Fh</td><td>brainpoolP384r1</td></tr>
<tr><td>10h</td><td>brainpoolP384t1</td></tr>
<tr><td>11h</td><td>brainpoolP512r1</td></tr>
<tr><td>12h</td><td>brainpoolP512t1</td></tr>
<tr><td>13h</td><td>prime192v2</td></tr>
<tr><td>14h</td><td>prime192v3</td></tr>
<tr><td>15h</td><td>prime239v1</td></tr>
<tr><td>16h</td><td>prime239v2</td></tr>
<tr><td>17h</td><td>prime239v3</td></tr>
<tr><td>18h</td><td>secp112r1</td></tr>
<tr><td>19h</td><td>wtls8</td></tr>
<tr><td>1Ah</td><td>wtls9</td></tr>
<tr><td>1Bh</td><td>secp160k1</td></tr>
<tr><td>1Ch</td><td>secp160r1</td></tr>
<tr><td>1Dh</td><td>secp160r2</td></tr>
<tr><td>1Eh</td><td>secp192k1</td></tr>
<tr><td>1Fh</td><td>secp224k1</td></tr>
<tr><td>20h</td><td>secp256k1</td></tr>
<tr><td>40h</td><td>B-163</td></tr>
<tr><td>41h</td><td>B-233</td></tr>
<tr><td>42h</td><td>B-283</td></tr>
<tr><td>43h</td><td>B-409</td></tr>
<tr><td>44h</td><td>B-571</td></tr>
</table>
</td></tr>
</table>

*Table continues on the next page...*

## Table 7-25. ECPARAM command, field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><td>45h</td><td>K-163</td></tr>
<tr><td>46h</td><td>K-233</td></tr>
<tr><td>47h</td><td>K-283</td></tr>
<tr><td>48h</td><td>K-409</td></tr>
<tr><td>49h</td><td>K-571</td></tr>
<tr><td>4Ah</td><td>wtls1</td></tr>
<tr><td>4Bh</td><td>sect113r1</td></tr>
<tr><td>4Ch</td><td>c2pnb163v1</td></tr>
<tr><td>4Dh</td><td>c2pnb163v2</td></tr>
<tr><td>4Eh</td><td>c2pnb163v3</td></tr>
<tr><td>4Fh</td><td>sect163r1</td></tr>
<tr><td>50h</td><td>sect193r1</td></tr>
<tr><td>51h</td><td>sect193r2</td></tr>
<tr><td>52h</td><td>sect239k1</td></tr>
<tr><td>53h</td><td>Oakley3a</td></tr>
<tr><td>54h</td><td>Oakley4a</td></tr>
<tr><td colspan="2">All values not specified are reserved.</td></tr>
</table>
</td></tr>
<tr><td>7-4</td><td>Reserved</td></tr>
<tr><td>3-0<br>PARAMETER</td><td>
Elliptic Curve Parameter. This field specifies which elliptic curve parameter is to be loaded into the PKHA register specified by DEST.
<table>
<tr><th>PARAMETER</th><th>EC Domain Parameter Selected</th></tr>
<tr><td>0h</td><td>q</td></tr>
<tr><td>1h</td><td>r [fn1]</td></tr>
<tr><td>2h</td><td>Gx</td></tr>
<tr><td>3h</td><td>Gy</td></tr>
<tr><td>4h</td><td>a</td></tr>
<tr><td>5h</td><td>b</td></tr>
<tr><td>6h</td><td>$R^2$mod q</td></tr>
<tr><td>7h</td><td>$R^2$mod r [fn1]</td></tr>
<tr><td>8h</td><td>c [fn2]</td></tr>
<tr><td>9h</td><td>k [fn1]</td></tr>
<tr><td colspan="2">All values not specified are reserved.</td></tr>
</table>
fn1: The "r", "$R^2$mod r" and "k" parameters are not valid for either OAKLEY domain.<br>
fn2: The "C" parameter is not valid for Fp domains (i.e. DOMAIN &lt; 40h).
</td></tr>
<tr><td></td><td></td></tr>
</table>

## 7.13 STORE command

**NOTE**

In the following discussion, the term 'STORE command' refers to both the SEQ and non-SEQ forms of the command.

STORE commands are used to read data from various registers and write them to a system address. The SEQ STORE command is identical to the STORE command except that no address is specified and the VLF bit replaces the SGF bit. See SEQ vs non-SEQ commands.

The definitions of the OFFSET and LENGTH fields in the STORE command can depend on the CLASS and source (SRC) fields. Table 7-26 shows the command fields, and Table 7-28 defines OFFSET and LENGTH as well as additional behaviors of the command.

As shown in the following table, STORE data sources can be both control and message data registers. Data stored from control data registers are regarded as word-oriented data, whereas data stored from message data registers are regarded as byte strings.

To facilitate operation in chips with different endianness configurations, the following data-swapping operations can be configured:

- byte-swapping
- half-word swapping
- word-swapping
- double-word swapping

and these swapping operations for control data registers can be configured independently from the swapping for message data registers.

The same swapping operations can be configured independently for:

- each job ring (see the Job Ring Configuration Register (JRCFGR))
- the Queue Manager Interface (see the Queue Interface Control Register (QICTL))

**Table 7-26. STORE command format**

<table>
<tr><th>31–27</th><th>26–25</th><th>24</th><th>23</th><th>22–16</th></tr>
<tr><td>CTYPE = 01010 or 01011</td><td>CLASS</td><td>SGF<br>or<br>VLF</td><td>IMM</td><td>SRC</td></tr>
<tr><th colspan="3">15–8</th><th colspan="2">7–0</th></tr>
<tr><td colspan="3">OFFSET</td><td colspan="2">LENGTH</td></tr>
<tr><td colspan="5"><i>Additional words of STORE command:</i></td></tr>
<tr><td colspan="5">Pointer ( one or two words, see Address pointers)</td></tr>
<tr><td colspan="5"></td></tr>
<tr><td colspan="5">If immediate (IMM = 1), one or more words of data appear here</td></tr>
</table>

## Table 7-27. STORE command field descriptions

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command Type<br>If CTYPE=01010b : STORE command<br>If CTYPE=01011b : SEQ STORE command |
| 26-25<br>CLASS | Algorithm class of the data object to be stored<br>See the SRC field for additional explanation. If IMM = 1 a value other than 00b in the CLASS field will cause an error to be generated.<br>If IMM = 0, the following definitions are used:<br>If CLASS=00b : Store class-independent objects from CCB.<br>If CLASS=01b : Store Class 1 objects from CCB.<br>If CLASS=10b : Store Class 2 objects from CCB.<br>If CLASS=11b : Store objects from DECO. |
| 24<br>SGF or VLF | Scatter/Gather Table Flag (SGF) or Variable Length Flag (VLF).<br>If CTYPE = 01010b (STORE), this bit is the Scatter/Gather table Flag (SGF).<br>If SGF=0, the pointer contains the address of the destination for the data to be stored.<br>If SGF=1, the pointer points to a Scatter/Gather Table, which defines the destinations for the data to be stored. Note that SGF should not be set to 1 when using SRC values 41h, 42h, 45h, or 46h. Doing so will cause an error to be generated.<br>**NOTE:** If the IMM bit is set, it is an error for the SGF bit to be set.<br>If CTYPE = 01011b (SEQ STORE), this bit is the Variable Length Flag (VLF).<br>If VLF=0, the LENGTH field indicates the length of the data.<br>If VLF=1, the data length is variable. SEC uses the length in the Variable Sequence Out Length register rather than the value the LENGTH field. However, an error will be generated if the values in the VSOL register and OFFSET field are not a valid combination as indicated in the table Table 7-28. |
| 23<br>IMM | Immediate data.<br>If IMM=0 : Data to be stored is found at the location specified by the SRC field.<br>If IMM=1 : Data to be stored follows as part of the descriptor, using as much space as defined by the LENGTH field and then rounded up to the nearest 4-byte word. For SEQ STORE, the data immediately follows the command; for STORE, the data immediately follows the pointer.<br>**NOTE:** It is an error if the IMM bit is set when the SGF bit is set. However, the destination of a SEQ STORE can have been defined by a Scatter/Gather Table pointed to by the SEQ OUT PTR Command that initiated the Output Sequence. It is an error if IMM =1 and the OFFSET field is non-zero. |
| 22-16<br>SRC | SRC value defines the source (e.g. CONTEXT, ICV, IV) of the data to be stored. See Table 7-28 for a list of supported sources.<br>If IMM = 1 the data to be stored is located as immediate data within the command. Although the SRC field does not specify the source of the data, the SRC field still determines whether the immediate data is treated as message data or control data. When SEC is configured for big endian operation, message data and control data are treated the same. When SEC is configured for little endian operation, control data is byte swapped within words as the immediate data is stored into memory but message data is stored as-is, without byte swapping. SRC = 00h will cause the immediate data to be treated as control data, so when SEC is configured for little-endian operation the data will be byte-swapped within words before it is written to memory. SRC = 7Eh will cause the immediate data to be treated as message data, so the data will be written as-is, without byte swapping. The use of any other SRC value with IMM=1 will cause an error to be generated. |

*Table continues on the next page...*

### Table 7-27. STORE command field descriptions (continued)

| Field | Description |
|---|---|
| 15-8<br>OFFSET | OFFSET defines the start point for reading within the SRC. For example, if the SRC indicates Class 1 context, the offset can be used to indicate that the data should be read from the fourth byte of context rather than from the beginning. The offset into the descriptor buffer is specified in 4-byte words, but in all other cases the offset is specified in bytes. See Table 7-28 for the legal combinations of OFFSET and LENGTH values. |
| 7-0<br>LENGTH | Length of the data. For the descriptor buffer, the length is specified in 4-byte words, but in all other cases the length is specified in bytes. See Table 7-28 for the legal combinations of OFFSET and LENGTH values. |
| *Additional words of STORE command:* | |
| 31-0<br>POINTER | This field is a pointer to the address in memory where the data is to be stored.<br>**NOTE:** This field is not present for any SEQ STORE commands or for STORE commands that store the job descriptor (41h and 45h) or shared descriptor (42h and 46h) from the descriptor buffer into memory which use the pointers previously specified for the job and shared descriptors. (Type 40h requires a pointer for the STORE command.) |
| | |
| 31-0<br>VALUE | If IMM = 1, the value is located here. Enough 4-byte words are used to hold the data of size LENGTH. |

### Table 7-28. STORE command SRC, OFFSET and LENGTH field values

| SRC Value (hex) | Class (binary) | Control Data or Message Data | Legal values in LENGTH/ OFFSET Fields | Tag | Source Internal Register | Comment |
|---|---|---|---|---|---|---|
| 00 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | MODE1 | Class 1 Mode Register | - |
| | 10 | | | MODE2 | Class 2 Mode Register | - |
| | 11 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | DJQCR | DECO Job Queue Control Register | - |
| 01 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | KEYS1 | Class 1 Key Size Register | - |
| | 10 | | | KEYS2 | Class 2 Key Size Register | |
| | 11 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | DDAR | DECO Descriptor Address Register | - |
| 02 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | DATAS1 | Class 1 Data Size Register | - |
| | 10 | | | DATAS2 | Class 2 Data Size Register | |
| | 11 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | DOPSTAT | DECO Operation Status Register | Storing DOPSTAT captures the current "math conditions" (see Table 7-90, TEST CONDITION |

*Table continues on the next page...*

## Table 7-28. STORE command SRC, OFFSET and LENGTH field values (continued)

| SRC Value<br>(hex) | Class<br>(binary) | Control Data or Message Data | Legal values in LENGTH/<br>OFFSET<br>Fields | Tag | Source Internal Register | Comment |
|---|---|---|---|---|---|---|
| | | | | | | field, TEST CONDITION bits when JSL = 0) as well as SEC's current command index. The status is in the left four bytes of this register. The right four bytes contain the number of bytes written to the SEQ OUT PTR address. |
| 03 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | C1ICVS | Class 1 ICV Size Register | - |
| | 10 | | | C2ICVS | Class 2 ICV Size Register | |
| 03 | 11 | Message | 2/0 bytes | DCHKSM | DECO Checksum | An error is generated if length is not 2 or if offset is not 0. |
| 04 | 11 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | DICID | DECO ICID Register | Consists of the following fields: CPL/CICID, OPL/OICID, TZ/SDID, IPL/IICID |
| 06 | 00 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | CCTRL | CHA Control Register | - |
| 07 | 00 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | ICTRL | IRQ Control Register | - |
| 08 | 00 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | CLRW | Clear Written Register | - |
| | 11 | Control | 0-64<br>0-7 bytes | MATH0W | DECO Math Register 0 (Words) | 1, 2 |
| 09 | 00 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | CSTA | CCB Status and Error Register | - |
| | 11 | Control | 0-56<br>0-7 bytes | MATH1W | DECO Math Register 1 (Words) | 1, 2 |
| 0A | 11 | Control | 0-48<br>0-7 bytes | MATH2W | DECO Math Register 2 (Words) | 1, 2 |

*Table continues on the next page...*

### Table 7-28. STORE command SRC, OFFSET and LENGTH field values (continued)

| SRC Value (hex) | Class (binary) | Control Data or Message Data | Legal values in LENGTH/ OFFSET Fields | Tag | Source Internal Register | Comment |
|---|---|---|---|---|---|---|
| 0B | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | AADSZR | AAD Size Register | 1, 2 |
|  | 11 | Control | 0-40<br>0-7 bytes | MATH3W | DECO Math Register 3 (Words) | 1, 2 |
| 0C | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | C1IVSZ | Class 1 IV Size Register | - |
| 0C | 11 | Control | 0-32/<br>0-7 bytes | MATH4W | DECO MATH Register 4 (Words) | 1, 2 |
| 0D | 11 | Control | 0-24/<br>0-7 bytes | MATH5W | DECO MATH Register 5 (Words) |  |
| 0E | 11 | Control | 0-16/<br>0-7 bytes | MATH6W | DECO Math Register 6 (Words) |  |
| 0F | 11 | Control | 0-8/<br>0-7 bytes | MATH7W | DECO Math Register 7 (Words) |  |
| 10 | 11 | Control | 16/0 bytes<br>32/0 bytes<br>48/0 bytes<br>64/0 bytes | GTR | Gather Table Registers |  |
| 10 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | PKASZ | PKHA A Size Register | - |
| 11 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | PKBSZ | PKHA B Size Register | - |
| 12 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | PKNSZ | PKHA N Size Register | - |
| 13 | 01 | Control | 4/0 bytes<br>8/0 bytes<br>4/4 bytes | PKESZ | PKHA E Size Register | - |

*Table continues on the next page...*

### Table 7-28. STORE command SRC, OFFSET and LENGTH field values (continued)

| SRC Value (hex) | Class (binary) | Control Data or Message Data | Legal values in LENGTH/ OFFSET Fields | Tag | Source Internal Register | Comment |
|---|---|---|---|---|---|---|
| 20 | 01 | Message | 0-128/<br>0-128 bytes | CTX1 | Class 1 Context Register | A STORE from the Class 1 Context Register will automatically block until the Class 1 CHA is done. |
| | 10 | | | CTX2 | Class 2 Context Register | A STORE from the Class 2 Context Register will automatically block until the Class 2 CHA is done. |
| 20 | 11 | Control | 16/0 bytes<br>32/0 bytes<br>48/0 bytes<br>64/0 bytes | STR | Scatter Table Registers | |
| 30 | 11 | Control | 0-64<br>0 bytes | MATH0DW | DECO Math Register 0 (Double Word) | 1, 3, 2 |
| 31 | 11 | Control | 0-56<br>0 bytes | MATH1DW | DECO Math Register 1 (Double Word) | |
| 32 | 11 | Control | 0-48<br>0 bytes | MATH2DW | DECO Math Register 2 (Double Word) | |
| 33 | 11 | Control | 0-40 | MATH3DW | DECO Math Register 3 (Double Word) | |
| 34 | 11 | Control | 0-32/<br>0 bytes | MATH4DW | DECO Math Register 4 (Double Word) | 1, 2 |
| 35 | 11 | Control | 0-24/<br>0 bytes | MATH5DW | DECO Math Register 5 (Double Word) | |
| 36 | 11 | Control | 0-16/<br>0 bytes | MATH6DW | DECO Math Register 6 (Double Word) | |
| 37 | 11 | Control | 0-8/<br>0 bytes | MATH7DW | DECO Math Register 7 (Double Word) | |
| 38 | 11 | Control | 0-64<br>0-7 bytes | MATH0B | DECO Math Register 0 (Bytes) | 1, 4 |
| 39 | 11 | Control | 0-56<br>0-7 bytes | MATH1B | DECO Math Register 1 (Bytes) | |
| 3A | 11 | Control | 0-48<br>0-7 bytes | MATH2B | DECO Math Register 2 (Bytes) | |

*Table continues on the next page...*

## Table 7-28. STORE command SRC, OFFSET and LENGTH field values (continued)

| SRC Value<br>(hex) | Class<br>(binary) | Control Data or Message Data | Legal values in LENGTH/<br>OFFSET<br>Fields | Tag | Source Internal Register | Comment |
|---|---|---|---|---|---|---|
| 3B | 11 | Control | 0-40<br>0-7 bytes | MATH3B | DECO Math Register 3 (Bytes) | |
| 3C | 11 | Control | 0-32/<br>0-7 bytes | MATH4B | DECO Math Register 4 (Bytes) | 1, 4 |
| 3D | 11 | Control | 0-24/<br>0-7 bytes | MATH5B | DECO Math Register 5 (Bytes) | |
| 3E | 11 | Control | 0-16/<br>0-7 bytes | MATH6B | DECO Math Register 6 (Bytes) | |
| 3F | 11 | Control | 0-8/<br>0-7 bytes | MATH7B | DECO Math Register 7 (Bytes) | |
| 40 | 01 | Message | 0-128/0-128 bytes | KEY1 | Class 1 Key Register | If the corresponding Key Size register has not been written, the STORE or SEQ STORE command may be used to store the key register into memory. After the key size has been written, the key register can be stored to memory only via a FIFO STORE or SEQ FIFO STORE command. |
| | 10 | Message | 0-128/0-128 bytes | KEY2 | Class 2 Key Register | |
| | 11 | Control | 0-64/<br>offset*<br>words | DESC_BUF | DECO descriptor buffer | See notes below. |
| | | | This SRC value can be used to store any portion of the descriptor buffer into memory.<br>The values in the LENGTH and OFFSET field are specified in 4-byte words.<br>offset* An error is generated if the sum of the LENGTH and OFFSET fields is greater than 64. The OFFSET is used to specify the starting word of the source within the descriptor buffer. Note that the OFFSET is relative to the start of the descriptor buffer. | | | |
| 41 | 11 | Control | 0-64/<br>offset*<br>words | DESC_BUF | DECO descriptor buffer | See notes below. |
| | | | This SRC value is valid only for STORE Commands, not SEQ STORE Commands. This SRC value is used for writing back modifications to Job Descriptors (including Trusted Descriptors). This overwrites the descriptor in memory, using the address from which the descriptor was fetched. Since no Pointer is used, this is a one-word command. Note that this will result in an error if the descriptor came in via QI. If an In-line descriptor, a replacement job descriptor, | | | |

*Table continues on the next page...*

## Table 7-28. STORE command SRC, OFFSET and LENGTH field values (continued)

| SRC Value (hex) | Class (binary) | Control Data or Message Data | Legal values in LENGTH/ OFFSET Fields | Tag | Source Internal Register | Comment |
|---|---|---|---|---|---|---|
| | | | or a Non-local JUMP was executed, an error will be generated for a STORE command with SRC=41h or 45h. Note that SGF should not be set to 1. Doing so will cause an error to be generated.<br>The values in the LENGTH and OFFSET field are specified in 4-byte words.<br>offset* An error is generated if the sum of the LENGTH and OFFSET fields is greater than 64. The OFFSET is used to specify the starting word of the source within the descriptor buffer, and the starting word of the destination within the descriptor in memory. Note that the OFFSET is relative to the start of the Job Descriptor (or Trusted Descriptor) (which will not be the start of the descriptor buffer if there is a Shared Descriptor). See Figure 7-2. | | | |
| 42 | 11 | Control | 0-64/<br>offset*<br>words | DESC_BUF | DECO descriptor buffer | See notes below. |
| | | | This SRC value is valid only for STORE commands, not SEQ STORE commands. This SRC value is used for writing back modifications to shared descriptors. This overwrites the shared descriptor in memory, using the address from which the shared descriptor was fetched. Note that a STORE with SRC=42h or 46h results in an error if there is no shared descriptor. Even if there is a shared descriptor in the original descriptor, an error is generated if there has been a non-local jump to another descriptor or an in-line descriptor is being executed, and that descriptor attempts a STORE with SRC=42h or 46h. Note that SGF should not be set to 1 for SRC values 41h, 42h, 45h or 46h. Doing so will cause an error to be generated.<br>Since no pointer is used, this is a one-word command. The values in the LENGTH and OFFSET field are specified in 4-byte words.<br>To correctly use sharing flows (wait or serial) in SEC, if one job in the flow updates the PDB in memory, all jobs in that flow must update the PDB in memory even if the PDB did not change for that particular packet. If all jobs in the flow update the PDB, SEC will ensure that subsequent jobs do not read the PDB from memory until all updates from prior jobs are complete.<br>offset* An error is generated if the sum of the LENGTH and OFFSET fields is greater than 64. The OFFSET is used to specify the starting word of the source within the descriptor buffer, and the starting word of the destination within the descriptor in memory. Note that the OFFSET is relative to the start of the shared descriptor in both the descriptor buffer and in memory. See Figure 7-2. | | | |
| 45 | 11 | Control | 0-64/<br>offset*<br>words | DESC_BUF | DECO Descriptor Buffer (Write Efficient) | This SRC value is used to write modifications to job descriptors back to memory using write-efficient bus transactions. See notes below. |
| | | | 45h is the same as 41h, and 46h is the same as 42h, except that these SRC values cause write-efficient bus transactions (see AXI master (DMA) interface). Since these bus transactions may write more of the descriptor buffer back to memory than is specified by OFFSET and LENGTH, these SRC values should be | | | |

*Table continues on the next page...*

Table 7-28. STORE command SRC, OFFSET and LENGTH field values (continued)

| SRC Value (hex) | Class (binary) | Control Data or Message Data | Legal values in LENGTH/ OFFSET Fields | Tag | Source Internal Register | Comment |
|---|---|---|---|---|---|---|
| | | | used only if it is permitted to write back to memory all of the descriptor (See AXI master (DMA) interface). Note that SGF should not be set to 1. Doing so will cause an error to be generated.<br>If write-safe operations are not implemented or not enabled in this version of SEC, write-efficient operations are also not available, and SRC value 45h operates exactly as 41h, and SRC value 46h operates exactly as 42h. | | | |
| 46 | 11 | Control | 1-64/<br>offset*<br>words | DESC_BUF | DECO Descriptor Buffer (Write Efficient) | This SRC value is used to write modifications to shared descriptors back to memory like SRC=42h, but using write-efficient bus transactions. See notes above for SRC=45h. |
| All combinations of SRC and CLASS that do not appear in Table 7-28 are reserved. | | | | | | |

1. Because the math registers are in contiguous addresses, it is possible to store more than one math register simultaneously.
2. When this source is used, the data stored from the Math register will be treated as words.
3. When this source is used, the data stored from the Math register will be treated as double words. Offset must be 0. Word swapping will be handled the same as address pointers.
4. When this source is used, the data stored from the Math register will be treated as bytes.

## 7.14 FIFO STORE command

**NOTE**

> In the following discussion, FIFO STORE command refers to both the SEQ and non-SEQ forms of the command.

FIFO STORE commands are used to move data from the output data FIFO to external memory by means of the DMA. Because the only source is the output data FIFO, this command does not include a SRC field. The SEQ FIFO STORE command is identical to the FIFO STORE command except that no address is specified and the SGT bit is replaced by the VLF bit. See SEQ vs non-SEQ commands.

Note that data output by means of the output data FIFO is considered message data. Therefore, it is byte-swapped, half-word-swapped, full-word-swapped, and double-word swapped in accordance with the message data swapping configuration. The swapping can be configured independently for each job ring and for the Queue Manager Interface (see the Job Ring Configuration Register (JRCFGR) and the Queue Interface Control Register (QICTL)).

The following types of data can be output from the output data FIFO.

- PKHA registers, other than the E-Memory.
- PKHA E Memory. This data is encrypted as a Black Key prior to being written to memory.
- Class 1 and Class 2 keys.
- RNG data, which can be left in the output data FIFO or stored away.
- Regular data, which is pulled and written as it appears in the output data FIFO. Note that bit length data stores are not available.
- Data in the input sequence or in the input data FIFO

Note that even though this command is not a store check point, it does not start if a prior STORE or FIFO STORE of any type has yet to be scheduled. It is a done checkpoint if asked to encrypt a key because it has to wait until both class CHAs are done. The FIFO STORE command will block if the internal CCB DMA is not available when storing C1 or C2 keys.

The FIFO LOAD command supports bit length data, (see Bit length data), but the FIFO STORE command does not support bit lengths.

It is occasionally necessary to skip over portions of the output buffer (meaning to advance the output sequence pointer without actually writing data) before writing more output. For instance, in certain networking protocols, portions of the output stream may depend on out-of-order portions of the input stream. This processing can be done in two or more passes through the input and output sequences by:

1. Skipping portions of the input and output data in one pass
2. Restoring the sequences for the next pass by means of the RTO bit in the SEQ IN PTR command and the REW field in the SEQ OUT PTR command
3. Skipping over the portions of the output data that were written in the previous pass

To achieve skipping with SEQ FIFO STORE, use output data type 3Fh. Note that scatter tables may be read while skipping if the sequence was defined with the SGF bit set in the SEQ OUT PTR command. However, no data will be written while skipping.

**Table 7-29. FIFO STORE command format**

| 31–27 | 26–25 | 24 | 23 | 22 | 21–16 |
|---|---|---|---|---|---|
| CTYPE = 01100 or 01101 | AUX | SGF or VLF | CONT | EXT | OUTPUT DATA TYPE |
| **15–0** | | | | | |
| LENGTH | | | | | |
| *Additional words of FIFO STORE command:* | | | | | |
| Pointer ( one or two words, see Address pointers) | | | | | |
| | | | | | |
| EXT LENGTH (Present if EXT = 1) (one word) | | | | | |

### Table 7-30. FIFO STORE command field descriptions

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>If CTYPE=01100b : FIFO STORE command<br>If CTYPE=01101b : SEQ FIFO STORE command |
| 26-25<br>AUX | Auxiliary control bits. Used only for certain output data type codes. Set AUX = 00 for all other output data type codes. See Table 7-31. |
| 24<br>SGF or VLF | Scatter/Gather table Flag (SGF) or Variable Length Flag (VLF). Meaning depends on CTYPE.<br>If CTYPE = 01100 (FIFO STORE), this bit is the Scatter/Gather table Flag (SGF).<br>If SGT=0 : The pointer points to actual data.<br>If SGT=1 : The pointer points to a scatter/gather table.<br>If CTYPE = 01101 (SEQ FIFO STORE), this bit is the Variable Length Flag (VLF).<br>If VLF=0 : The number of bytes of data to be stored is specified in the LENGTH (if EXT=0) or EXT LENGTH (if EXT=1) field.<br>If VLF=1 : The data length is variable. The number of bytes of data to be stored is specified in the VSOL register. The LENGTH field is ignored.<br>**NOTE:** It is legal to set VLF=1 when storing a key<br>**NOTE:** It is illegal to set VLF=1 when EXT=1. |
| 23<br>CONT | Continue<br>If CONT=0 : If the FIFO STORE pulls data from the output FIFO and finishes at an alignment other than at the end of a dword, the remainder of the last dword is popped and discarded. If the read from the output FIFO ends with the last byte of the dword, that dword is always popped.<br>If CONT=1 : This is not the last FIFO STORE command for this data. The final dword that contributed data is not popped if the data did not end at an 8-byte boundary. This is used to prevent data loss when a store leaves off in the middle of a dword.<br>**NOTE:** If this bit is set when there is no remaining data, subsequent operations may not work as expected. |
| 22<br>EXT | Use Extended Length<br>If EXT=0 : Output data length is solely determined by the 16-bit LENGTH field in the first word of the command.<br>If EXT=1 : Output data length is determined by the 32-bit EXT LENGTH field.<br>**NOTE:** It is illegal to set VLF=1 when EXT=1. |
| 21-16<br>Output Data Type | This field identifies the type of data that the output data FIFO stores. See Table 7-31 for a list of the supported types. |
| 15-0<br>LENGTH | The length of the data to be stored.<br>If EXT=0 : The LENGTH field specifies the number of bytes to store.<br>If EXT=1 : The EXT FIELD specifies the number of bytes to store. The LENGTH field is ignored. |
| *Additional words of the FIFO STORE command:* | |
| POINTER | Address pointer where to store the data in memory.<br>**NOTE:** This field is not present for SEQ FIFO STORE commands, nor is it present for FIFO STORE commands if the data type is for RNG and the data is to be left in the output data FIFO. |
| | |
| EXT LENGTH | Extended length field.<br>If EXT=0 : This field is not present. The LENGTH field specifies the number of bytes of data to be stored. |

### Table 7-30. FIFO STORE command field descriptions

| Field | Description |
|---|---|
| | If EXT=1 : The EXT LENGTH field specifies the number of bytes of data to be stored. The LENGTH field in the first word of the command is ignored. |

Table 7-31 lists the various built-in FIFO STORE output data types.

### Table 7-31. FIFO STORE output data type field

| Bits 21-16<br>(hex) | Meaning | Comment |
|---|---|---|
| 00 | PKHA A0 | **NOTE:** The appropriate size register is automatically written. A FIFO STORE from a PKHA register should never be attempted with size greater than the PKHA register size. |
| 01 | PKHA A1 | |
| 02 | PKHA A2 | |
| 03 | PKHA A3 | |
| 04 | PKHA B0 | |
| 05 | PKHA B1 | |
| 06 | PKHA B2 | |
| 07 | PKHA B3 | |
| 08 | PKHA N | |
| 0C | PKHA A | |
| 0D | PKHA B | |
| 12 | PKHA E, AES-CCM encrypted using the job descriptor key encryption key | |
| 13 | PKHA E, AES-CCM encrypted using the trusted descriptor key encryption key. | Available only to trusted descriptors. |
| 14 | Key Register AES-CCM encrypted using the job descriptor key encryption key. | The AUX field determines the source register for the FIFO STORE.<br>• AUX = 01 selects the Class 1 Key Register to be stored.<br>• AUX = 10 selects the Class 2 Key Register to be stored.<br>AUX values 00 and 11 are illegal. |
| 15 | Key register, AES-CCM encrypted using the trusted descriptor key encryption key. | Available only to trusted descriptors. The AUX field determines the source register for the FIFO STORE.<br>• AUX = 01 selects the Class 1 Key Register to be stored.<br>• AUX = 10 selects the Class 2 Key Register to be stored.<br>AUX values 00 and 11 are illegal. |
| 16 | Class 2 Key Register MDHA Split Key, AES-CCM encrypted using the job descriptor key encryption key. | For performance and security, use of an MDHA split key is highly recommended. Details about split keys can be found in Using the MDHA Key Register with IPAD/OPAD "split keys". The length of such a split key is twice the length of the chosen MDHA algorithm's running digest (see MDHA |

*Table continues on the next page...*

## Table 7-31. FIFO STORE output data type field (continued)

| Bits 21-16<br>(hex) | Meaning | Comment |
|---|---|---|
| | | use of the Context Register). If the Class 2 Key register was loaded with a split key using a KEY command with PTS=1, or if a key was loaded into the C2 Key register and then the MDHA was run in INIT mode to create a split key, the C2 Key register will be stored in plaintext form. |
| 17 | Class 2 Key Register MDHA Split Key, AES-CCM encrypted using the trusted descriptor key encryption key. | Available only to trusted descriptors. The comments for type 16h apply here as well. |
| 22 | PKHA E, AES-ECB encrypted using the job descriptor key encryption key | - |
| 23 | PKHA E, AES-ECB encrypted using the trusted descriptor key encryption key. | Available only to trusted descriptors. |
| 24 | Key Register, AES-ECB encrypted using the job descriptor key encryption key. | The AUX field determines the source register for the FIFO STORE.<br>• AUX = 01 selects the Class 1 key register to be stored.<br>• AUX = 10 selects the Class 2 key register to be stored.<br>AUX values 00 and 11 are illegal. |
| 25 | Key Register, AES-ECB encrypted using the trusted descriptor key encryption key. | Available only to trusted descriptors. The AUX field determines the source register for the FIFO STORE.<br>• AUX = 01 selects the Class 1 Key Register to be stored.<br>• AUX = 10 selects the Class 2 Key Register to be stored.<br>AUX values 00 and 11 are illegal. |
| 26 | Class 2 Key Register MDHA split key, AES-ECB encrypted using the job descriptor key encryption key. | For performance and security, use of an MDHA split key is highly recommended. Details on this split key can be found in Using the MDHA Key Register with IPAD/OPAD "split keys". The length of such a split key is twice the length of the chosen MDHA algorithm's running digest (see MDHA use of the Context Register). If the Class 2 Key register was loaded with a split key using a KEY command with PTS=1, or if a key was loaded into the C2 Key register and then the MDHA was run in INIT mode to create a split key, the C2 Key register will be stored in plaintext form. |
| 27 | Class 2 Key Register MDHA split key, AES-ECB encrypted using the trusted descriptor key encryption key. | Available only to trusted descriptors. The comments for type 26h apply here as well. |
| 30 | Message Data | If a type 31h has been used, type 30h disables automatic checksum calculation. Any current checksum value will remain. A 0-length command may be used to effect this change. |
| 31 | Message Data | The first time this type is used, the running check sum is cleared. The use of this type enables the check sum hardware. A 0-length command may be used to effect this change. |

*Table continues on the next page...*

## Table 7-31. FIFO STORE output data type field (continued)

| Bits 21-16<br>(hex) | Meaning | Comment |
|---|---|---|
| 34 | Store the specified amount of data to be obtained from RNG to memory. | **NOTE:** The Class 1 Data Size Register is automatically written and extended lengths are illegal.<br>The different types of random data that can be generated are:<br>• Random data with no restriction<br>• Nonzero Random data<br>• Odd Parity Random data.<br>The Mode Register controls the type of random data. Note that the RNG must be selected by writing the Mode register. |
| 35 | Obtain the specified amount of data from RNG and leave it in the output data FIFO. | In addition to the comments for type 34h, there is no pointer and it is illegal to use type 35h with SEQ FIFO STORE. |
| 3E | Meta Data | For this output data type, CONT and EXT must both be 0. Either bit set to a 1 generates an error. This type can be used only with SEQ FIFO STORE; an error is generated if this output data type is used with FIFO STORE. Length can be specified in the command (VLF = 0) or in the Variable Sequence Out Length register (VLF = 1). If VLF = 1, the length must fit in the lower 16 bits of the VSIL register or an error is generated.<br>The AUX bits control the behavior of the SEQ FIFO STORE command as follows:<br>00 Use the DECO alignment block to move the specified number of bytes from the input FIFO to the output FIFO and store them to the output frame. This variant of the command is used when handling meta data that has already been read. An example of this would be for a shared descriptor where the RIF bit is set.<br>01 The same as 00, except that the VSIL is decremented by the specified length. This form should be used when the RIF bit is set and the VSIL contains the input frame length of the packet. If the VSIL were not decremented in this case, the descriptor would have to subtract the meta data length from the VSIL register before running a protocol.<br>10 Load the specified number of bytes from the input frame, as defined by a prior SEQ IN PTR command, to the input FIFO and decrement the Sequence In Length by this number of bytes. Move these bytes to the output FIFO by means of the DECO alignment block and store them to the output frame. This variant of the command is used when handling meta data that precedes the packet data.<br>11 The same as 10, except that the Sequence In Length is not decremented. This form should be used when moving meta data that follows the packet data. Normally the length of trailing meta data has to be subtracted from the input frame length prior to running a protocol so that the protocol knows how long the packet is. When using the AUX = 11 |

Table continues on the next page...

**Table 7-31. FIFO STORE output data type field (continued)**

| Bits 21-16<br>(hex) | Meaning | Comment |
|---|---|---|
| | | variant the descriptor does not have to add the meta data length back to the Sequence In Length before executing the SEQ FIFO STORE Meta Data command. |
| 3F | Skip | Skip over the specified length in memory without using bus cycles. Permitted to be used only by SEQ FIFO STORE. |
| **NOTE:** AUX must be set to 00 except when otherwise specified above. All combinations of output data type and AUX values not specified are reserved. | | |

## 7.15 MOVE, MOVEB, MOVEDW, and MOVE_LEN commands

**NOTE**

In this section "Move Command" is used to refer to the MOVE, MOVEB, MOVEDW, and MOVE_LEN forms of the command.

The MOVE command is used to copy data between two resources internal to a DECO/CCB. This allows data to be put in the proper registers without having to store data to external memory and then load it.

The OFFSET field is used to define an offset into either the source or destination, depending on the values in the SRC, DST, and AUX fields (see table Table 7-35). The MOVE command has a limited number of sources and destinations as indicated in the SRC and DST field descriptions below.

**NOTE**

MOVE cautions and restrictions:

- Keys can't be copied from a key register by means of a MOVE command if the corresponding key size register has been written.
- Observe the cautions noted in the "RJD" field of SEQ IN PTR command if using a MOVE command in a Replacement Job Descriptor.
- Moves may be checkpoints. For example, a move from the Class 2 Context Register to the Input Data FIFO for the Class 1 CHA is a Load Checkpoint and is a Done Checkpoint for the Class 2 CHA.

When moving data to or from the Descriptor Buffer or a MATH register, the MOVEB command is identical to the MOVE command if byte swapping is not enabled (the norm for Big-Endian configuration), but, if byte swapping is enabled, MOVEB treats data as 32-bit words in those cases that MOVE treats data as bytes, and MOVEB treats data as bytes when MOVE treats data as words. The MOVEDW command and the MOVE_LEN command for dwords always treat data as double words (i.e. 64 bits) and perform word swapping when SEC is configured to swap dwords. The MOVEDW command and the MOVE_LEN command for dwords never do byte swapping.

NOTE: For MOVEDW or MOVE_LEN for dwords, with one exception, the offset must be a multiple of dwords (8 bytes). The one exception is when the offset is into the descriptor buffer, in which case the offset is allowed to be a multiple of words (4 bytes). If the source is the output FIFO dword moves will always result in an error if the OFIFO offset is not zero.

In the MOVE, MOVEB, or MOVEDW command, the LENGTH field specifies the amount of data to be moved. The MOVE_LEN command is identical to the MOVE, MOVEB, and MOVEDW commands except that the length of the data being moved is specified in a MATH register, rather than specified as a constant in the LENGTH field. In the MOVE_LEN command, the MRSEL (Math Register Select) field, the TYPE field, and a reserved field replace the MOVE command's LENGTH field.

The AUX field is used to select among a number of different options, depending on the values in the SRC and DST fields (see the table Table 7-35 below).

The MOVE command will block if the CCB DMA is busy. Other conditions where the MOVE command will block include:

- The SRC is context and the corresponding class CHA is not done or there is a data in flight to either context register.
- The SRC is the output FIFO but a request for the external DMA to pull data from the output FIFO is pending.
- The DST is a context register and there is data in flight to either context register.
- The DST is the input data FIFO but there is data in flight to the input data FIFO.
- The DST is the input data FIFO for either of the C1 or C2 alignment blocks and an NFIFO entry is to be written and there is a context load pending.
- The DST is the C1 Key Register and a context load is pending and the write is to the extended key range.

**NOTE**

For this device, the default is that byte swapping is disabled. This means that MOVE, MOVEB, and MOVE_LEN (when the TYPE is 00 or 10) will not swap bytes according to the table below.

**NOTE**

For this device, the default is that word swapping is disabled. This means that MOVEDW and MOVE_LEN (when the TYPE is 11) will not swap words.

**NOTE**

> This device does not prevent the byte and word swapping defaults from being overridden. However, care should be used when changing this behavior.

The output FIFO provides data through two access points. The first is for the external DMA and the second is shared by three consumers: the CCB DMA, DECO access via the MATH command, and the NFIFO. The two access points have separate indices into the output FIFO so each can track separately allowing consumption of data at different rates. However, the only time these indices track separately is when the NFIFO is consuming data from the output FIFO. Therefore, when using a move command to extract data from the output FIFO, it is critical that the descriptor writer know whether the indices are tracking together and, if not, which index needs to be used to obtain the desired data. Note that this is an extremely unusual circumstance which most descriptor writers will seldom, if ever, encounter.

In prior versions of SEC, different entities handled the move from the output FIFO depending on alignment. If the OFFSET specified was a multiple of words and the current ofifo offset was 0, then the CCB DMA handled the move. Otherwise, the external DMA handled the move. The DMA used determined the index that was used to access the data in the output FIFO. This led to significant confusion about which data was being read. In this version of SEC, the CCB DMA handles all moves from the output FIFO, eliminating the confusion. However, it is possible for the descriptor to manipulate the index.

**NOTE**

> It is possible to make the earlier behavior forward compatible by careful descriptor construction.

- To ensure that the move reads from the index where the external DMA left off, perform a LOAD IMM to the DECO Control Register to reset the CHA pointer in the output FIFO. This has the effect of setting the shared index to the same value as the index used by the external DMA. While this will lose the current index of the CHA pointer, the move will get the expected data. (Remember that this is only necessary when the two indices are different. If the amount snooped and the amount read by the external DMA are the same, the indices will be the same.)
- To ensure that the move reads from the index where the NFIFO left off, ensure that the ofifo offset is 0 and that the OFFSET in the move command is 0.

The ofifo offset is used in two ways. First, it is used by DECO to tell the external DMA where in the 8-byte interface to the output FIFO to start reading. Second, it is used by the CCB DMA to know where in the 8-byte interface to the output FIFO to start reading. However, the two DMAs use different indices to access the output FIFO so that the offset could be referencing different dwords. While those indices are usually synchronized, they can become different when the NFIFO has pulled data from the output FIFO. It is therefore critical that the descriptor writer keep track of where each index is when moves from the output FIFO follow snooping.

**Table 7-32. MOVE, MOVEB, MOVEDW, and MOVE_LEN command format**

<table>
<tr><th></th><th>31–27</th><th>26–25</th><th>24</th><th colspan="2">23–20</th><th colspan="2">19–16</th></tr>
<tr><td></td><td>CTYPE = 01111, 00110, 00111 or 01110</td><td>AUX</td><td>WC</td><td colspan="2">SRC</td><td colspan="2">DST</td></tr>
<tr><td></td><td colspan="3"><b>15–8</b></td><td colspan="4"><b>7-0</b></td></tr>
<tr><td><i>Fields as they appear in the MOVE, MOVEB, or MOVEDW command:</i></td><td colspan="3">OFFSET</td><td colspan="4">LENGTH</td></tr>
<tr><td></td><td colspan="3"><b>15–8</b></td><td><b>7-6</b></td><td colspan="2"><b>5-3</b></td><td><b>2-0</b></td></tr>
<tr><td><i>Fields as they appear in the MOVE_LEN command:</i></td><td colspan="3">OFFSET</td><td>TYPE</td><td colspan="2">Reserved</td><td>MRSEL</td></tr>
</table>

**Table 7-33. MOVE command field descriptions**

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command Type<br>01111 - MOVE. Performs an internal move between two internal DECO/CCB locations. The length of the data is specified by the value in the LENGTH field. If byte swapping is enabled, MOVE swaps bytes within words in certain cases (see table Table 7-34).<br>00111 - MOVEB. When byte swapping is not enabled, the legal MOVEB moves are identical to the corresponding MOVE moves. However, when byte swapping is enabled, the MOVEB moves byte swap within words when the corresponding MOVE moves do not swap and vice versa (see table Table 7-34).<br>00110 - MOVEDW. Move Double Words. Performs an internal move between two internal DECO/CCB locations. If word swapping is not enabled, the legal MOVEDW moves are identical to the corresponding MOVE moves when byte swapping is disabled. If word swapping is enabled for the descriptor, the MOVEDW command swaps the order of the two words in a double word. No byte swapping is done.<br>01110 - MOVE_LEN. Performs an internal move between two internal DECO/CCB locations. The length of the data is specified by the value in the MATH register selected by the MRSEL field. Byte or word swapping may be done based on the endianess settings and the value in the TYPE field. |
| 26-25<br>AUX | AUX bits are used for some SRC and DST combinations to specify additional options. See table Table 7-35 below. |
| 24<br>WC | Wait for Completion<br>0 - Do not Wait for Completion<br>1 - Wait for Completion. Causes the MOVE command to stall until the move operation completes. This is necessary when the data to be moved must be in place before a subsequent command executes. While it is sometimes possible to know a priori that the MOVE command will complete prior to reaching the subsequent command in question, such completion can not always be guaranteed. |

*Table continues on the next page...*

## Table 7-33. MOVE command field descriptions (continued)

| Field | Description |
|---|---|
| 23-20<br>SRC | Source. This specifies the internal source of data that will be moved. See table Table 7-37 below for additional information. Note that not all combinations of source and destination are allowed. The tables Table 7-35 and Table 7-36 indicate which source and destination combinations are permitted. |
| 19-16<br>DST | Destination. This specifies the internal destination of the data that will be moved. See table Table 7-38 below for additional information. Note that not all combinations of source and destination are allowed. The tables Table 7-35 and Table 7-36 indicate which source and destination combinations are permitted. |
| 15-8<br>OFFSET | Offset. (in bytes)<br>The interpretation of the OFFSET field depends on the source and destination, as shown in table Table 7-36. The OFFSET is limited to 128 bytes except when the Descriptor Buffer is the source or destination, in which case the OFFSET may be as large as 255 bytes. For MOVEDW and MOVE_LEN for dwords, the OFFSET must be a multiple of 8 bytes unless the OFFSET is into the Descriptor Buffer, in which case the OFFSET must be a multiple of 4 bytes. |
| 7-0<br>LENGTH | Length for internal move. (in bytes, 128 max) This field appears only in the MOVE, MOVEB, or MOVEDW forms of the command. In the MOVE_LEN form of the command this field is replaced by reserved bits and the MRSEL field and TYPE field, as shown below. |
| *Note that in the MOVE_LEN form of the command the LENGTH field is replaced by the following three fields:* | |
| 7-6 TYPE | Type of the data items that are to be moved.<br>00 - Data is treated the same as in the MOVE command<br>01 - Data is treated as dwords the same way as in the MOVEDW command<br>10 - Data is treated as bytes the same way as in the MOVEB command<br>11 - Reserved and reported as an error |
| 5-3 | These bits are reserved in the MOVE_LEN form of the command. These bits, the TYPE field and the MRSEL field below replace the LENGTH field that appears in the MOVE form of the command. |
| 2-0<br>MRSEL | MATH Register Select<br>This field is used only in the MOVE_LEN form of the command. The MRSEL field, TYPE field, and the reserved bits above replace the LENGTH field that appears in the MOVE, MOVEB, and MOVEDW forms of the command. The length (in bytes) of the data to be moved is specified in the MATH Register selected by the MRSEL field. If the move is from the input FIFO or any of the alignment blocks to the output FIFO, bits 15:0 of the MATH Register are used for the length; otherwise, only bits 7:0 are used. Other bits are simply ignored.<br>000 - Math Register 0<br>001 - Math Register 1<br>010 - Math Register 2<br>011 - Math Register 3<br>100 - Math Register 4<br>101 - Math Register 5<br>110 - Math Register 6<br>111 - Math Register 7 |

## Table 7-34. Byte swapping in move commands

| |
|---|
| When byte swapping is enabled, this table indicates when bytes within a word are swapped. Legend: |
| M refers to the MOVE command and the MOVE_LEN command when TYPE=00 |
| B refers to the MOVEB command and the MOVE_LEN command when TYPE=10 |

*Table continues on the next page...*

## Table 7-34. Byte swapping in move commands (continued)

<table>
<tr><td colspan="11">Swap: indicates the move will swap bytes within words<br>Not: indicates the move will not swap bytes within words<br>Err: indicates the move command will generate an error</td></tr>
<tr><td>DST →<br>SRC<br>↓</td><td>0h: C1 Context<br>1h: C2 Context</td><td>2h: Output<br>Data FIFO</td><td>3h: Descriptor<br>Buffer</td><td>4h: Math 0<br>5h: Math 1<br>6h: Math 2<br>7h: Math 3</td><td>8h: Class 1 Input<br>Data FIFO</td><td>9h: Class 2 Input<br>Data FIFO</td><td>Ah: Input<br>Data FIFO<br>(no NFIFO entries)</td><td>Ch:<br>PKHA A RAM<br>(always flushed)</td><td>Dh: C1 Key<br>Eh: C2 Key</td><td>Fh: Aux Data FIFO</td></tr>
<tr><td>0h: C1 Context<br>1h: C2 Context</td><td colspan="2">M:Not<br>B:Err</td><td rowspan="2">M:Swap<br>B:Not</td><td rowspan="3">M:Not<br>B:Swap</td><td colspan="6" rowspan="2">M:Not<br>B:Err</td></tr>
<tr><td>2h: Output FIFO</td><td>M:Not<br>B:Err</td><td>M:Err<br>B:Err</td></tr>
<tr><td>3h: Descr Buffer</td><td colspan="2">M:Swap<br>B:Not</td><td>M:Err<br>B:Err</td><td colspan="3">M:Swap<br>B:Not</td><td colspan="2">M:Not<br>B:Swap</td><td>M:Swap<br>B:Not</td></tr>
<tr><td>4h: Math Reg 0<br>5h: Math Reg 1<br>6h: Math Reg 2<br>7h: Math Reg 3</td><td colspan="10">M:Not<br>B:Swap</td></tr>
<tr><td>8h: DECO Alignment Block (flushed)</td><td colspan="2" rowspan="4">M:Not<br>B:Err</td><td rowspan="4">M:Swap<br>B:Not</td><td rowspan="4">M:Not<br>B:Swap</td><td colspan="4" rowspan="3">M:Err<br>B:Err</td><td rowspan="3">M:Not<br>B:Err</td><td rowspan="3">M:Err<br>B:Err</td></tr>
<tr><td>9h: Class 1 or Class 2 Alignment Block</td></tr>
<tr><td>Ah: DECO, Class 1 or Class 2 Alignment Block</td></tr>
<tr><td>Dh: Class 1 Key<br>Eh: Class 2 Key</td><td colspan="6">M:Not<br>B:Err</td></tr>
</table>

## Table 7-35. Usage of the AUX field in move commands

| DST →<br>SRC<br>↓ | 0h: C1 Context<br>1h: C2 Context | 2h: Output<br>Data FIFO | 3h: Descriptor<br>Buffer | 4h: Math 0<br>5h: Math 1<br>6h: Math 2<br>7h: Math 3 | 8h: Class 1 Input<br>Data FIFO | 9h: Class 2 Input<br>Data FIFO | Ah: Input<br>Data FIFO<br>(no NFIFO entries) | Ch:<br>PKHA A RAM<br>(automatically flushed) | Dh: C1 Key<br>Eh: C2 Key | Fh: Aux Data FIFO |
|---|---|---|---|---|---|---|---|---|---|---|
| 0h: C1 Context<br>1h: C2 Context | | | AUX selects offset into Context Register | AUX selects offset into Math Register | $\text{AUX}_{MS}$ : Flush<br>$\text{AUX}_{LS}$ : Last | $\text{AUX}_{LS}$: Last | | | $\text{AUX}_{LS}$=0 : OFFSET field into Context Reg | |

*Table continues on the next page...*

## Table 7-35. Usage of the AUX field in move commands (continued)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 00: 0 bytes<br>01: 16 bytes<br>10: 32 bytes<br>11: 48 bytes | 00: 0 bytes<br>01: 4 bytes<br>10: 6 bytes<br>11: 7 bytes | | | | | AUX$_{LS}$=1 : OFFSET into Key Reg | |
| 2h: Output FIFO | | move not allowed | | | | | | | | |
| 3h: Descr Buffer | AUX selects offset into Context Register<br>00: 0 bytes<br>01: 16 bytes<br>10: 32 bytes<br>11: 48 bytes | | move not allowed | AUX selects offset into Math Register<br>00: 0 bytes<br>01: 4 bytes<br>10: 6 bytes<br>11: 7 bytes | | | | | | |
| 4h: Math Reg 0<br>5h: Math Reg 1<br>6h: Math Reg 2<br>7h: Math Reg 3 | AUX selects offset into Math Register<br>00: 0 bytes><br>01: 4 bytes<br>10: 6 bytes<br>11: 7 bytes | | AUX selects offset into Math Register<br>00: 0 bytes<br>01: 4 bytes<br>10: 6 bytes<br>11: 7 bytes | AUX selects offset into the source Math Register<br>00: 0 bytes<br>01: 4 bytes<br>10: 6 bytes<br>11: 7 bytes | | | | | | |
| 8h: DECO Alignment Block (automatically flushed) | | | | | move not allowed | | | | | move not allowed |
| 9h: Class 1 or Class 2 Alignment Block | AUX$_{MS}$ : Flush<br>AUX$_{LS}$ =0 : Source is Class 2 Alignment Block<br>AUX$_{LS}$ =1 : Source is Class 1 Alignment Block | | | | | | | | AUX$_{MS}$ : Flush<br>AUX$_{LS}$ =0 : Source is C2 Align Block<br>AUX$_{LS}$ =1 : Source is C1 Align Block | |

*Table continues on the next page...*

## Table 7-35. Usage of the AUX field in move commands (continued)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Ah: DECO, Class 1 or Class 2 Alignment Block | AUX field selects Alignment Block<br>00: DECO Alignment Block<br>01: Class 1 Alignment Block<br>10: Class 2 Alignment Block<br>11: error | | | | | | | | AUX field selects Alignment Block<br>00: DECO Alignment Block<br>01: C1 AB<br>10: C2 AB<br>11: error | |
| Dh: Class 1 Key<br>Eh: Class 2 Key | Determines SRC/DST offset | | | | $\text{AUX}_{MS}$ : Flush<br>$\text{AUX}_{LS}$ : Last | $\text{AUX}_{LS}$: Last | | | Determines SRC/DST offset | |

## Table 7-36. Usage of the OFFSET field in move commands

| DST →<br>SRC<br>↓ | 0h: C1 Context<br>1h: C2 Context | 2h: Output<br>Data FIFO | 3h: Descriptor<br>Buffer | 4h: Math 0<br>5h: Math 1<br>6h: Math 2<br>7h: Math 3 | 8h: Class 1 Input<br>Data FIFO | 9h: Class 2 Input<br>Data FIFO | Ah: Input<br>Data FIFO<br>(no NFIFO entries) | Ch:<br>PKHA A RAM<br>(always flushed ) | Dh: C1 Key<br>Eh: C2 Key | Fh: Aux Data FIFO |
|---|---|---|---|---|---|---|---|---|---|---|
| 0h: C1 Context<br>1h: C2 Context | OFFSET field is used for SRC | OFFSET field is used for SRC | OFFSET field is used for Descriptor Buffer (offset into Context Reg is determined by AUX field) | OFFSET field is used for SRC | | | | | $\text{AUX}_{LS}$=0:<br>OFFSET field is used for Context Reg<br>$\text{AUX}_{LS}$=1:<br>OFFSET field is used for Key Reg | OFFSET field is used for SRC |
| 2h: Output FIFO | OFFSET field is used for DST | move not allowed | OFFSET field is used for DST | | Error generated if OFFSET≠0 | | | | OFFSET field is used for DST | Error generated if OFFSET≠0 |
| 3h: Descr Buffer | OFFSET field is used for SRC<br>(offset into Context Reg is determined by AUX field) | OFFSET field is used for SRC | move not allowed | OFFSET field is used for SRC | | | | | OFFSET field is used for SRC | OFFSET field is used for SRC |

*Table continues on the next page...*

## Table 7-36. Usage of the OFFSET field in move commands (continued)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4h: Math Reg 0<br>5h: Math Reg 1<br>6h: Math Reg 2<br>7h: Math Reg 3 | OFFSET field is used for DST | OFFSET field is used for SRC | OFFSET field is used for DST | | OFFSET field is used for SRC | OFFSET field is used for DST | OFFSET field is used for SRC |
| 8h: DECO Align Block (flushed) | OFFSET field is used for DST | OFFSET is ignored in MOVE_LEN<br>In MOVE, OFFSET field is prepended to the LENGTH field to form a 16-bit length | OFFSET field is used for DST | | move not allowed | OFFSET field is used for DST | move not allowed |
| 9h: Class 1 or Class 2 Alignment Block | | | | | | | |
| Ah: DECO, Class 1 or Class 2 Alignment Block | | | | | | | |
| Dh: Class 1 Key<br>Eh: Class 2 Key | AUX$_{LS}$ =0 : SRC; else DST | SRC | DST | SRC | SRC | AUX$_{LS}$ =0 : SRC; else DST | SRC |

## Table 7-37. Move sources

| Value | Move Source | Notes |
|---|---|---|
| 0h | Class 1 Context Reg | — |
| 1h | Class 2 Context Reg | — |
| 2h | Output Data FIFO | — |
| 3h | Descriptor Buffer | — |
| 4h | Math Register 0 | A MOVE command that reads past the end of MATH Register 3 will continue reading into Math Registers 4-7<br>**NOTE:** Math Registers 4-7 are not available as sources for the MOVE commands. |
| 5h | Math Register 1 | |
| 6h | Math Register 2 | |
| 7h | Math Register 3 | |
| 8h | DECO Alignment Block (always Flushed) | Input to the DECO Alignment Block is specified by an NFIFO entry which is automatically generated if Automatic NFIFO entries are enabled. |
| 9h | Class 1 or Class 2 Alignment Block | The choice between the Class 1 and Class 2 Alignment Blocks is determined by the least-significant bit of the AUX field:<br>• $A_{LS}$ = 0 selects C2 Alignment Block<br>• $A_{LS}$ = 1 selects C1 Alignment Block<br>Input to the Class 1 or Class 2 Alignment Block is specified by an NFIFO entry which is automatically generated if Automatic NFIFO entries are enabled.<br>The most-significant bit of the AUX field must be set (causing a FLUSH) only if the destination is the Output Data FIFO. |
| Ah | DECO, Class 1 or Class 2 Alignment Block, as specified via the AUX field. | no NFIFO entry generated;<br>AUX = 00b: use DECO Alignment Block<br>AUX = 01b: use Class 1 Alignment Block |

*Table continues on the next page...*

Table 7-37. Move sources (continued)

| Value | Move Source | Notes |
|---|---|---|
| | | AUX = 10b: use Class 2 Alignment Block<br>AUX = 11b: error |
| Dh | Class 1 Key | Error if C1 Key Size has been written either directly or by the KEY command and not cleared. |
| Eh | Class 2 Key | Error if C2 Key Size has been written either directly or by the KEY command and not cleared. |
| All other values are reserved | | |

Table 7-38. Move destinations

| Value | Move Destination | Notes |
|---|---|---|
| 0h | Class 1 Context | — |
| 1h | Class 2 Context | — |
| 2h | Output Data FIFO | |
| 3h | Descriptor Buffer | — |
| 4h | Math Register 0 | — |
| 5h | Math Register 1 | — |
| 6h | Math Register 2 | — |
| 7h | Math Register 3 | — |
| 8h | Input Data FIFO (C1) | If Automatic NFIFO entries are enabled, the entries are generated for a Class 1 CHA. |
| 9h | Input Data FIFO (C2) | If Automatic NFIFO entries are enabled, the entries are generated for a Class 2 CHA. |
| Ah | Input Data FIFO | No NFIFO entry generated |
| Ch | PKHA A | If Automatic NFIFO entries are enabled, the entries are generated for the PKHA A RAM and the Flush bit is automatically set. |
| Dh | Class 1 Key | — |
| Eh | Class 2 Key | |
| Fh | Auxiliary Data FIFO | Data can be moved to the Auxiliary Data FIFO so that it can later be used as an input to one or more of the Alignment Blocks. (An NFIFO entry with AST = 1 and STYPE = 00 should be created before the MOVE, else DECO may hang.) Note that a LOAD IMM to destination 78 can also be used to supply data to the Auxiliary Data FIFO. If multiple MOVEs and/or MOVEs and LOADs are used to provide data to the Auxiliary Data FIFO, the MOVE commands may need the WC bit set to ensure that the data is not overwritten. |
| All other values are reserved | | |

## 7.16 ALGORITHM OPERATION command

The OPERATION command (CTYPE = 10000) defines the type of cryptographic operation that SEC performs. Setting OPTYPE = 010 or 100 specifies an ALGORITHM OPERATION. Setting OPTYPE = 000, 110, or 111 specifies a PROTOCOL OPERATION (see PROTOCOL OPERATION commands). Setting OPTYPE = 001 specifies a PKHA OPERATION (see PKHA OPERATION command). The operation can range from performing a simple operation using a single CHA to performing a complex operation involving multiple CHAs and multiple steps. More than one OPERATION command can be used in a descriptor, allowing Class 1 and Class 2 operations to be specified separately. For the ALGORITHM OPERATION command, the fields of the command are as shown in the following table. Note that bits 23-0 of the ALGORITHM OPERATION command are automatically written to the appropriate CHA's mode register.

### Table 7-39. ALGORITHM OPERATION command format

| | 31-27 | | | | 26-24 | | | 23-16 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CTYPE = 10000b | | | | OPTYPE = 010b or 100b | | | ALG | | | | |
| | 15-14 | 13 | 12-4 | | | | | | | 3-2 | 1 | 0 |
| *format as it appears for CHAs other than RNG:* | Reserved | C2K | AAI | | | | | | | AS | ICV | ENC |
| | 15-13 | | 12 | 11 | 10 | 9 | 8 | 7-6 | 5-4 | 3-2 | 1 | 0 |
| *format as it appears for RNG:* | Reserved | | SK | AI | PS | OBP | NZB | Reserved | SH | AS | PR | TST |

### Table 7-40. ALGORITHM OPERATION command field descriptions

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>If CTYPE=10000b : OPERATION command; (ALGORITHM OPERATION or PKHA OPERATION or PROTOCOL OPERATION, as determined by the OPTYPE field) |
| 26-24<br>OPTYPE | Operation Type<br>If OPTYPE = 010b or 100b : ALGORITHM OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in the field descriptions below.<br>If OPTYPE = 010b : Class 1 algorithm operation<br>If OPTYPE = 100b : Class 2 algorithm operation<br>If OPTYPE = 001b : PKHA OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PKHA OPERATION command.<br>If OPTYPE = 000b, 011b or 111b : PROTOCOL OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PROTOCOL OPERATION command. |
| 23-16<br>ALG | Algorithm<br>This field specifies the algorithm that is to be used for the operations.<br>• If OPTYPE = 010b (Class 1 algorithm)<br>  • If ALG=10h : AES<br>  • If ALG=20h : DES<br>  • If ALG=21h : 3DES<br>  • If ALG=50h : RNG<br>  • If ALG=60h : SNOW 3G f8<br>  • If ALG=70h : Kasumi f8 or f9 |

*Table continues on the next page...*

## Table 7-40. ALGORITHM OPERATION command field descriptions (continued)

| Field | Description |
|---|---|
| | - If ALG=B0h : ZUC encryption<br>- All other values are reserved.<br>• If OPTYPE = 100b (Class 2 algorithm)<br>- If ALG=10h : AES (only valid for CMAC and XCBC modes)<br>- If ALG=40h : MD5<br>- If ALG=41h : SHA-1<br>- If ALG=42h : SHA-224<br>- If ALG=43h : SHA-256<br>- If ALG=44h : SHA-384<br>- If ALG=45h : SHA-512<br>- If ALG=46h : SHA-512/224<br>- If ALG=47h : SHA-512/256<br>- If ALG=90h : CRC<br>- If ALG=A0h : SNOW 3G f9<br>- If ALG=C0h : ZUC authentication<br>- All other values are reserved. |
| 15-14 | Reserved |
| 13<br>C2K | Class 2 Key<br>This bit is ignored for all algorithms other than AES.<br>0: AES uses the Class 1 key for CCM and GCM modes.<br>1: AES uses the Class 2 key for CCM and GCM modes. Setting this bit = 1 results in a mode error for other AES modes. |
| 12-4<br>AAI | Additional Algorithm Information<br>This field contains additional mode information that is associated with the algorithm that is being executed. See the tables below for details specific to individual algorithms. See also the section describing the appropriate CHA. Note that some algorithms do not require additional algorithm information and in those cases this field should be all 0s.<br>For RNG OPERATION commands the AAI field is interpreted as shown in the shaded SK, AI, PS, OBP, NZ and SH fields below. |
| 3-2<br>AS | Algorithm State<br>This field defines the state of the algorithm that is being executed. This may not be used by every algorithm. For RNG commands, see the shaded AS field below.<br>00 Update<br>01 Initialize<br>10 Finalize<br>11 Initialize/finalize |
| 1<br>ICV | ICV Checking<br>For the definition of this bit in RNG commands, see the shaded PR field below. This bit selects whether the current algorithm should compare the known ICV versus the calculated ICV. This bit is ignored by algorithms that do not support ICV checking. |
| 0<br>ENC | Encrypt/Decrypt<br>For the definition of this bit in RNG commands, see the shaded TST field below This bit selects encryption or decryption. This bit is ignored by all algorithms that do not have distinct encryption and decryption modes. However, for performance counting to be done correctly, this bit must be set appropriately even if the CHA or Algorithm does not use it to select cryptographic modes.<br>0 Decrypt<br>1 Encrypt |

*Table continues on the next page...*

## Table 7-40. ALGORITHM OPERATION command field descriptions (continued)

| Field | Description |
|---|---|
| *The rows below describe how bits 12-0 are interpreted for RNG commands.* | |
| 12<br>SK<br>(RNG only) | Secure Key. For RNG OPERATION commands this bit of the AAI field is interpreted as the Secure Key field. If SK=1 and AS=00 (Generate), the RNG will generate data to be loaded into the JDKEK, TDKEK and TDSK. If a second Generate command is issued with SK=1, a Secure Key error will result. If SK=0 and AS=00 (Generate), the RNG will generate data to be stored as directed by the FIFO STORE command. The SK field is ignored if AS≠00. |
| 11<br>AI<br>(RNG only) | Additional Input Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Additional Input Included field. If AS=00 (Generate) and AI=1, the 256 bits of additional data supplied via the Class 1 Context Register will be used as additional entropy during random number generation. If AS=10 (Reseed) and AI=1, the additional data supplied via the Class 1 Context register will be used as additional entropy input during the reseeding operation. The AI field is ignored if AS=01 (Instantiate) or AS=11 (Uninstantiate). |
| 10<br>PS<br>(RNG only) | Personalization String Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Personalization String Included field. If AS=01 (Instantiate) and PS=1, a personalization string of 256 bits supplied via the Class 1 Context register is used as additional "entropy" input during instantiation. Note that the personalization string does not need to be random. A device-unique value can be used to further guarantee that no two RNGs are ever instantiated with the same seed value. (Note that the entropy generated by the TRNG already ensures this with high probability.) The PS field is ignored if AS≠01. |
| 9<br>OBP<br>(RNG only) | Odd Byte Parity. For RNG OPERATION commands this bit of the AAI field is interpreted as the Odd Byte Parity field. If AS=00 (Generate) and OBP=1, every byte of data generated during random number generation will have odd parity. That is, the 128 possible bytes values that have odd parity will be generated at random. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The OBP field is ignored if AS≠00. |
| 8<br>NZB<br>(RNG only) | NonZero bytes. For RNG OPERATION commands this bit of the AAI field is interpreted as the NonZero Bytes field. If AS=00 (Generate) and NZB=1, no byte of data generated during random number generation will be 00, but (if OBP=0) the remaining 255 values will be generated at random. Note that setting NZB=1 has no effect if OBP=1, since zero bytes are already excluded when odd byte parity is selected. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The NZB field is ignored if AS≠00. |
| 7-6<br>(RNG only) | Reserved. For RNG commands these bits of the AAI field are reserved. |
| 5-4<br>SH<br>(RNG only) | State Handle. For RNG OPERATION commands these bits of the AAI field are interpreted as the State Handle field. The command is issued to the State Handle selected via this field. An error will be generated if the selected state handle is not implemented.<br>00 State Handle 0<br>01 State Handle 1<br>10 Reserved<br>11 Reserved |
| 3-2<br>AS<br>(RNG only) | Algorithm State. For RNG OPERATION commands these bits select RNG commands as shown in Table 7-48. |
| 1<br>PR<br>(RNG only) | Prediction Resistance. For RNG OPERATION commands this bit is interpreted as shown in Table 7-49. |
| 0 | Test Mode Request. For RNG OPERATION commands this bit is interpreted as shown in Table 7-50. |

## Table 7-40. ALGORITHM OPERATION command field descriptions

| Field | Description |
|---|---|
| TST<br>(RNG only) | |

## Table 7-41. AAI Interpretation for AES modes

| AAI Interpretation for AES Modes (See AES accelerator (AESA) functionality) | | | |
|---|---|---|---|
| Code | Interpretation | Code | Interpretation |
| 00h | CTR (mod $2^{128}$) | 80h | CCM (mod $2^{128}$) Note, if C2K= 0 CCM and GCM use the key in the Class 1 Key Register. If C2K = 1 CCM and GCM use the key in the Class 2 Key Register. |
| 10h | CBC | 90h | GCM (mod $2^{32}$) |
| 20h | ECB | A0h | CBC_XCBC_MAC |
| 30h | CFB | B0h | CTR_XCBC_MAC |
| 40h | OFB | C0h | CBC_CMAC |
| 50h | XTS | D0h | CTR_CMAC_LTE |
| 60h | CMAC | E0h | CTR_CMAC |
| 70h | XCBC-MAC | | |
| The codes listed above are mutually exclusive, which means that they cannot be ORed with each other. | | | |
| Note that for AES the MSB of AAI is the DK (Decrypt Key) bit. Setting the DK bit (that is, ORing 100h with any AES code above) causes the Key Register to be loaded with the AES Decrypt key, rather than the AES Encrypt key. See the discussion in AES accelerator (AESA) functionality. Note that AES normally acts as a Class 1 CHA, but for CMAC or XCBC-MAC modes, it can also be used as a Class 2 CHA. When a Class 2 OPERATION command specifies AES with CMAC or XCBC-MAC, it may be accompanied by a Class 1 OPERATION command specifying AES, if (and only if) the Class 1 OPERATION command specifies a Confidentiality-only mode. Specifying a Class 2 AES OPERATION command in concert with a Class 1 AES Operation command specifying either CCM or GCM is not permitted and will result in an error. Combo modes CBC_XCBC_MAC, CTR_XCBC_MAC, CBC_CMAC, CTR_CMAC_LTE, and CTR_CMAC are available for general use, but were specified for IPsec and LTE protocol use. With the extension to AES permitting simultaneous Class 1 Confidentiality-only and Class 2 Integrity OPERATION, these Combo modes are no longer recommended and may be deprecated in the future. | | | |

## Table 7-42. AAI Interpretation for DES modes

| AAI Interpretation for DES modes (See Data encryption standard accelerator (DES) functionality) | | | |
|---|---|---|---|
| Code | Interpretation | Code | Interpretation |
| 10h | CBC | 30h | CFB |
| 20h | ECB | 40h | OFB |
| The codes listed above are mutually exclusive, which means that they cannot be ORed with each other. | | | |
| 80h ORed with any DES code above: Check odd parity | | | |

## Table 7-43. AAI Interpretation for MD5 and SHA modes

| AAI Interpretation for MD5, SHA-1, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 (See Message digest hardware accelerator (MDHA) functionality) | | | | |
|---|---|---|---|---|
| Code | Interpretation | | Code | Interpretation |
| 00h | Hash without key | | 02h | SMAC |
| 01h | HMAC | | 04h | HMAC with precomputed IPAD and OPAD |

## Table 7-44. AAI Interpretation for CRC modes

| AAI Interpretation for CRC modes (See Cyclic-redundancy check accelerator (CRCA) functionality) | | | | |
|---|---|---|---|---|
| Code | Interpretation | | Code | Interpretation |
| 01h | IEEE 802 | | 10h | DIS |
| 02h | IETF 3385 | | 20h | DOS |
| 04h | CUST_POLY | | 40h | DOC |
| | | | 80h | IVZ (initial value zero) |
| CRC codes in the right column can be ORed with CRC codes in the left column to specify DIS, DOS, DOC and IVZ<br>(See CRCA use of the Mode Register) | | | | |

## Table 7-45. AAI Interpretation for Kasumi modes

| AAI Interpretation for Kasumi 3G modes (See Kasumi f8 and f9 hardware accelerator(KFHA) functionality) | | | | |
|---|---|---|---|---|
| Code | Interpretation | | Code | Interpretation |
| C0h | Kasumi 3G f8 (encryption/decryption) | | 10h | GSM |
| C8h | Kasumi 3G f9 (authentication) | | 20h | EDGE |

## Table 7-46. AAI Interpretation for SNOW 3G modes

| AAI Interpretation for SNOW 3G modes (See SNOW 3G f8 accelerator functionality and SNOW 3G f9 accelerator functionality) | |
|---|---|
| Code | Interpretation |
| C0h | SNOW 3G f8 (encryption/decryption) |
| C8h | SNOW 3G f9 (authentication) |

## Table 7-47. AAI Interpretation for ZUC modes

| AAI Interpretation for ZUC (See ZUC encryption accelerator (ZUCE) functionality) | |
|---|---|
| Code | Interpretation |
| C0h | ZUCE (encryption/decryption) |
| C8h | ZUCA (authentication) |

## Table 7-48. AS RNG OPERATION command settings

| AS Value | State Handle is already instantiated | State Handle is NOT already instantiated |
|---|---|---|
| 00 Generate | Generate random data per the mode in which the state handle was instantiated. | Error |
| 01 Instantiate | Error | Instantiate the state handle in either test mode or nondeterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR. |
| 10 Reseed | Reseed the state handle. | Error |
| 11 Uninstantiate | Uninstantiate the state handle. | Error |

## Table 7-49. PR RNG Operation commands setting

| AS Value | PR = 0 | PR = 1 |
|---|---|---|
| 00 Generate | Do NOT reseed prior to generating new random data. PR bit must be zero. | If the state handle was instantiated to support prediction resistance, reseed prior to generating new random data. If the state handle was NOT instantiated to support prediction resistance, generate an error. |
| 01 Instantiate | Instantiate the state handle to NOT support prediction resistance | Instantiate the state handle to support prediction resistance |
| 10 Reseed | Reseed the state handle. PR bit must be zero. | Reseed the state handle. PR bit is ignored. |
| 11 Uninstantiate | Uninstantiate the state handle. PR bit must be zero. | Uninstantiate the state handle. PR bit is ignored. |

## Table 7-50. TST RNG OPERATION command settings

| AS Value | TST = 0 | TST = 1 |
|---|---|---|
| 00 Generate | • If the selected state handle is in nondeterministic mode, generate new random data.<br>• If the selected state handle is in deterministic mode, generate a Test error.[1] | • If the selected state handle is in deterministic mode, generate new random data.<br>• If the selected state handle is in nondeterministic mode, generate a Test error. |
| 01 Instantiate | Instantiate the state handle in normal (nondeterministic) mode. | Instantiate the state handle in test (deterministic) mode. |
| 10 Reseed | • If the selected state handle is in nondeterministic mode, reseed the state handle.<br>• If the selected state handle is in deterministic mode, generate a Test error.[2] | • If the selected state handle is in deterministic mode, reseed the state handle.<br>• If the selected state handle is in nondeterministic mode, generate a Test error. |
| 11 Uninstantiate | • If the selected state handle is in nondeterministic mode, uninstantiate the state handle.<br>• If the selected state handle is in deterministic mode, generate a Test error.[3] | • If the selected state handle is in deterministic mode, uninstantiate the state handle.<br>• If the selected state handle is in nondeterministic mode, generate a Test error. |

[1] There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in test mode but a Generate operation requests nondeterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.

2. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in test mode but a non test reseed operation is requested State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.
3. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in test mode but a non test uninstantiate operation is requested for State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.

## 7.17 PROTOCOL OPERATION commands

The OPERATION command (CTYPE = 10000) defines the type of cryptographic operation that SEC performs. The OPERATION command's Protocol OpType takes advantage of well-known processing steps for standardized security protocols, so that the user can invoke an existing hard-coded command sequence rather than having to use SEQ commands to create a complex descriptor.

If the OPTYPE specifies a protocol operation (000, 110, 111), the OPERATION command fields are as shown in Table 7-52. If OPTYPE specifies an algorithm operation (OPTYPE = 010: Class 1, OPTYPE = 100: Class 2), see ALGORITHM OPERATION command. If OPTYPE specifies a PKHA operation (OPTYPE = 001), see PKHA OPERATION command.

**Table 7-51. PROTOCOL OPERATION command format**

| 31-27 | 26-24 | 23-16 |
|---|---|---|
| CTYPE = 10000 | OPTYPE = 000, 110, or 111 | PROTID |
| 15-0 | | |
| PROTINFO | | |

Protocols are used to execute complex built-in functions. Before beginning a protocol operation, DECO waits for any pending (SEQ) FIFO STORE operations to complete. When starting the protocol operation, DECO resets the output data FIFO; any data remaining in the output data FIFO from previous operations is lost. It is the responsibility of the programmer to ensure that once the protocol starts, no data is pushed into the output data FIFO as a result of commands executed prior to the protocol operation. It is the responsibility of the programmer to ensure that once the protocol starts, no data is in, or will be pushed into, the input data FIFO or information FIFO as a result of commands executed prior to the protocol operation.

The protocol ID codes and information on PROTINFO encoding are shown in Table 7-53,

## Table 7-52. PROTOCOL OPERATION command field descriptions

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>If CTYPE=10000b : OPERATION command; (ALGORITHM OPERATION or PKHA OPERATION or PROTOCOL OPERATION, as determined by the OPTYPE field) |
| 26-24<br>OPTYPE | Operation Type<br>If OPTYPE = 000b, 110b or 111b : PROTOCOL OPERATION; The OPTYPE field indicates the "directionality" of the protocol as shown below. The PROTID field is interpreted as shown in the following PROTID field description table.<br>If OPTYPE=000b : Unidirectional protocol<br>If OPTYPE=110b : Decapsulation protocol<br>If OPTYPE=111b : Encapsulation protocol<br>If OPTYPE = 010b or 100b : ALGORITHM OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in ALGORITHM OPERATION command.<br>If OPTYPE = 001b : PKHA OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PKHA OPERATION command.<br>All others: reserved |
| 23-16<br>PROTID | Protocol Identifier<br>This field represents the specific protocol that this descriptor is implementing. See Table 7-53 for the complete description. |
| PROTINFO<br>15-0 | This value is protocol-dependent. |

## Table 7-53. PROTID and PROTINFO field description

| PROTID (hex) | Description | PROTINFO Information |
|---|---|---|
| 01 | OPTYPE 000: IKE PRF<br>OPTYPE 11x: IPsec ESP Transport (and legacy ESP Tunnel)<br>else Reserved | For IKE PRF and PRF+, the PROTINFO field is defined in Table 7-56. For further information concerning IKE PRF, refer to IKEv2 PRF overview.<br>For IPsec ESP Transport (and legacy Tunnel) and IPsec ESP Tunnel, the PROTINFO field is defined in Table 7-54. For further information concerning IPsec ESP Transport (and legacy Tunnel), IPsec ESP Tunnel, refer to IPsec ESP encapsulation and decapsulation overview. |
| 02 | OPTYPE 000: IKE PRF+<br>OPTYPE 11x: SRTP<br>else Reserved | For IKE PRF and PRF+, the PROTINFO field is defined in Table 7-56. For further information concerning IKE PRF+, refer to Implementation Details for IKE PRF+ function<br>For SRTP, the PROTINFO field is defined in Table 7-54. For further information concerning SRTP, refer to SRTP packet encapsulation and decapsulation. |
| 03 | OPTYPE 11x: MACsec (802.1AE)<br>else Reserved | For MACsec, use PROTINFO = 0000_0000_0000_0001 to specify AES_GCM_16. Use PROTINFO = 0000_0000_0000_0002 to specify AES_GCM_32. Use PROTINFO = 0000_0000_0000_0003 to specify AES_GCM_16 with extended PN. Use PROTINFO = 0000_0000_0000_0004 to specify AES_GCM_32 with |

*Table continues on the next page...*

## Table 7-53. PROTID and PROTINFO field description (continued)

| PROTID (hex) | Description | PROTINFO Information |
|---|---|---|
| | | extended PN. For further information concerning MACsec, refer to IEEE 802.1AE MACsec encapsulation and decapsulation overview. |
| 04 | OPTYPE 11x: IEEE 802.11-2012 WPA2 MPDU for WiFi<br>else Reserved | For WPA2, use PROTINFO = 1010_1100_0000_0100 to specify AES_CCM_8 (CCMP).<br>For further information concerning WPA2, refer to IEEE 802.11-2012 WPA2 MPDU encapsulation and decapsulation . |
| 05 | OPTYPE 11x: WiMAX (802.16)<br>else Reserved | For WiMAX OFDM, use PROTINFO = 0000_0010_0000_0001 to specify AES_CCM_16 with CRC for OFDM. Use PROTINFO = 0000_0000_0000_0001 to specify CRC-only for OFDM. For further information concerning WiMAX refer to IEEE 802.16 WiMAX encapsulation and decapsulation overview.<br>For WiMAX OFDMa, use PROTINFO = 0000_0010_0011_0001 to specify AES_CCM_16 with CRC for OFDMa. Use PROTINFO = 0000_0000_0011_0001 to specify CRC-only for OFDMa. For further information concerning WiMAX refer to IEEE 802.16 WiMAX encapsulation and decapsulation overview. |
| 08 | OPTYPE 000: SSL 3.0 PRF<br>**NOTE:** Descriptors that include a TLS PRF command are limited to 50 words in length.<br>OPTYPE 110: SSL3.0 Decapsulation<br>OPTYPE 111: SSL 3.0 Encapsulation<br>else Reserved | For the SSL/TLS/DTLS protocol family, the PROTINFO field is defined in Table 7-55. For information on SSL 3.0 PRF, refer to Process for SSL 3.0 PRF. For further information concerning SSL/3.0T record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview. |
| 09 | OPTYPE 000: TLS 1.0 PRF<br>**NOTE:** Descriptors that include a TLS PRF command are limited to 50 words in length.<br>OPTYPE 110: TLS 1.0 Decapsulation<br>OPTYPE 111: TLS 1.0 Encapsulation<br>else Reserved | For the SSL/TLS/DTLS protocol family, the PROTINFO field is defined in Table 7-55. For information on TLS 1.0 PRF, refer to Process for TLS 1.0, TLS 1.1, DTLS PRF. For further information concerning TLS 1.0 record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview. |
| 0A | OPTYPE 000: TLS 1.1 PRF<br>**NOTE:** Descriptors that include a TLS PRF command are limited to 50 words in length.<br>OPTYPE 110: TLS 1.1 Decapsulation<br>OPTYPE 111: TLS 1.1 Encapsulation<br>else Reserved | For the SSL/TLS/DTLS protocol family, the PROTINFO field is defined in Table 7-55. For information on TLS 1.1 PRF, refer to Process for TLS 1.0, TLS 1.1, DTLS PRF. For further information concerning TLS 1.1 record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview. |
| 0B | OPTYPE 000: TLS 1.2 PRF<br>using HMAC-SHA-256 except when another HMAC is expressly stated as shown in Table 7-55 | For the SSL/TLS/DTLS protocol family, the PROTINFO field is defined in Table 7-55. For information on TLS 1.2 PRF, refer to Process for TLS 1.2 PRF. For further information concerning TLS 1.2 record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview. |

*Table continues on the next page...*

## Table 7-53. PROTID and PROTINFO field description (continued)

| PROTID (hex) | Description | PROTINFO Information |
|---|---|---|
| | **NOTE:** Descriptors that include a TLS PRF command are limited to 50 words in length.<br>OPTYPE 110: TLS 1.2 Decapsulation<br>OPTYPE 111: TLS 1.2 Encapsulation<br>else Reserved | |
| 0C | OPTYPE 000: DTLS 1.0 PRF<br>**NOTE:** Descriptors that include a TLS PRF command are limited to 50 words in length.<br>OPTYPE 110: DTLS 1.0 Decapsulation<br>OPTYPE 111: DTLS 1.0 Encapsulation<br>else Reserved | For the SSL/TLS/DTLS protocol family, the PROTINFO field is defined in Table 7-55. For information on DTLS 1.0 PRF, refer to Process for TLS 1.0, TLS 1.1, DTLS PRF. For further information concerning DTLS 1.0 record processing, refer to SSL/TLS/DTLS record encapsulation and decapsulation overview. |
| 0D | For OPTYPE 110 or 111: Blob | For blobs encapsulation or decapsulation, the PROTINFO field is defined in Table 7-59 and Table 7-60. For further information concerning blobs, see Blobs. |
| 0F | For OPTYPE 110: Anti-Replay | Stand-alone Anti-Replay checking always uses a PROTINFO code of 0000h, and is described in Anti-Replay built-in checking. |
| 11 | OPTYPE 11x: IPsec ESP Tunnel<br>else Reserved | For IPsec ESP Tunnel, the PROTINFO field is defined in Table 7-54. For further information concerning these protocols, refer to IPsec ESP encapsulation and decapsulation overview. |
| 12 | OPTYPE 000: (EC)DSA Verify with Private Key<br>else Reserved | For (EC)DSA Verify using Private Key, the PROTINFO field is defined in Table 7-62. For further information, see Verifying DSA and ECDSA signatures. |
| 14 | OPTYPE 000: DH, DSA, and ECC Key Pair Generation<br>OPTYPE 110: MPPubK generation<br>OPTYPE 111: MPPrivK generation<br>else Reserved | For Key Pair Generation, MPPubK and MPPrivK, the PROTINFO field is defined in Table 7-62. For further information, see Discrete-log key-pair generation |
| 15 | OPTYPE 000: (EC)DSA_Sign<br>OPTYPE 110: MPSign<br>else Reserved | For (EC)DSA Sign, and MPSign, the PROTINFO field is defined in Table 7-62. For further information, see Generating DSA and ECDSA signatures. |
| 16 | OPTYPE 000: (EC)DSA_Verify<br>else Reserved | For (EC)DSA Verify, the PROTINFO field is defined in Table 7-62. For further information, see Verifying DSA and ECDSA signatures. |
| 17 | OPTYPE 000: (EC)Diffie-Hellman<br>else Reserved | For (EC)Diffie-Hellman, the PROTINFO field is defined in Table 7-62. For further information, see Using the Diffie_Hellman function. |
| 18 | OPTYPE 000: RSA_Encrypt<br>else Reserved | For RSA_Encrypt, the PROTINFO field is defined in Table 7-64. For further information concerning RSA Encrypt see Implementation of the RSA encrypt operation. |

*Table continues on the next page...*

## Table 7-53. PROTID and PROTINFO field description (continued)

<table>
<tr><th>PROTID (hex)</th><th>Description</th><th>PROTINFO Information</th></tr>
<tr><td>19</td><td>OPTYPE 000: RSA_Decrypt<br>else Reserved</td><td>For RSA_Decrypt, the PROTINFO field is defined in Table 7-66. For further information concerning RSA Decrypt see Implementation of the RSA decrypt operation.</td></tr>
<tr><td>1A</td><td>OPTYPE 000: RSA_Finish_KeyGen<br>else Reserved</td><td>For RSA_Finish_Keygen, the PROTINFO field is defined in Table 7-67. See RSA Finalize Key Generation (RFKG) for further information.</td></tr>
<tr><td>1E</td><td>OPTYPE 000: EC Public Key Validation<br>else Reserved</td><td>For EC Public Key Validation use bit [0] to select F2m validation. No other PROTINFO bits are used.</td></tr>
<tr><td>20</td><td>OPTYPE 000: Derived Key MD5<br>else Reserved</td><td>For Derived Key MD5, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.</td></tr>
<tr><td>21</td><td>OPTYPE 000: Derived Key SHA1<br>else Reserved</td><td>For Derived Key SHA1, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.</td></tr>
<tr><td>22</td><td>OPTYPE 000: Derived Key SHA224<br>else Reserved</td><td>For Derived Key SHA224, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.</td></tr>
<tr><td>23</td><td>OPTYPE 000: Derived Key SHA256<br>else Reserved</td><td>For Derived Key SHA256, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.</td></tr>
<tr><td>24</td><td>OPTYPE 000: Derived Key SHA384<br>else Reserved</td><td>For Derived Key SHA384, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.</td></tr>
<tr><td>25</td><td>OPTYPE 000: Derived Key SHA512<br>else Reserved</td><td>For Derived Key SHA512, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol.</td></tr>
<tr><td>2F</td><td>else Reserved</td><td></td></tr>
<tr><td>31</td><td>OPTYPE 11x: 3G Double CRC<br>else Reserved</td><td>For 3G Double CRC, the PROTINFO field is defined in Table 7-57</td></tr>
<tr><td>32</td><td>OPTYPE 11x: 3G RLC<br>else Reserved</td><td>For 3G RLC, the PROTINFO field is defined in Table 7-57. For further information concerning 3G RLC, see 3G RLC PDU Encapsulation and Decapsulation overview.</td></tr>
<tr><td>42</td><td>OPTYPE 11x: LTE PDCP User Plane<br>else Reserved</td><td>For LTE PDCP, the PROTINFO field is defined in Table 7-57. For further information concerning LTE, see 3G RLC PDU Encapsulation and Decapsulation overview.</td></tr>
<tr><td>43</td><td>OPTYPE 11x: LTE PDCP Control Plane - deprecated in favor of PROTID 44, which supports mixed cipher suites. PROTID 43 does not.<br>else Reserved</td><td>For LTE PDCP (PROTID=43h), the PROTINFO field is defined in Table 7-57. For further information concerning LTE, see LTE PDCP PDU encapsulation and decapsulation overview</td></tr>
<tr><td>44</td><td>OPTYPE 11x: LTE PDCP Control Plane - supports mixed cipher suites<br>else Reserved</td><td>For LTE Control Plane (PROTID=44h) the PROTINFO field is interpreted as follows:
<table>
<tr><th colspan="2">(bits 15:8) - Encrypt/ Decrypt Algorithm</th><th colspan="2">(bits 7:0) - Authentication Algorithm</th></tr>
<tr><td>00h</td><td>null</td><td>00h</td><td>null</td></tr>
</table>
</td></tr>
</table>

*Table continues on the next page...*

## Table 7-53. PROTID and PROTINFO field description (continued)

<table>
<tr><th>PROTID (hex)</th><th>Description</th><th>PROTINFO Information</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th colspan="2">(bits 15:8) - Encrypt/ Decrypt Algorithm</th><th colspan="2">(bits 7:0) - Authentication Algorithm</th></tr>
<tr><td>01h</td><td>SNOW3G</td><td>01h</td><td>SNOW3G</td></tr>
<tr><td>02h</td><td>AES</td><td>02h</td><td>AES</td></tr>
<tr><td>03h</td><td>ZUC</td><td>03h</td><td>ZUC</td></tr>
<tr><td colspan="2">all other values are reserved</td><td colspan="2">all other values are reserved</td></tr>
<tr><td colspan="4">Note: Header meta data is not allowed for the following combinations:<br>SNOW3G Encrypt and AES Authentication or ZUC Encrypt and AES Authentication</td></tr>
<tr><td colspan="4">Note: Trailing meta data is not allowed for the following combinations:<br>SNOW3G Encrypt/Decrypt and AES Authentication or ZUC Encrypt/Decrypt and AES Authentication</td></tr>
</table>
</td></tr>
<tr><td>45</td><td>OPTYPE 11x: LTE PDCP PDU User Plane for RN<br>else Reserved</td><td>For 3G RLC, the PROTINFO field is defined as shown in the table above (PROTID=44). For further information concerning 3G RLC, see 3G RLC PDU Encapsulation and Decapsulation overview.</td></tr>
<tr><td>60</td><td>OPTYPE 000: Derived Key MD5 with RIF<br>else Reserved</td><td>For Derived Key MD5 with RIF, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 7-12.</td></tr>
<tr><td>61</td><td>OPTYPE 000: Derived Key SHA1 with RIF<br>else Reserved</td><td>For Derived Key SHA1 with RIF, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 7-12.</td></tr>
<tr><td>62</td><td>OPTYPE 000: Derived Key SHA224 with RIF<br>else Reserved</td><td>For Derived Key SHA224 with RIF, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 7-12.</td></tr>
<tr><td>63</td><td>OPTYPE 000: Derived Key SHA256 with RIF<br>else Reserved</td><td>For Derived Key SHA256 with RIF, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 7-12.</td></tr>
<tr><td>64</td><td>OPTYPE 000: Derived Key SHA384 with RIF<br>else Reserved</td><td>For Derived Key SHA384 with RIF, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 7-12.</td></tr>
</table>

*Table continues on the next page...*

## Table 7-53. PROTID and PROTINFO field description (continued)

| PROTID (hex) | Description | PROTINFO Information |
|---|---|---|
| 65 | OPTYPE 000: Derived Key SHA512 with RIF<br>else Reserved | For Derived Key SHA512 with RIF, the PROTINFO field is defined in Table 7-58. For further information concerning Derived Key Protocol, see Implementation of the derived key protocol. For further information concerning the RIF option, see Table 7-12. |
| 6F | else Reserved | |
| All other values reserved. | | |

## Table 7-54. PROTINFO definition when used with IPsec or SRTP protocols

| PROTINFO [15:8] (hex) | Encryption algorithm | Notes | PROTINFO [7:0] (hex) | Authentication algorithm |
|---|---|---|---|---|
| 01 | DES | For IPsec ESP Transport (and legacy Tunnel), IPsec ESP Tunnel, any encryption algorithm at left can be used with any authentication algorithm at right.<br>For SRTP, these encryption and authentication algorithms are not permitted (with the exception of AES-CTR, see below). | 00 | NULL authentication, also used with AES-CCM and AES-GCM |
| 02 | DES (same as above) | | 01 | HMAC_MD5_96 |
| 03 | 3DES | | 02 | HMAC_SHA1_96 |
| 0B | NULL Encryption | | 05 | AES_XCBC_MAC_96 |
| 0C | AES-CBC | | 06 | HMAC_MD5_128 |
| 0D | AES-CTR | | 07 | HMAC_SHA1_160 |
| | | | 08 | AES-CMAC-96 |
| | | | 0C | HMAC_SHA2_256_128 |
| | | | 0D | HMAC_SHA2_384_192 |
| | | | 0E | HMAC_SHA2_512_256 |
| 0D | AES-CTR | For SRTP, when using AES_CTR as the encryption algorithm HMAC_SHA1_160 must be selected as the authentication algorithm. | 07 | HMAC_SHA1_160 |
| 0E | AES-CCM-8 | For both IPsec and SRTP, when using the algorithms at left PROTINFO[7:0] must be 00. | 00 | Authentication is incorporated into the encryption algorithm. |
| 0F | AES-CCM-12 | | | |
| 10 | AES-CCM-16 | | | |
| 12 | AES-GCM-8 | | | |
| 13 | AES-GCM-12 | | | |
| 14 | AES-GCM-16 | | | |
| 15 | AES-NULL-WITH-GMAC | | | |
| All other values are reserved. | | | | |

## Table 7-55. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs)

| PROT INFO [15:0] (hex) | Description | | PROT INFO [15:0] (hex) | Description |
|---|---|---|---|---|
| 0000 | NULL_TLS. PRF not supported. | | | |
| 0001 | NULL_MD5. PRF not supported. | | | |
| 0002 | NULL_SHA. PRF not supported. | | | |
| 0003 | Reserved. | | 002F | AES_128_CBC_SHA |
| 0004 | Reserved. | | 0030 | AES_128_CBC_SHA |
| 0005 | Reserved. | | 0031 | AES_128_CBC_SHA |
| | | | 0032 | AES_128_CBC_SHA |
| 0008 | DES40_CBC_SHA | | 0033 | AES_128_CBC_SHA |
| 0009 | DES_CBC_SHA | | 0034 | AES_128_CBC_SHA |
| 000A | 3DES_EDE_CBC_SHA | | 0035 | AES_256_CBC_SHA |
| 000B | DES40_CBC_SHA | | 0036 | AES_256_CBC_SHA |
| 000C | DES_CBC_SHA | | 0037 | AES_256_CBC_SHA |
| 000D | 3DES_EDE_CBC_SHA | | 0038 | AES_256_CBC_SHA |
| 000E | DES40_CBC_SHA | | 0039 | AES_256_CBC_SHA |
| 000F | DES_CBC_SHA | | 003A | AES_256_CBC_SHA |
| 0010 | 3DES_EDE_CBC_SHA | | 003B | NULL_SHA-256. PRF not supported. |
| 0011 | DES40_CBC_SHA | | 003C | AES_128_CBC_SHA-256 |
| 0012 | DES_CBC_SHA | | 003D | AES_256_CBC_SHA-256 |
| 0013 | 3DES_EDE_CBC_SHA | | 003E | AES_128_CBC_SHA-256 |
| 0014 | DES40_CBC_SHA | | 003F | AES_128_CBC_SHA-256 |
| 0015 | DES_CBC_SHA | | 0040 | AES_128_CBC_SHA-256 |
| 0016 | 3DES_EDE_CBC_SHA | | | |
| 0017 | Reserved. | | 0067 | AES_128_CBC_SHA-256 |
| 0018 | Reserved. | | 0068 | AES_256_CBC_SHA-256 |
| 0019 | DES40_CBC_SHA | | 0069 | AES_256_CBC_SHA-256 |
| 001A | DES_CBC_SHA | | 006A | AES_256_CBC_SHA-256 |
| 001B | 3DES_EDE_CBC_SHA | | 006B | AES_256_CBC_SHA-256 |
| | | | 006C | AES_128_CBC_SHA-256 |
| 001E | DES_CBC_SHA | | 006D | AES_256_CBC_SHA-256 |
| 001F | 3DES_EDE_CBC_SHA | | | |
| 0020 | Reserved. | | 008A | Reserved. |
| | | | 008B | 3DES_EDE_CBC_SHA |
| 0022 | DES_CBC_MD5 | | 008C | AES_128_CBC_SHA |
| 0023 | 3DES_EDE_CBC_MD5 | | 008D | AES_256_CBC_SHA |
| 0024 | Reserved. | | 008E | Reserved. |
| | | | 008F | 3DES_EDE_CBC_SHA |
| 0026 | DES_CBC_40_SHA | | | |
| | | | 0090 | AES_128_CBC_SHA |

*Table continues on the next page...*

## Table 7-55. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs) (continued)

| PROT INFO [15:0] (hex) | Description | | PROT INFO [15:0] (hex) | Description |
|---|---|---|---|---|
| 0028 | Reserved. | | 0091 | AES_256_CBC_SHA |
| 0029 | DES_CBC_40_MD5 | | 0092 | Reserved. |
| | | | 0093 | 3DES_EDE_CBC_SHA |
| 002B | Reserved. | | 0094 | AES_128_CBC_SHA |
| 002C | NULL_SHA. PRF not supported. | | 0095 | AES_256_CBC_SHA |
| 002D | NULL_SHA. PRF not supported. | | | |
| 002E | NULL_SHA. PRF not supported. | | | |
| | | | | |
| | | | 009C | Encap: AES_128_GCM; PRF: SHA-256 |
| | | | 009D | Encap: AES_256_GCM; PRF: SHA-384 |
| | | | 009E | Encap: AES_128_GCM; PRF: SHA-256 |
| | | | 009F | Encap: AES_256_GCM; PRF: SHA-384 |
| | | | | |
| 00A0 | Encap: AES_128_GCM; PRF: SHA-256 | | | |
| 00A1 | Encap: AES_256_GCM; PRF: SHA-384 | | | |
| 00A2 | Encap: AES_128_GCM; PRF: SHA-256 | | | |
| 00A3 | Encap: AES_256_GCM; PRF: SHA-384 | | | |
| 00A4 | Encap: AES_128_GCM; PRF: SHA-256 | | | |
| 00A5 | Encap: AES_256_GCM; PRF: SHA-384 | | | |
| 00A6 | Encap: AES_128_GCM; PRF: SHA-256 | | | |
| 00A7 | Encap: AES_256_GCM; PRF: SHA-384 | | | |
| 00A8 | Encap: AES_128_GCM; PRF: SHA-256 | | | |
| 00A9 | Encap: AES_256_GCM; PRF: SHA-384 | | C020 | AES_256_CBC_SHA |
| 00AA | Encap: AES_128_GCM; PRF: SHA-256 | | C021 | AES_256_CBC_SHA |
| 00AB | Encap: AES_256_GCM; PRF: SHA-384 | | C022 | AES_256_CBC_SHA |
| 00AC | Encap: AES_128_GCM; PRF: SHA-256 | | C023 | AES_128_CBC_SHA-256 |
| 00AD | Encap: AES_256_GCM; PRF: SHA-384 | | C024 | AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384 |
| 00AE | AES-128-CBC_SHA-256 | | C025 | AES_128_CBC_SHA-256 |
| 00AF | AES-256-CBC_SHA-384; PRF (if TLS 1.2): SHA-384 | | C026 | AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384 |
| 00B0 | NULL_SHA-256. PRF not supported. | | | |
| 00B1 | NULL_SHA-384. PRF not supported. | | C027 | AES_128_CBC_SHA-256 |
| 00B2 | AES-128-CBC_SHA-256 | | C028 | AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384 |
| 00B3 | AES-256-CBC_SHA-384; PRF (if TLS 1.2): SHA-384 | | C029 | AES_128_CBC_SHA-256 |
| 00B4 | NULL_SHA-256. PRF not supported. | | | |
| 00B5 | NULL_SHA-384. PRF not supported. | | C02A | AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384 |
| 00B6 | AES-128-CBC_SHA-256 | | C02B | Encap: AES_128_GCM; PRF: SHA-256 |

*Table continues on the next page...*

## Table 7-55. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs) (continued)

| PROT INFO [15:0] (hex) | Description | | PROT INFO [15:0] (hex) | Description |
|---|---|---|---|---|
| 00B7 | AES-256-CBC_SHA-384; PRF (if TLS 1.2): SHA-384 | | C02C | Encap: AES_256_GCM; PRF: SHA-384 |
| 00B8 | NULL_SHA-256. PRF not supported. | | | |
| 00B9 | NULL_SHA-384. PRF not supported. | | | |
| C001 | NULL_SHA. PRF not supported. | | C02D | Encap: AES_128_GCM; PRF: SHA-256 |
| C002 | Reserved. | | C02E | Encap: AES_256_GCM; PRF: SHA-384 |
| C003 | 3DES_EDE_CBC_SHA | | C02F | Encap: AES_128_GCM; PRF: SHA-256 |
| C004 | AES_128_CBC_SHA | | C030 | Encap: AES_256_GCM; PRF: SHA-384 |
| C005 | AES_256_CBC_SHA | | C031 | Encap: AES_128_GCM; PRF: SHA-256 |
| C006 | NULL_SHA. PRF not supported. | | C032 | Encap: AES_256_GCM; PRF: SHA-384 |
| C007 | Reserved. | | C033 | Reserved. |
| C008 | 3DES_EDE_CBC_SHA | | C034 | 3DES-CBC_SHA |
| C009 | AES_128_CBC_SHA | | C035 | AES_128_CBC_SHA-1 |
| C00A | AES_256_CBC_SHA | | C036 | AES_128_CBC_SHA-1 |
| C00B | NULL_SHA. PRF not supported. | | C037 | AES_128_CBC_SHA-256 |
| C00C | Reserved. | | C038 | AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384 |
| C00D | 3DES_EDE_CBC_SHA | | C039 | NULL_SHA. PRF not supported. |
| C00E | AES_128_CBC_SHA | | C03A | NULL_SHA-256. PRF not supported. |
| C00F | AES_256_CBC_SHA | | C03B | NULL_SHA-384. PRF not supported. |
| | | | | |
| | | | C09C | AEAD_AES_128_CCM_16. |
| | | | C09D | AEAD_AES_256_CCM_16. |
| | | | C09E | AEAD_AES_128_CCM_16. |
| | | | C09F | AEAD_AES_256_CCM_16. |
| C010 | NULL_SHA. PRF not supported. | | C0A0 | AEAD_AES_128_CCM_8. |
| C011 | Reserved. | | C0A1 | AEAD_AES_256_CCM_8. |
| C012 | 3DES_EDE_CBC_SHA | | C0A2 | AEAD_AES_128_CCM_8. |
| C013 | AES_128_CBC_SHA | | C0A3 | AEAD_AES_256_CCM_8. |
| C014 | AES_256_CBC_SHA | | C0A4 | AEAD_AES_128_CCM_16. |
| C015 | NULL_SHA. PRF not supported. | | C0A5 | AEAD_AES_256_CCM_16. |
| C016 | Reserved. | | C0A6 | AEAD_AES_128_CCM_16. |
| C017 | 3DES_EDE_CBC_SHA | | C0A7 | AEAD_AES_256_CCM_16. |
| C018 | AES_128_CBC_SHA | | C0A8 | AEAD_AES_128_CCM_8. |
| C019 | AES_256_CBC_SHA | | C0A9 | AEAD_AES_256_CCM_8. |
| C01A | 3DES_EDE_CBC_SHA | | C0AA | AEAD_AES_128_CCM_8. |
| C01B | 3DES_EDE_CBC_SHA | | C0AB | AEAD_AES_256_CCM_8. |
| C01C | 3DES_EDE_CBC_SHA | | | |
| C01D | AES_128_CBC_SHA | | | |

*Table continues on the next page...*

## Table 7-55. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs) (continued)

| PROT INFO [15:0] (hex) | Description | | PROT INFO [15:0] (hex) | Description |
|---|---|---|---|---|
| C01E | AES_128_CBC_SHA | | | |
| C01F | AES_128_CBC_SHA | | | |
| | | | | |
| FF22 | NULL_SHA-224. PRF not supported. | | | |
| FF23 | 3DES_EDE_CBC_MD5 | | FF83 | AES_128_CBC_SHA-384<br>PRF (if TLS 1.2): SHA-384 |
| | | | FF84 | AES_128_CBC_SHA-224 (not valid for PRF) |
| FF30 | 3DES_EDE_CBC_SHA | | FF85 | AES_128_CBC_SHA-512 (not valid for PRF) |
| | | | FF86 | AES_128_CBC_SHA-256 |
| FF33 | 3DES_EDE_CBC_SHA-384; PRF (if TLS 1.2): SHA-384 | | | |
| FF34 | 3DES_EDE_CBC_SHA-224 (not valid for PRF) | | FF90 | AES_192_CBC_SHA |
| FF35 | 3DES_EDE_CBC_SHA-512 (not valid for PRF) | | | |
| FF36 | 3DES_EDE_CBC_SHA-256 | | FF93 | AES_192_CBC_SHA-384; PRF (if TLS 1.2): SHA-384 |
| | | | FF94 | AES_192_CBC_SHA-224 (not valid for PRF) |
| FF51 | NULL_SHA-512. PRF not supported. | | FF95 | AES_192_CBC_SHA-512 (not valid for PRF) |
| | | | FF96 | AES_192_CBC_SHA-256 |
| FF60 | AES_256_CBC_SHA | | | |
| | | | | |
| FF63 | AES_256_CBC_SHA-384; PRF (if TLS 1.2): SHA-384 | | | |
| FF64 | AES_256_CBC_SHA-224 (not valid for PRF) | | | |
| FF65 | AES_256_CBC_SHA-512 (not valid for PRF) | | | |
| FF66 | AES_256_CBC_SHA-256 | | | |
| | | | | |
| FF70 | AES_128_CTR_SHA. Not supported for PRF. | | FFC0 | AEAD_AES_128_CCM_8. |
| FF71 | AES_192_CTR_SHA. Not supported for PRF. | | FFC1 | AEAD_AES_256_CCM_8. |
| FF72 | AES_256_CTR_SHA. Not supported for PRF. | | FFC2 | AEAD_AES_128_CCM_16. |
| FF73 | AES_128_CTR_SHA-224. Not supported for PRF. | | FFC3 | AEAD_AES_256_CCM_16. |
| FF74 | AES_192_CTR_SHA-224. Not supported for PRF. | | FFC4 | AEAD_AES_128_CCM_8. PRF SHA-384. |

*Table continues on the next page...*

### Table 7-55. PROTINFO definition when used with SSL/TLS/DTLS protocol family (including PRFs) (continued)

| PROT INFO [15:0] (hex) | Description | | PROT INFO [15:0] (hex) | Description |
|---|---|---|---|---|
| FF75 | AES_256_CTR_SHA-224. Not supported for PRF. | | FFC5 | AEAD_AES_256_CCM_8. PRF SHA-384. |
| FF76 | AES_128_CTR_SHA-256. Not supported for PRF. | | FFC6 | AEAD_AES_128_CCM_16. PRF SHA-384. |
| FF77 | AES_192_CTR_SHA-256. Not supported for PRF. | | FFC7 | AEAD_AES_256_CCM_16. PRF SHA-384. |
| FF78 | AES_256_CTR_SHA-256. Not supported for PRF. | | FFC8 | AEAD_AES_128_CCM_8. Not supported for PRF. |
| FF79 | AES_128_CTR_SHA-384. Not supported for PRF. | | FFC9 | AEAD_AES_256_CCM_8. Not supported for PRF. |
| FF7A | AES_192_CTR_SHA-384. Not supported for PRF. | | FFCA | AEAD_AES_128_CCM_16. Not supported for PRF. |
| FF7B | AES_256_CTR_SHA-384. Not supported for PRF. | | FFCB | AEAD_AES_256_CCM_16. Not supported for PRF. |
| FF7C | AES_128_CTR_SHA-512. Not supported for PRF. | | | |
| FF7D | AES_192_CTR_SHA-512. Not supported for PRF. | | | |
| FF7E | AES_256_CTR_SHA-512. Not supported for PRF. | | | |
| | | | | |
| | | | FFFE | (SSL, TLS 1.0, TLS 1.1 PRF only):<br>master secret generation using SHA-1 and MD5<br>(TLS 1.2 PRF only): master secret generation using SHA-384 |
| FF80 | AES_128_CBC_SHA | | FFFF | (SSL, TLS 1.0, TLS 1.1 PRF only):<br>master secret generation using SHA-1 and MD5<br>(TLS 1.2 PRF only): master secret generation using SHA-256 |
| All other values reserved. Not all codes permitted with all members of TLS family | | | | |

### Table 7-56. PROTINFO definition when used with IKE PRF and IKE PRF+ protocols

| PROTINFO<br>[15:0] (hex) | Description |
|---|---|
| 0100 | PRF_HMAC_MD5 |
| 0200 | PRF_HMAC_SHA1 |
| 0400 | PRF_AES128_CBC |
| 0500 | PRF_HMAC_SHA2_256 |
| 0600 | PRF_HMAC_SHA2_384 |

*Table continues on the next page...*

### Table 7-56. PROTINFO definition when used with IKE PRF and IKE PRF+ protocols (continued)

| PROTINFO<br>[15:0] (hex) | Description |
|---|---|
| 0700 | PRF_HMAC_SHA2_512 |
| 0800 | PRF_AES128_CMAC |
| All other values reserved. | |

### Table 7-57. PROTINFO definition when used with 3G and LTE (PROTID=43h) protocols

| PROTINFO<br>[15:0] (hex) | Description |
|---|---|
| 0710 | 3G Double CRC with 7-bit and 16-bit CRCs |
| 0B10 | 3G Double CRC with 11-bit and 16-bit CRCs |
| 0000 | 3G RLC with Null encryption |
| 0001 | 3G RLC with Kasumi encryption |
| 0002 | 3G RLC with SNOW3G encryption |
| 0000 | LTE PDCP with Null encryption and authentication |
| 0001 | LTE PDCP with SNOW3G encryption and authentication |
| 0002 | LTE PDCP with AES encryption and authentication |
| All other values reserved. | |

### Table 7-58. PROTINFO definition when used with derived key protocol (DKP) for HMACs

| PROTINFO | Description |
|---|---|
| PROTINFO[15:14]<br>SRC | Input Source Control<br>00 - IMM - negotiated key is in words immediately following the DKP Operation Command. This option can only be used with an Immediate Output Destination (OD=00).<br>01 - SEQ - negotiated key is found in the input frame as defined by the SEQ IN PTR command.<br>10 - PTR - the input key is referenced by the address found immediately following the DKP Operation Command.<br>11 - SGF - the input key is distributed amongst different memory locations as indicated by the Scatter/Gather Table address found immediately following the DKP Operation Command. |
| PROTINFO[13:12]<br>DST | Output Destination Control<br>00 - IMM - resulting derived HMAC "split key" will be written back to the descriptor, immediately after the KEY command written to the descriptor, consuming as many words as required. The contents of those words will be overwritten and will not be preserved. The length of the resulting derived HMAC key is twice the underlying hash context length. See Table 10-32<br>Note that IMM is not restricted when used as an Output Destination as it is when used as an Input Source. |

*Table continues on the next page...*

## Table 7-58. PROTINFO definition when used with derived key protocol (DKP) for HMACs (continued)

| PROTINFO | Description |
|---|---|
| | 01 - SEQ - the resulting derived HMAC "split key" will be written to the output frame as defined by the SEQ OUT PTR command. Note that SEQ is a valid Output Destination only when SEQ is provided as an Input Source.<br>10 - PTR - the resulting derived HMAC "split key" will be written back to the memory location specified by the address found immediately after the DKP Operation Command. This option is not valid with Input Source options IMM or SGF.<br>11 - SGF - the resulting derived HMAC "split key" will be written back to memory per the scatter/gather table found at the address immediately following the DKP operation command. This option is not valid with Input Source options IMM or PTR. |
| PROTINFO[11:0]<br>LEN | Length of the negotiated key provided to the DKP Operation command in bytes. |

## Table 7-59. PROTINFO format when used with Blob Operations

| 15-10 | 9 | 8 | 7-4 | 3 | 2 | 1-0 |
|---|---|---|---|---|---|---|
| Reserved | TK | EKT | K2KR | Reserved | Black_Key | Blob_ Format |

## Table 7-60. PROTINFO field descriptions when used with Blob Operations

| Field | Description |
|---|---|
| 15-10 | Reserved. |
| 9<br>TK | Trusted Key<br>Used only for trusted descriptors with black blob encapsulation/decapsulation. Ignored otherwise.<br>0 Use the JDKEK when encrypting or decrypting black keys.<br>1 Use the TDKEK when encrypting or decrypting black keys. |
| 8<br>EKT | Encrypted Key Type<br>Used only for black blob encapsulation/decapsulation. Ignored otherwise. Specifies the encryption/decryption mode for black keys. Also used when deriving the blob key encryption key. Consequently, the same EKT setting must be used when decapsulating a black blob as was used when encapsulating that black blob. This prevents a black key being converted between AES-ECB and AES-CCM by encapsulating it as a blob and then decapsulating the blob in the other encryption mode.<br>0 Use AES-ECB mode when encrypting/decrypting black keys.<br>1 Use AES-CCM mode when encrypting/decrypting black keys. |
| 7-4<br>K2KR | Key to Key Register<br>Specifies the destination for the result of black blob decapsulation. Ignored otherwise. Black blob encapsulation always uses a source from memory. The source and destination for red blob encapsulation and decapsulation is always memory. (See Blob types differentiated by content)<br>0000 Memory<br>0001 Class 1 key register<br>0011 Class 2 key register |

*Table continues on the next page...*

**Table 7-60. PROTINFO field descriptions when used with Blob Operations (continued)**

| Field | Description |
|---|---|
| | 0111 Class 2 key register (split key)<br>1001 PKHA E RAM<br>All other values are reserved. |
| 3 | Reserved. |
| 2<br>Black_key | 0 Red Blob. The data encapsulated into the blob or decapsulated from the blob is treated as plaintext.<br>1 Black Blob. The data encapsulated into the blob or decapsulated from the blob is treated as a black key encrypted with the appropriate KEK (JDKEK or TDKEK). For blob encapsulation operations, the input data is first decrypted using the appropriate KEK and then encrypted using the blob key. For blob decapsulation operations, the data portion of the blob is decrypted using the blob key. If the resulting plaintext is to be written into memory rather than into a key register, the plaintext is encrypted using the appropriate KEK. |
| 1-0<br>Blob_ Format | The format of the blob.<br>00 Normal Blob. The output is composed of the encrypted blob key, the encrypted data, and MAC tag.<br>01 Reserved<br>10 Master Key Verification Blob. This blob type is intended for verifying the master key and the key derivation. The master key is used for key derivation in the Trusted and Secure security states. The test key is used in the Nonsecure state. Only the derived blob key encryption key is output. Note that the Blob_Format value is an input to the BKEK derivation, which ensures that the BKEK value that is exposed in a master key verification blob is different than the BKEK value used for any other blob format. Furthermore, the use of SHA-256 in BKEK derivation ensures that the BKEK values used for other blob formats cannot be learned by analyzing the BKEK values used for master key verification blobs.<br>11 Test Blob. The non-volatile test key is used for key derivation. The output is composed of the derived blob key encryption key, the actual blob key, the encrypted blob key, the encrypted data, and MAC tag. Test blobs can be exported or imported only when SEC is in non-secure mode. |

Table 7-61 shows the format of the PROTINFO field for discrete log public key protocols, including:

- Key pair generation (see Discrete-log key-pair generation)
- DSA sign (see Generating DSA and ECDSA signatures)
- DSA verify (see Verifying DSA and ECDSA signatures)
- Diffie-Hellman (see Using the Diffie_Hellman function).

Table 7-62. describes the bit values of this field.

**Table 7-61. PROTINFO format when used with Discrete Log Protocol**

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Format for Sign function | Reserved | | | SIGN_NO_TEQ | MES_REP | | HASH | | | SIGN_2ND_HALF_ONLY | SIGN_1ST_HALF_ONLY | EXT_PRI | TEST | ENC_PRI | ECC/DL | F2M/Fp |
| Format for MPSign functions | Reserved | | | SIGN_ | MES_REP | | HASH | | | Reserved | Reserved | EXT_PRI | TEST | ENC_PRI | ECC/DL | F2M/Fp |

*Table continues on the next page...*

### Table 7-61. PROTINFO format when used with Discrete Log Protocol (continued)

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | NO_TEQ | | | | | | | | | | |
| Format for Verify function | Reserved | | MES_REP | HASH | | Reserved | | | | | ECC/DL | F2M/Fp |
| Format for Keypair Generation functions | Reserved | | | | KPG_IETF_DH | EKT_Z | ENC_Z | EXT_PRI | KPG_NO_TEQ | ENC_PRI | ECC/DL | F2M/Fp |
| Format for MP Keypair Generation functions | Reserved | | | | | | | | KPG_NO_TEQ | Reserved | | |

### Table 7-62. PROTINFO field descriptions when used with Discrete Log Protocol

| Field | Description |
|---|---|
| MES_REP | Build the message representative from the message<br>0 Use the message representative that is pointed to by the f field in the protocol data block.<br>1 Calculate the message representative from the message (using a SEQ IN PTR command), and the hash function specified by the HASH field. The message representative is calculated using the equivalent of EMSA1 (IEEE-1363). |
| SIGN_NO_TEQ | For Signature Generation (SIGN) protocol and MPSign: disable Timing Equalization during SIGN.<br>0 Run SIGN using normal Timing Equalization protection.<br>1 Run SIGN with NO Timing Equalization protection. |
| MES_REP | For Signature Generation (SIGN) and Verification (VERIFY) protocols, this field indicates the format of the message.<br>00 : F input is a message representative.<br>01 : Calculate the message representative from the message (using a SEQ IN PTR command), and the hash function specified by the HASH field. The message representative is calculated using the equivalent of EMSA1 (IEEE-1363).<br>10 : F input is a hashed message, with length specified in the PDB. Protocol will format the message as required.<br>11 : Reserved. |
| HASH | Hash function used to calculate a message representative from a message; valid when MES_REP=01.<br>000 MD5<br>001 SHA-1<br>010 SHA-224<br>011 SHA-256<br>100 SHA-384<br>101 SHA-512 |

*Table continues on the next page...*

## Table 7-62. PROTINFO field descriptions when used with Discrete Log Protocol (continued)

| Field | Description |
|---|---|
| KPG_IETF_DH | For KPG, this bit enables running IETF_style DH<br>0 No IETF-style DH.<br>1 Run KPG with IETF-style Diffie-Hellman. |
| PRI_VERIFY_NO_TEQ | For Signature Verification with Private Key (PRI_VERIFY) protocol only. Disable Timing Equalization protection.<br>0 Run PRI_VERIFY with Timing Equalization protection enabled.<br>1 Run PRI_VERIFY with Timing Equalization protection disabled. |
| SIGN_2ND_HALF_ONLY | For Signature Generation (SIGN) protocol only; otherwise reserved. Run 2nd half (signature "d" generation) only.<br>0 Run full SIGN or 1st half, depending on SIGN_1ST_HALF_ONLY setting.<br>1 Run 2nd half of SIGN only, generating 'd' result. Requires SIGN_1ST_HALF_ONLY = 0. |
| SIGN_1ST_HALF_ONLY | For Signature Generation (SIGN) protocol only; otherwise reserved. Run 1st half (signature "c" generation) only.<br>0 Run full SIGN or 2nd half, depending on SIGN_2ND_HALF_ONLY setting.<br>1 Run 1st half of SIGN only, generating 'c' result. Requires SIGN_2ND_HALF_ONLY = 0. |
| EKT_Z | if ENC_Z=1, Key Encryption type (Used only with DH; otherwise reserved.)<br>0 Secret output is encrypted with AES-ECB mode.<br>1 Secret output is encrypted with AES-CCM mode. |
| ENC_Z | Encrypt the DH shared secret (Used only with DH; otherwise reserved.)<br>0 The DH output is public and is unencrypted.<br>1 The DH output is secret and encrypted. |
| EXT_PRI | if ENC_PRI=1, Encrypted key type for private key<br>0 Private key is encrypted with AES-ECB mode.<br>1 Private key is encrypted with AES-CCM mode. |
| KPG_NO_TEQ | KPG_NO_TEQ For KPG, MPPrivK and MPPubK.<br>0 Key Pair Generation runs with Timing Equalization protection.<br>1 Kep Pair Generation runs with Timing Equalization disabled. |
| TEST | TEST<br>0 Signature generation protects the per message secret.<br>1 Signature generation outputs the per message secret, to aid in testing and verification. This is not allowed in trusted or secure states. |
| ENC_PRI | Encrypted private key |

*Table continues on the next page...*

## Table 7-62. PROTINFO field descriptions when used with Discrete Log Protocol (continued)

| Field | Description |
|---|---|
| | 0 Private key is not encrypted. ENC_PRI must be 0 if SIGN_2ND_HALF_ONLY=1.<br>1 Private key must be decrypted before use (see KEY command for further information.). For Key Generation, this causes the Private Key to be encrypted. |
| ECC/DL | Public Key operation type<br>0 DL: Discrete Log<br>1 ECC: Elliptic Curve Cryptography |
| F2M/Fp | Finite Field type<br>0 Fp: Prime Field<br>1 F2M: Binary field |

Table 7-63 shows the format of the PROTINFO field for the RSA encrypt protocol. Table 7-64 describes the bit values.

## Table 7-63. PROTINFO format when used with RSA Encrypt Protocol

| 15-13 | 12 | 1-7 | 6-4 | 3-2 | 1-0 |
|---|---|---|---|---|---|
| Reserved | FMT | Reserved | fff | Reserved | OP |

## Table 7-64. PROTINFO field descriptions when used with RSA Encrypt Protocol

| Field | Description |
|---|---|
| 15-13 | Reserved. |
| 12<br>FMT | Format of data<br>0 No formatting<br>1 EME-PKCS1-v1_5 encryption encoding function |
| 11-7 | Reserved. |
| 6-4<br>fff | Encryption type for f<br>000b f is not encrypted (This is the only value permitted when OP = 00b).<br>001b f is to be encrypted with the JDKEK using ECB mode.<br>011b f is to be encrypted with the JDKEK using CCM mode.<br>101b f is to be encrypted with the TDKEK using ECB mode.<br>111b f is to be encrypted with the TDKEK using CCM mode.<br>All other values are reserved. |
| 3-2 | Reserved. |
| 1-0<br>OP | Operation.<br>00b Public Key n, e, f in - f is a user-supplied value (fff must be 000b)<br>01b Public Key n, e, f out - f is a random value (f can be encrypted on output, fff can be any non-reserved value)<br>All other values are reserved. |

Table 7-65 shows the format of the PROTINFO field for the RSA Decrypt Protocol. Table 7-66 describes the bit values.

Table 7-65. PROTINFO format when used with RSA Decrypt Protocol

| 15-13 | 12 | 11 | 10-8 | 7 | 6-4 | 3-2 | 1-0 |
|---|---|---|---|---|---|---|---|
| Reserved | FMT | Reserved | ppp | Reserved | fff | Reserved | Key Form |

Table 7-66. PROTINFO field descriptions when used with RSA Decrypt Protocol

| Field | Description |
|---|---|
| 15-13 | Reserved. |
| 12<br>FMT | Format of data<br>0 No formatting<br>1 EME-PKCS1-v1_5 encryption decoding function |
| 11 | Reserved. |
| 10-8<br>ppp | Type of private key encryption<br>000 private key is not encrypted<br>001b private key components are each encrypted with the JDKEK using ECB mode<br>011b private key components are each encrypted with the JDKEK using CCM mode<br>101b private key components are each encrypted with the TDKEK using ECB mode<br>111b private key components are each encrypted with the TDKEK using CCM mode<br>All other values are reserved. |
| 7 | Reserved. |
| 6-4<br>fff | Type of encryption for f.<br>000b f is not to be encrypted<br>001b f is to be encrypted with the JDKEK using ECB mode<br>011b f is to be encrypted with the JDKEK using CCM mode<br>101b f is to be encrypted with the TDKEK using ECB mode<br>111b f is to be encrypted with the TDKEK using CCM mode<br>All other values are reserved. |
| | |
| 3 | No TEQ option. Set to 1 to enable no-TEQ. |
| 2 | Reserved. |
| 1-0<br>Key Form | Form of the Private Key<br>00b Private Key input in the form #1: n, d<br>01b Private Key input in the form #2: p, q, d<br>10b Private Key input in the form #3: p, q, dp, dq, c<br>All other values are reserved. |

Table 7-67. PROTINFO format when used with RSA Finish KeyGen

| 15-8 | 7 | 6 | 5 | 4 | 3 | 2 | 1-0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | ENC_OUT | Reserved | EKT | SKIP_D | SKIP_PQ | FUNCTION |

Table 7-68. PROTINFO field descriptions when used with RSA Finish Keygen Protocol

| Field | Description |
|---|---|
| 15-8 | Reserved. |
| 7 | Reserved. |
| 6<br>ENC_OUT | Encrypt Outputs<br>0 Do not encrypt generated private key components<br>1 Encrypt generated private key (ECB mode, unless EKT=1)<br>(Note that *n* and *d* size are not encrypted.) |
| 5<br>Reserved | Reserved. Must be 0. |
| 4<br>EKT | Encrypted Key Type<br>0 Do not use CCM-encryption<br>1 CCM-encrypt private key components (valid only if PROTOCOL command's ENC bit is 1) |
| 3<br>SKIP_D | Skip length check of *d*<br>0 Check that *d* is at least one bit longer than 1/2 of the bit length of *n*.<br>1 Skip length check of *d*. |
| 2<br>SKIP_PQ | Skip check of upper 100 bits of *p* and *q*<br>0 Check upper 100 bits of *p* and *q* to see whether \|*p*-*q*\| is too small<br>1 Do not check upper 100 bits of *p* and *q* |
| 1-0<br>FUNCTION | Function<br>00 Compute all key components listed in Key Form, including *d*<br>01 Compute all key components listed in Key Form except *d*, which is an input<br>10 From *p*, *q*, *e*, compute *n*, *d* and *d* size.<br>11 Reserved |

## 7.18 PKHA OPERATION command

If OPTYPE = 001 (PKHA), the fields are as shown in Table 7-69. This OPTYPE is used to perform public key operations in the public key hardware accelerator (PKHA). All data for a PKHA operation must already be in place before the function will begin executing. Therefore, this operation does not start until all data transactions have completed and the input data FIFO is empty.

The format of the PKHA MODE field depends on which of the four types of PKHA functions the OPERATION command specifies:

- Clear memory
- Modular arithmetic
- Elliptic curve
- Copy memory

A detailed description of the PKHA MODE fields is found in Table 7-70. The OPERATION command does not complete until the PKHA is done.

When the PKHA operation completes without error, DECO clears the DONE flag and the Mode Register so another operation can be specified.

**Table 7-69. PKHA OPERATION command format**

| 31-27 | 26-24 | 23-20 | 19-16 |
|---|---|---|---|
| CTYPE = 10000 | OPTYPE = 001 | ALG = 1000 | PKHA_MODE_MS |
| 15-12 | 11-0 | | |
| Reserved | PKHA_MODE_LS | | |

**Table 7-70. PKHA OPERATION command field descriptions**

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>If CTYPE=10000b : OPERATION command; (ALGORITHM OPERATION or PKHA OPERATION or PROTOCOL OPERATION, as determined by the OPTYPE field) |
| 26-24<br>OPTYPE | Operation Type<br>If OPTYPE = 001b : PKHA OPERATION: The PKHA_MODE fields are interpreted as shown in the following tables.<br>If OPTYPE = 010b or 100b : ALGORITHM OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in ALGORITHM OPERATION command.<br>If OPTYPE = 000b, 011b or 111b : PROTOCOL OPERATION; The ALG, AAI, AS, ICV, and ENC fields are interpreted as shown in PROTOCOL OPERATION command. |
| 23-20<br>ALG | Algorithm<br>Set ALG=1000b. All other values are reserved. |
| 19-16<br>PKHA_MODE_MS | PKHA Mode<br>This field contains the value that will be loaded into the upper 4 bits of the PKHA Mode register. Its content depends on which of the four types of PKHA functions, clear memory, modular arithmetic function, or copy memory, is specified in the Function field (bits 5-0). The formats for these four types of functions are shown in the following sections: Clear Memory (CLEAR_MEMORY) function, PKHA OPERATION: Arithmetic Functions, PKHA OPERATION: Elliptic Curve Functions and PKHA OPERATION: copy memory functions. |
| 15-12 | Reserved |
| 11-0<br>PKHA_MODE_LS | PKHA Mode |

*Table continues on the next page...*

**Table 7-70. PKHA OPERATION command field descriptions (continued)**

| Field | Description |
|---|---|
| | This field contains the value that will be loaded into the lowest 12 bits of the PKHA Mode register. The least-significant six bits of this field is interpreted as a Function field, as shown in the row below. The format of the PKHA_MODE_MS field and the other bits of the PKHA_MODE_LS field depend on the PKHA function specified in the Function field: clear memory, modular arithmetic function, or copy memory. The formats for these four types of functions are shown in the following sections: Clear Memory (CLEAR_MEMORY) function, PKHA OPERATION: Arithmetic Functions, PKHA OPERATION: Elliptic Curve Functions, and PKHA OPERATION: copy memory functions. |
| 5-0<br>*Function* | PKHA function to be performed. *(Note that the function is encoded in the least-significant six bits of the PKHA_MODE_LS field.)*<br>If Function=000001b : Clear Memory. (See Clear Memory (CLEAR_MEMORY) function)<br>If Function=010000b or 010001b : Copy Memory. (See PKHA OPERATION: copy memory functions)<br>If Function=001001b, 001010b, 001011b, or 011100b : Elliptic Curve function. (See PKHA OPERATION: Elliptic Curve Functions)<br>If Function=000010b - 001111b or 010110b - 011111b : Modular Arithmetic function. (See PKHA OPERATION: Arithmetic Functions)<br>All other values of the Function field are reserved. |

## 7.18.1 PKHA OPERATION: clear memory function

**Table 7-71. PKHA Mode register format for clear memory function**

| 19 | 18 | 17 | 16 | | 11-10 | 9 | 8 | 7 | 6 | 5-0 |
|---|---|---|---|---|---|---|---|---|---|---|
| Aram | Bram | Eram | Nram | . | Reserved | Q3 | Q2 | Q1 | Q0 | Function |
| *PKHA_MODE_MS* | | | | . . | *PKHA_MODE_LS* | | | | | |

If the Function field in PKHA MODE specifies the clear memory function, PKHA expects to be in the format shown in Table 7-71. The PKHA RAMs to be cleared may be selected in any combination. Selecting one or more Quadrants for clearing will cause only the specified quadrants (of the specified RAMs) to be cleared. If no Quadrants are selected, then the whole RAM will be cleared.

**Table 7-72. PKHA mode register field descriptions for clear memory function**

| Bits | Description |
|---|---|
| 19 | Aram<br>This bit selects the A RAM for zeroization.<br>0: A not selected<br>1: A selected. |
| 18 | Bram<br>This bit selects the B RAM for zeroization. |

*Table continues on the next page...*

### Table 7-72. PKHA mode register field descriptions for clear memory function (continued)

| Bits | Description |
|---|---|
| | 0: B not selected<br>1: B selected |
| 17 | Eram<br>This bit selects the E RAM for zeroization.<br>0: E not selected<br>1: E selected |
| 16 | Nram<br>This bit selects the N RAM for zeroization.<br>0: N not selected<br>1: N selected |
| 11-10 | Reserved |
| 9 | Quadrant 3<br>This bit selects the Quadrant 3 RAM for zeroization.<br>0: not selected<br>1: selected. Clearing will be only specified quadrant(s). Not valid if E RAM is selected. |
| 8 | Quadrant 2<br>This bit selects the Quadrant 2 RAM for zeroization.<br>0: not selected<br>1: selected. Clearing will be only specified quadrant(s). Not valid if E RAM is selected. |
| 7 | Quadrant 1<br>This bit selects the Quadrant 1 RAM for zeroization.<br>0: not selected<br>1: selected. Clearing will be only specified quadrant(s). Not valid is E RAM is selected. |
| 6 | Quadrant 0<br>This bit selects the Quadrant 0 RAM for zeroization.<br>0: not selected<br>1: selected. Clearing will be only specified quadrant(s). Not valid if E RAM is selected. |
| 5-0 | Function<br>The Function value for clearmemory is 000001. |

## 7.18.2 PKHA OPERATION: Arithmetic Functions

### Table 7-73. PKHA Mode Register Format for Arithmetic Functions

<table>
<tr><td>19</td><td>18</td><td>17</td><td>16-12</td><td>11</td><td>10</td><td>9-8</td><td>7-6</td><td>5-0</td></tr>
<tr><td>inM</td><td>outM</td><td>F2m</td><td>Reserved</td><td>Reserved</td><td>Teq</td><td>OutSel</td><td>Reserved</td><td>Function</td></tr>
<tr><td colspan="4"><i>PKHA_MODE_MS</i></td><td colspan="5"><i>PKHA_MODE_LS</i></td></tr>
</table>

## Table 7-74. PKHA Mode register, format for arithmetic operation

| Bits | Description |
|---|---|
| 19<br>inM | Inputs in Montgomery form. Indicates whether the inputs are in Montgomery form.<br>If inM=0 : Normal value representation<br>If inM=1 : Montgomery form. (Not valid for all functions.) |
| 18<br>outM | Outputs in Montgomery form. Indicates whether the outputs are to be left in Montgomery form or converted to normal values.<br>If outM=0 : Normal value representation<br>If outM=1 : Montgomery form. (Not valid for all functions.) |
| 17<br>F2m | F2m. Indicates whether to use integer or binary polynomial arithmetic in executing the function.<br>If F2m=0 : Integer<br>If F2m=1 : Binary polynomial. (Not valid for all functions.) |
| 16-12 | Reserved |
| 11 | Reserved |
| 10<br>Teq | Timing Equalized. Indicates that a timing equalized version of the function should be executed.<br>If Teq=0 : No timing equalization<br>If Teq=1 : Timing equalization. (Not valid for all functions.) |
| 9-8<br>OutSel | Output destination select. Indicates which memory should contain the output of the selected function.<br>If OutSel=00b : B<br>If OutSel=01b : A<br>If OutSel=10b : Reserved<br>If OutSel=11b : Reserved |
| 7-6 | Reserved |
| 5-0<br>Function | Function. Indicates which arithmetic function to execute.<br>If Function=000010b : Modular Addition (A + B) mod N<br>If Function=000011b : Modular Subtraction 1 (A - B) mod N<br>If Function=000100b : Modular Subtraction 2 (B - A) mod N<br>If Function=000101b : Modular Multiplication (A x B) mod N<br>If Function=000110b : Modular Exponentiation $A^E$ mod N<br>If Function=000111b : Modular Reduction A mod N<br>If Function=001000b : Modular Inversion $A^{-1}$ mod N<br>If Function=001100b : Montgomery Radix Constant $R^2$ mod N<br>If Function=001110 : Greatest Common Divisor GCD(A,N)-see note below<br>If Function=001111 : Miller-Rabin Primality Test -see note below<br>If Function=010110b : Modular Simultaneous Exponentiation $A0^E$ * $A2^B$ mod N<br>If Function=011000b : Modular Double A (A + A) mod N<br>If Function=011001b : Modular Double B (B + B) mod N<br>If Function=011010b : Modular Square A (A x A) mod N<br>If Function=011011b : Modular Cube A (A x A x A) mod N |

### Table 7-74. PKHA Mode register, format for arithmetic operation

| Bits | Description |
|---|---|
| | If Function=011101b : Shift Right A<br>If Function=011110b : Compare A B<br>If Function=011111b : Evaluate A<br>All other values for this field are currently reserved or are Table 7-71,Table 7-78, or Table 7-82.<br>**NOTE:** When using the GCD function or any ECC function, a divide-by-zero error occurs if the value of the most significant digit of N is all zeros.<br>**NOTE:** When using the Miller-Rabin primality test function, if the most-significant digit of N is all zeros, the result is composite regardless of the value of N. |

## NOTE

Note that the arithmetic functions with outputs going to the A RAM are identical to those with outputs going to the B RAM. The only difference is the output destination.

### Table 7-75. List of mode values for PKHA Integer Arithmetic Functions

| Function name | Brief description | Output reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| MOD_ADD | Integer Modular Addition | B | 0 | 00002 | Integer Modular Addition (MOD_ADD) function |
| | | A | 0 | 00102 | |
| MOD_SUB_1 | Integer modular subtraction (A - B) | B | 0 | 00003 | Integer Modular Subtraction (MOD_SUB_1) function |
| | | A | 0 | 00103 | |
| MOD_SUB_2 | Integer modular subtraction (B - A) | B | 0 | 00004 | Integer Modular Subtraction (MOD_SUB_2) function |
| | | A | 0 | 00104 | |
| MOD_MUL<br>MOD_MUL_TEQ | Integer modular multiplication | B | 0 | 00005 | Integer Modular Multiplication (MOD_MUL) |
| | | A | 0 | 00105 | |
| | Timing equalized version | B | 1 | 00405 | |
| | | A | 1 | 00505 | |
| MOD_MUL_IM<br>MOD_MUL_IM_TEQ | Integer Modular Multiplication with Montgomery Inputs | B | 0 | 80005 | Integer Modular Multiplication with Montgomery Inputs (MOD_MUL_IM) |
| | | A | 0 | 80105 | |
| | Timing equalized version | B | 1 | 80405 | |
| | | A | 1 | 80505 | |
| MOD_MUL_IM_OM<br>MOD_MUL_IM_OM_TEQ | Integer Modular Multiplication with Montgomery Inputs and Outputs | B | 0 | C0005 | Integer Modular Multiplication with Montgomery Inputs and Outputs (MOD_MUL_IM_OM) Function |
| | | A | 0 | C0105 | |
| | Timinq equalized version | B | 1 | C0405 | |
| | | A | 1 | C0505 | |

*Table continues on the next page...*

## Table 7-75. List of mode values for PKHA Integer Arithmetic Functions (continued)

| Function name | Brief description | Output reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| MOD_EXP<br>MOD_EXP_TEQ | Integer Modular Exponentiation | B | 0 | 00006 | Integer Modular Exponentiation (MOD_EXP and MOD_EXP_TEQ) |
| | | A | 0 | 00106 | |
| | Timing equalized version | B | 1 | 00406 | |
| | | A | 1 | 00506 | |
| MOD_EXP_IM<br>MOD_EXP_IM_TEQ | Integer Modular Exponentiation with Montgomery Inputs | B | 0 | 80006 | Integer Modular Exponentiation, Montgomery Input (MOD_EXP_IM and MOD_EXP_IM_TEQ) Function |
| | | A | 0 | 80106 | |
| | Timing equalized version | B | 1 | 80406 | |
| | | A | 1 | 80506 | |
| MOD_AMODN | Integer Modular Reduction | B | 0 | 00007 | Integer Modulo Reduction (MOD_AMODN) |
| | | A | 0 | 00107 | |
| MOD_INV | Integer Modular Inversion | B | 0 | 00008 | Integer Modular Inversion (MOD_INV) |
| | | A | 0 | 00108 | |
| MOD_R2 | Integer $R^2$ mod N | B | 0 | 0000C | Integer Montgomery Factor Computation (MOD_R2) |
| | | A | 0 | 0010C | |
| MOD_GCD | Integer Greatest Common Divisor | B | 0 | 0000E | Integer Greatest Common Divisor (MOD_GCD) |
| | | A | 0 | 0010E | |
| PRIME_TEST | Miller_Rabin primality test | B | 0 | 0000F | Miller_Rabin Primality Test (PRIME_TEST) |
| | | A | 0 | 0010F | |
| MOD_SML_EXP | Integer Modular Simultaneous Exponentiation | B | 0 | 00016 | Integer Simultaneous Modular Exponentiation (MOD_SML_EXP) |
| | | A | 0 | 00116 | |
| MOD_SQRT | Integer Modular Square Root | B | 0 | 00017 | Integer Modular Square Root (MOD_SQRT) |
| | | A | 0 | 20017 | |
| MOD_DBL_A | Integer Modular Double A | B | 0 | 20018 | |
| | | A | 0 | 20118 | |
| MOD_DBL_B | Integer Modular Double B | B | 0 | 20019 | |
| | | A | 0 | 20119 | |
| MOD_SQR<br>MOD_SQR_TEQ | Integer Modular Square A | B | 0 | 0001A | Integer Modular Square (MOD_SQR and MOD_SQR_TEQ) |
| | | A | 0 | 0011A | |
| | Timing equalized version | B | 1 | 0041A | |
| | | A | 1 | 0051A | |
| MOD_IM_SQR<br>MOD_IM_SQR_TEQ | Integer Modular Square A. Montgomery Input | B | 0 | 8001A | Integer Modular Square, Montgomery inputs (MOD_SQR_IM and MOD_SQR_IM_TEQ) |
| | | A | 0 | 8011A | |
| | Timing equalized version | B | 1 | 8041A | |
| | | A | 1 | 8051A | |

*Table continues on the next page...*

### Table 7-75. List of mode values for PKHA Integer Arithmetic Functions (continued)

| Function name | Brief description | Output reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| MOD_IM_OM_SQR<br>MOD_IM_OM_SQR_TEQ | Integer Modular Square A, Montgomery Input, Montgomery Output | B | 0 | C001A | Integer Modular Square, Montgomery inputs and outputs (MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ) |
| | | A | 0 | C011A | |
| | Timing equalized version | B | 1 | C041A | |
| | | A | 1 | C051A | |
| MOD_CUBE<br>MOD_CUBE_TEQ | Integer Modular Cube A | B | 0 | 0001B | Integer Modular Cube (MOD_CUBE and MOD_CUBE_TEQ) |
| | | A | 0 | 0011B | |
| | Timing equalized version | B | 1 | 0041B | |
| | | A | 1 | 0051B | |
| MOD_CUBE_IM<br>MOD_CUBE_IM_TEQ | Integer Modular Cube A, Montgomery input | B | 0 | 8001B | Integer Modular Cube, Montgomery input (MOD_CUBE_IM and MOD_CUBE_IM_TEQ) |
| | | A | 0 | 8011B | |
| | Timing equalized version | B | 1 | 8041B | |
| | | A | 1 | 8051B | |
| MOD_CUBE_IM_OM<br>MOD_CUBE_IM_OM_TEQ | Integer Modular Cube A, Montgomery input, Montgomery output | B | 0 | C001B | Integer Modular Cube, Montgomery input and output (MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ) |
| | | A | 0 | C011B | |
| | Timing equalized version | B | 1 | C041B | |
| | | A | 1 | C051B | |

[1] PKHA_MODE_MS concatenated with 0000b concatenated with PKHA_MODE_LS

Arithmetic functions on a binary polynomials (characterestic two) (F2M). All operate in polynomial basis.

### Table 7-76. List of mode values for PKHA Binary Polynomial Arithmetic Functions

| Function name | Brief description | Output reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| F2M_ADD | Binary Polynomial Modular Addition | B | 0 | 20002 | Binary Polynomial ($F_{2^m}$) Addition (F2M_ADD) function |
| | | A | 0 | 20102 | |
| F2M_MUL<br>F2M_MUL_TEQ | Binary Polynomial Modular Multiplication | B | 0 | 20005 | Binary Polynomial ($F_{2^m}$) Modular Multiplication (F2M_MUL) |
| | | A | 0 | 20105 | |
| | Timing equalized version | B | 1 | 20405 | |
| | | A | 1 | 20505 | |

*Table continues on the next page...*

## Table 7-76. List of mode values for PKHA Binary Polynomial Arithmetic Functions (continued)

| Function name | Brief description | Output reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| F2M_MUL_IM<br>F2M_MUL_IM_TEQ | Binary Polynomial Modular Multiplication with Montgomery Inputs | B | 0 | A0005 | Binary Polynomial ($F_{2^m}$) Modular Multiplication with Montgomery Inputs (F2M_MUL_IM) Function |
| | | A | 0 | A0105 | |
| | Timing equalized version" | B | 1 | A0405 | |
| | | A | 1 | A0505 | |
| F2M_MUL_IM_OM<br>F2M_MUL_IM_OM_TEQ | Binary Polynomial Modular Multiplication with Montgomery Inputs and Output | B | 0 | E0005 | Binary Polynomial ($F_{2^m}$) Modular Multiplication with Montgomery Inputs and Outputs (F2M_MUL_IM_OM) Function |
| | | A | 0 | E0105 | |
| | Timing equalized version | B | 1 | E0405 | |
| | | A | 1 | E0505 | |
| F2M_EXP<br>F2M_EXP_TEQ | Binary Polynomial Modular Exponentiation | B | 0 | 20006 | Binary Polynomial ($F_{2^m}$) Modular Exponentiation (F2M_EXP and F2M_EXP_TEQ) |
| | | A | 0 | 20106 | |
| | Timing equalized version | B | 1 | 20406 | |
| | | A | 1 | 20506 | |
| F2M_AMODN | Binary Polynomial Modular Reduction | B | 0 | 20007 | Binary Polynomial ($F_{2^m}$) Modulo Reduction (F2M_AMODN) |
| | | A | 0 | 20107 | |
| F2M_INV | Binary Polynomial Modular Inversion | B | 0 | 20008 | Binary Polynomial ($F_{2^m}$) Modular Inversion (F2M_INV) |
| | | A | 0 | 20108 | |
| F2M_R2 | Binary Polynomial $R^2$ mod n | B | 0 | 2000C | Binary Polynomial ($F_{2^m}$) $R^2$ Mod N (F2M_R2) Function |
| | | A | 0 | 2010C | |
| F2M_GCD | Binary Polynomial Greatest Common Divisor | B | 0 | 2000E | Binary Polynomial ($F_{2^m}$) Greatest Common Divisor (F2M_GCD) Function |
| | | A | 0 | 2010E | |
| F2M_SQR<br>F2M_SQR_TEQ | Binary Polynomial Modular A Square | B | 0 | 2001A | |
| | | A | 0 | 2011A | |
| | Timing equalized version | B | 1 | 2041A | |
| | | A | 1 | 2051A | |
| F2M_IM_SQR<br>F2M_IM_SQR_TEQ | Binary Polynomial Modular A Square. Montgomery input | B | 0 | A001A | |
| | | A | 0 | A011A | |
| | Timing equalized version | B | 1 | A041A | |
| | | A | 1 | A051A | |
| F2M_IM_OM_SQR<br>F2M_IM_OM_SQR_TEQ | Binary Polynomial Modular Square A, Montgomery input, Montgomery output | B | 0 | E001A | |
| | | A | 0 | E011A | |

*Table continues on the next page...*

### Table 7-76. List of mode values for PKHA Binary Polynomial Arithmetic Functions (continued)

| Function name | Brief description | Output reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| | Timing equalized version | B | 1 | E041A | |
| | | A | 1 | E051A | |
| F2M_CUBE<br>F2M_CUBE_TEQ | Binary Polynomial Modular Cube A | B | 0 | 2001B | |
| | | A | 0 | 2011B | |
| | Timing equalized version | B | 1 | 2041B | |
| | | A | 1 | 2051B | |
| F2M_CUBE_IM<br>F2M_CUBE_IM_TEQ | Binary Polynomial Modular Cube A. Montgomery input | B | 0 | A001B | |
| | | A | 0 | A011B | |
| | Timing equalized version | B | 1 | A041B | |
| | | A | 1 | A051B | |
| F2M_CUBE_IM_OM<br>F2M_CUBE_IM_OM_TEQ | Binary Polynomial Modular Cube A, Montgomery input, Montgomery output | B | 0 | E001B | Binary Polynomial ($F_{2^m}$) Modular Cube, Montgomery Input and Output (F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_TEQ) |
| | | A | 0 | E011B | |
| | Timing equalized version | B | 1 | E041B | |
| | | A | 1 | E051B | |
| F2M_SML_EXP | Binary Polynomial Modular Simultaneous Exponentiation | B | 0 | 20016 | Binary Polynomial ($F_{2^m}$) Simultaneous Modular Exponentiation (F2M_SML_EXP) |
| | | A | 0 | 20116 | |

[1] PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

These functions are grouped here because they do not fall into one of the previous categories of PKHA functions.

### Table 7-77. List of mode values for Miscellaneous Functions

| Function name | Brief description | Output reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| Shift Right A | Right Shift | B | 0 | 0001D | Right Shift A (R_SHIFT) function |
| | | A | 0 | 0011D | |
| Compare A B | Comparison | (no output) | 0 | 0001E | Compare A B (COMPARE) function |

*Table continues on the next page...*

Table 7-77. List of mode values for Miscellaneous Functions (continued)

| Function name | Brief description | Output<br>reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| Evaluate A | Compute sizes | B | 0 | 0001F | Evaluate A (EVALUATE) function |
| | | A | 0 | 0011F | |

[1] PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

## 7.18.3 PKHA OPERATION: copy memory functions

Table 7-78. PKHA Mode register, format for copy memory functions

| 19-17 | 16 | 11-10 | 9-8 | 7-6 | 5-0 |
|---|---|---|---|---|---|
| Source Register | Destination Register | | Source<br>Segment | Destination<br>Segment | Function |
| *PKHA_MODE_MS* | | *PKHA_MODE_LS* | | | |

Table 7-79. PKHA Mode register, field descriptions for copy memory functions

| Bits | Description |
|---|---|
| 19-17<br>Source Register | Source Register. Specifies the register to be copied from.<br>If Source Register=000 : A<br>If Source Register=001 : B<br>If Source Register=011 : N<br>All other values are currently reserved. |
| 16<br>Destination Register | Destination Register. Specifies the register to be copied to.<br>If Destination Register=000 : A<br>If Destination Register=001 : B<br>If Destination Register=010 : E<br>If Destination Register=011 : N<br>All other values are currently reserved.<br>**NOTE:** The source register and destination register fields must not be the same. |
| 11-10 | |
| 9-8<br>Source Segment | Source Segment. Used when copying a register segment to specify which segment in the source register to copy from.<br>If Source Segment=00 : Segment 0<br>If Source Segment=01 : Segment 1<br>If Source Segment=10 : Segment 2<br>If Source Segment=11 : Segment 3 |

*Table continues on the next page...*

### Table 7-79. PKHA Mode register, field descriptions for copy memory functions (continued)

| Bits | Description |
|---|---|
| | **NOTE:** These bits must be zero when E is the destination register. |
| 7-6<br>Destination Segment | Destination Segment. Used when copying a register segment to specify which segment in the Destination Register to copy to.<br>If Destination Segment=00 : Segment 0<br>If Destination Segment=01 : Segment 1<br>If Destination Segment=10 : Segment 2<br>If Destination Segment=11 : Segment 3<br>**NOTE:** These bits must be zero when E is the destination register. |
| 5-0<br>Function | Function. Indicates which copy function to execute.<br>If Function=010000 : Copy Memory N-Size (copies the same number of words as are in the modulus.)<br>If Function=010001 : Copy Memory SRC-Size (copies the number of words specified in the source's size register) |

This table gives the encodings for the PKHA memory-to-memory copy functions. The top encoding in each cell is for Copy Memory, N-Size, and the bottom encoding is for Copy Memory, Source-Size ( Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)).

The encoding is in bits 19-0, including PKHA_MODE (i.e. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS) and reserved bits. (Hex)

### Table 7-80. Mode values for PKHA copy memory functions

| | Destination Memory | | | |
|---|---|---|---|---|
| **Source Memory** | **A** | **B** | **N** | **E** |
| A | | 00410<br>00411 | 00C10<br>00C11 | 00810<br>00811 |
| B | 20010<br>20011 | | 20C10<br>20C11 | 20810<br>20811 |
| N | 60010<br>60011 | 60410<br>60411 | | 60810<br>60811 |

This table gives the encodings for the PKHA memory-to-memory copy functions, when segments are involved. The top encoding in each cell is for Copy Memory, N-Size, and the bottom encoding is for Copy Memory, Source-Size ( Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)).

The encoding is in bits 19-0, including PKHA_MODE (i.e. PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS) and reserved bits. (Hex)

### Table 7-81. Mode values for PKHA copy memory by segment functions

| | Destination Quadrant | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Source Quadrant** | **A0** | **A1** | **A2** | **A3** | **B0** | **B1** | **B2** | **B3** | **N0** | **N1** | **N2** | **N3** |
| A0 | | | | | 00410 | 00450 | 00490 | 004D0 | 00C10 | 00C50 | 00C90 | 00CD0 |

*Table continues on the next page...*

Table 7-81. Mode values for PKHA copy memory by segment functions (continued)

| | Destination Quadrant | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Source Quadrant** | **A0** | **A1** | **A2** | **A3** | **B0** | **B1** | **B2** | **B3** | **N0** | **N1** | **N2** | **N3** |
| | | | | | 00411 | 00451 | 00491 | 004D1 | 00C11 | 00C51 | 00C91 | 00CD1 |
| A1 | | | | | 00510<br>00511 | 00550<br>00551 | 00590<br>00591 | 005D0<br>005D1 | 00D10<br>00D11 | 00D50<br>00D51 | 00D90<br>00D91 | 00DD0<br>00DD1 |
| A2 | | | | | 00610<br>00611 | 00650<br>00651 | 00690<br>00691 | 006D0<br>006D1 | 00E10<br>00E11 | 00E50<br>00E51 | 00E90<br>00E91 | 00ED0<br>00ED1 |
| A3 | | | | | 00710<br>00711 | 00750<br>00751 | 00790<br>00791 | 007D0<br>007D1 | 00F10<br>00F11 | 00F50<br>00F51 | 00F90<br>00F91 | 00FD0<br>00FD1 |
| B0 | 20010<br>20011 | 20050<br>20051 | 20090<br>20091 | 200D0<br>200D1 | | | | | 20C10<br>20C11 | 20C50<br>20C51 | 20C90<br>20C91 | 20CD0<br>20CD1 |
| B1 | 20110<br>20111 | 20150<br>20151 | 20190<br>20191 | 201D0<br>201D1 | | | | | 20D10<br>20D11 | 20D50<br>20D51 | 20D90<br>20D91 | 20DD0<br>20DD1 |
| B2 | 20210<br>20211 | 20250<br>20251 | 20290<br>20291 | 202D0<br>202D1 | | | | | 20E10<br>20E11 | 20E50<br>20E51 | 20E90<br>20E91 | 20ED0<br>20ED1 |
| B3 | 20310<br>20311 | 20350<br>20351 | 20390<br>20391 | 203D0<br>203D1 | | | | | 20F10<br>20F11 | 20F50<br>20F51 | 20F90<br>20F91 | 20FD0<br>20FD1 |
| N0 | 60010<br>60011 | 60050<br>60051 | 60090<br>60091 | 600D0<br>600D1 | 60410<br>60411 | 60450<br>60451 | 60490<br>60491 | 604D0<br>604D1 | | | | |
| N1 | 60110<br>60111 | 60150<br>60151 | 60190<br>60191 | 601D0<br>601D1 | 60510<br>60511 | 60550<br>60551 | 60590<br>60591 | 605D0<br>605D1 | | | | |
| N2 | 60210<br>60211 | 60250<br>60251 | 60290<br>60291 | 602D0<br>602D1 | 60610<br>60611 | 60650<br>60651 | 60690<br>60691 | 606D0<br>606D1 | | | | |
| N3 | 60310<br>60311 | 60350<br>60351 | 60390<br>60391 | 603D0<br>603D1 | 60710<br>60711 | 60750<br>60751 | 60790<br>60791 | 607D0<br>607D1 | | | | |

## 7.18.4 PKHA OPERATION: Elliptic Curve Functions

**NOTE**

Note that the elliptic curve functions with outputs going to the A RAM are identical to those with outputs going to the B RAM. The only difference is the output destination.

Table 7-82. PKHA Mode Register Format for Elliptic Curve Functions

| 19 | 18 | 17 | 16 | | 11 | 10 | 9-8 | 7-6 | 5-0 |
|---|---|---|---|---|---|---|---|---|---|

*Table continues on the next page...*

### Table 7-82. PKHA Mode Register Format for Elliptic Curve Functions (continued)

| Reserved | Reserved | F2m | R2 | ... | Reserved | Teq | OutSel | Reserved | Function |
|---|---|---|---|---|---|---|---|---|---|
| *PKHA_MODE_MS* | | | | | *PKHA_MODE_LS* | | | | |

### Table 7-83. PKHA Mode register, format for elliptic curve operation

| Bits | Description |
|---|---|
| 17<br>F2m | F2m. Indicates whether to use integer or binary polynomial arithmetic in executing the function.<br>If F2m=0 : Integer (prime) curve<br>If F2m=1 : Binary polynomial curve. (Not valid for all curve types.) |
| 16<br>R2 | (R2 mod N). Indicates whether the term (R2 mod N) must be supplied as an input or will be calculated by the routine.<br>If R2=0 : ($R^2$ mod N) is calculated and applied, if needed<br>If R2=1 : ($R^2$ mod N) is an input. (Not valid for all functions.) |
| 11<br>Reserved | Reserved |
| 10<br>Teq | Timing Equalized. Indicates that a timing equalized version of the function should be executed.<br>If Teq=0 : No timing equalization<br>If Teq=1 : Timing equalization. (Not valid for all functions.) |
| 9-8<br>OutSel | Output destination select. Indicates which memory should contain the output of the selected function.<br>If OutSel=00b : B<br>If OutSel=01b : A<br>If OutSel=10b : Reserved<br>If OutSel=11b : Reserved |
| 7-6 | Reserved |
| 5-0<br>Function | Function. Indicates which elliptic curve function to execute.<br>If Function=001001b : ECC Point Add (P1 + P2)<br>If Function=001010b : ECC Point Double (P2 + P2)<br>If Function=001011b : ECC Point Multiply (E x P1)<br>if Function=011100b : ECC Check Point<br>All other values for this field are currently reserved or are Table 7-71,Table 7-78, or Table 7-73. |

Elliptic Curve Functions over a prime field (ECC_MOD), where prime p > 3.

### Table 7-84. List of mode values for Prime Field ($F_p$) Elliptic Curve Arithmetic Functions

| Function name | Brief description | Output reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| ECC_MOD_ADD | ECC prime field point add - affine coordinates | B | 0 | 00009 | ECC $F_p$ Point Add, Affine Coordinates (ECC_MOD_ADD) Function |
| | | A | 0 | 00109 | |
| ECC_MOD_ADD_R2 | ECC prime field point add - affine coordinates, R2 input | B | 0 | 10009 | ECC $F_p$ Point Add, Affine Coordinates, $R^2$ Mod N Input (ECC_MOD_ADD_R2) Function |
| | | A | 0 | 10109 | |
| ECC_MOD_DBL | ECC prime field point double - affine coordinates | B | 0 | 0000A | ECC $F_p$ Point Double, Affine Coordinates (ECC_MOD_DBL) Function |
| | | A | 0 | 0010A | |
| ECC_MOD_MUL<br>ECC_MOD_MUL_TEQ | ECC prime field point multiply - affine coordinates | B | 0 | 0000B | ECC $F_p$ Point Multiply, Affine Coordinates (ECC_MOD_MUL and ECC_MOD_MUL_TEQ) Function |
| | | A | 0 | 0010B | |
| | Timing equalized version | B | 1 | 0040B | |
| | | A | 1 | 0050B | |
| ECC_MOD_MUL_R2<br>ECC_MOD_MUL_R2_TEQ | ECC prime field point multiply - affine coordinates, r2 mod n input | B | 0 | 1000B | ECC $F_p$ Point Multiply, $R^2$ Mod N Input, Affine Coordinates (ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ) Function |
| | | A | 0 | 1010B | |
| | Timing equalized version | B | 1 | 1040B | |
| | | A | 1 | 1050B | |
| ECC_MOD_CHECK_POINT | ECC Prime Field Point Validation | B | 0 | 0001C | ECC $F_p$ Check Point (ECC_MOD_CHECK_POINT) Function |
| | | A | 0 | XXXXX | |
| ECC_MOD_CHECK_POINT_R2 | ECC Prime Field Point Validation, R2 input | B | 0 | 1001C | ECC $F_p$ Check Point, $R^2$ Mod N Input, Affine Coordinates (ECC_MOD_CHECK_POINT_R2) Function |
| | | A | 0 | XXXXX | |

[1] PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

Elliptic Curve Functions over a binary field (ECC_F2M). All operate in polynomial basis.

## Table 7-85. List of mode values for Binary Field ($F_{2^m}$) Elliptic Curve Arithmetic Functions

| Function name | Brief description | Output reg | Teq | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) | Detailed description |
|---|---|---|---|---|---|
| ECC_F2M_ADD | ECC binary field point add - affine coordinates | B | 0 | 20009 | ECC $F_{2^m}$ Point Add, Affine Coordinates (ECC_F2M_ADD) Function |
| | | A | 0 | 20109 | |
| ECC_F2M_ADD_R2 | ECC binary field point add - affine coordinates, R2 input | B | 0 | 30009 | ECC $F_{2^m}$ Point Add, Affine Coordinates, $R^2$ Mod N Input (ECC_F2M_ADD_R2) Function |
| | | A | 0 | 30109 | |
| ECC_F2M_DBL | ECC binary field point double - affine coordinates | B | 0 | 2000A | ECC $F_{2^m}$ Point Double - Affine Coordinates (ECC_F2M_DBL) Function |
| | | A | 0 | 2010A | |
| ECC_F2M_MUL<br>ECC_F2M_MUL_TEQ | ECC binary field point multiply - affine coordinates | B | 0 | 2000B | ECC $F_{2^m}$ Point Multiply, Affine Coordinates (ECC_F2M_MUL and ECC_F2M_MUL_TEQ) Function |
| | | A | 0 | 2010B | |
| | Timing equalized version | B | 1 | 2040B | |
| | | A | 1 | 2050B | |
| ECC_F2M_MUL_R2<br>ECC_F2M_MUL_R2_TEQ | ECC binary field point multiply - affine coordinates, r2 mod n input | B | 0 | 3000B | ECC $F_{2^m}$ Point Multiply, $R^2$ Mod N Input, Affine Coordinates (ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ) Function |
| | | A | 0 | 3010B | |
| | Timing equalized version | B | 1 | 3040B | |
| | | A | 1 | 3050B | |
| ECC_F2M_CHECK_POINT | ECC Binary Polynomial Point Validation | B | 0 | A001C | ECC $F_{2^m}$ Check Point (ECC_F2M_CHECK_POINT) Function |
| | | A | 0 | XXXXX | |
| ECC_F2M_CHECK_POINT_R2 | ECC Binary Polynomial Field Point Validation, R2 input | B | 0 | B001C | ECC $F_{2^m}$ Check Point, $R^2$ (ECC_F2M_CHECK_POINT_R2) Function |
| | | A | 0 | XXXXX | |

[1] PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

## 7.19 SIGNATURE command

Trusted descriptors end with a SIGNATURE command, which requires the descriptor's signature (HMAC) to be validated before allowing it to run. SIGNATURE commands also support regeneration of the signature if the trusted descriptor modifies itself.

Trusted descriptors can be created and signed with a signature (a keyed hash) when executed from a specially privileged job ring. (See Trusted descriptors.) Trusted descriptors can be used to integrity protect the descriptor and to bind a key to a descriptor.

The SIGNATURE command that generates and verifies the keyed hash is always the last command of a trusted descriptor, although additional SIGNATURE commands can appear within the descriptor. The signature (HMAC) immediately follows the last SIGNATURE command in the trusted descriptor. When the descriptor is created:

- Room must be left at the end of the buffer for the 32-byte signature
- The length of the descriptor must include the signature.

DECO does not read the signature when creating the signature, so any initial value can be placed there.

If a trusted descriptor has a shared descriptor, the shared descriptor is part of the keyed hash computation. The shared descriptor is hashed first, followed by the descriptor; this is the order in which they appear in the descriptor buffer. The final hash is the value computed for both.

**NOTE**

It is an error for a SIGNATURE command to be in a descriptor that is not trusted or being made trusted.

**NOTE**

Because the SIGNATURE command must be the last command executed in the descriptor, trusted descriptors cannot have the REO bit set in their header. Doing so results in an error.

SIGNATURE types are available that allow a portion of the following command to not be included in the keyed hash. This provides flexibility in changing the address or the immediate data specified by a command. For example, the following command may a LOAD command, which contains the command word itself followed by a pointer. These SIGNATURE types would allow the command word to be part of the keyed hash but would exclude the pointer from the calculation. The writer of the trusted descriptor is responsible for using these SIGNATURE types only when the skipped information does not need to be integrity protected, meaning any immediate data or any address is permissible.

## NOTE

Skipping the signature over immediate data would allow a malicious user to shorten the length of the immediate data and insert additional commands that would not be included in the keyed hash. Note that this could be done without altering the overall length of the descriptor. To prevent this, it is recommended that the first four bytes of an immediate command always be protected by the keyed hash. Because the length of the immediate data is included in the keyed hash, the length cannot be altered such that additional commands can substitute for a portion of the immediate data.

### Table 7-86. SIGNATURE command format

| 31–27 | 26-20 | 19–16 |
|---|---|---|
| CTYPE = 10010 | Reserved | TYPE |
| **15-0** | | |
| Reserved | | |
| *Additional words of SIGNATURE command* | | |
| 8 Words to hold the Signature (these are used in types 0000, 0001 and 0010 only) | | |

### Table 7-87. SIGNATURE command field descriptions

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>IF CTYPE=10010b : Signature command |
| 26-20 | Reserved |
| 19-16<br>TYPE | See Table 7-88 |
| 15-0 | Reserved |

### Table 7-88. TYPE field description

| Type | Meaning | Instructions |
|---|---|---|
| 0000 | SIGNATURE command types 0000, 0001, or 0010 must be the last command that is executed in a trusted descriptor. If one of these types is used, the trusted descriptor signature (the keyed hash value) immediately follows the command. It is an error for a SIGNATURE command with one of these types to appear anywhere other than at the end of the descriptor. | Type 0000, when executed, terminates execution of the descriptor normally. |
| 0001 | | Type 0001 indicates that the descriptor should be rehashed and the keyed hash updated following descriptor execution. This type is used in cases where the descriptor could modify itself during execution. Note that the rehash and update is always done whether the descriptor was modified or not. Following the rehash and update, descriptor execution terminates normally. |

*Table continues on the next page...*

**Table 7-88. TYPE field description (continued)**

| Type | Meaning | Instructions |
|---|---|---|
| 0010 | | Type 0010 indicates that the descriptor should be rehashed and the keyed hash updated following descriptor execution if, upon completion, the MATH_Z bit is set. This type is used in cases where the descriptor could modify itself during execution but updating the keyed hash should be conditional. This version allows the rehash and update to be skipped when no change has been made to the descriptor. If MATH_Z is 0, descriptor execution immediately terminates normally. Otherwise, descriptor execution terminates normally after the rehash and update. |
| 1010 | SIGNATURE command types 1010, 1011, and 1100 are used to include only a portion of the following command in the keyed hash calculation, omitting the remainder of the command from the calculation. There is no hash value associated with this type, so it is an error for this type to appear at the end of the descriptor. These types allow the trusted descriptor to be modified with other offsets, addresses and lengths without invalidating the signature. Note that the SIGNATURE command is, itself, included in the hash so that it cannot be added later. | Type 1010 instructs SEC to hash only the first 2 bytes of the next command. |
| 1011 | | Type 1011 instructs SEC to hash only the first 3 bytes of the next command. |
| 1100 | | Type 1100 instructs SEC to hash only the first 4 bytes of the next command. |
| Others | Reserved | |

Two types of the final SIGNATURE command, 0001 and 0010 described in the above table, will recompute and update the signature in memory. These types are used when the trusted descriptor modifies itself and the modified version is to be used thereafter. Note that it is up to the descriptor writer to ensure that the copy of the descriptor in memory is updated using a STORE command. However, this update should only be done once all other commands in the descriptor have completed successfully. That is, the penultimate command should be the STORE to update the descriptor and the final command must be the SIGNATURE command. (If the update was done earlier and an error was detected prior to the SIGNATURE command running, the trusted descriptor could never be run again since the signature won't match.) The final signature command will wait to run until all reads have completed, all write data has been taken by the DMA, and all internal moves have completed. However, the final signature command is not a Done Checkpoint, which means that it will not wait for CHAs to complete.

## 7.20 JUMP (HALT) command

The JUMP command has the following uses:

- Alters the execution order of descriptor commands

- Pauses execution until specified conditions are satisfied
- Halts the execution of the descriptor if specified conditions are satisfied

JUMP command format shows the format of the JUMP command, and Table 7-90 describes the JUMP command field definitions.

The JUMP command may or may not be a checkpoint depending on its conditions and type.

## 7.20.1 Jump type

The JUMP command has eight different types, distinguished by the value in the JUMP TYPE field. All of these types specify a tested condition and take some execution flow action if the tested condition evaluates as true, and simply continue with the next command if the tested condition is false. See Test type for an explanation of what it means for the tested condition to be true.

Four of these jump types are true conditional jumps, another two are conditional halts, and the last two are a conditional subroutine call and a conditional subroutine return. Regardless of the jump type, the execution of the command waits for any specified wait conditions to be satisfied before the conditional action (jump, call, return, or halt) is taken. Some wait conditions can be specified in the CLASS field (wait for the Class 1 CHA to be done, wait for the Class 2 CHA to be done, or both), and additional wait conditions can be specified with the TEST CONDITION field (if JSL = 1).

### 7.20.1.1 Local conditional jump

The local conditional jump works as follows:

- If the tested condition is true, a JUMP command of the local conditional jump type continues the execution sequence at a new point within the descriptor buffer.
- If the tested condition is false, the jump is not taken and execution continues with the command that follows the JUMP command.

Local jumps are relative. The LOCAL OFFSET field is treated as an 8-bit 2's complement number that is added to the position of the JUMP command within the descriptor buffer. For example, a jump of one goes to the next 32-bit word and a jump of two skips one 32-bit word. Backward jumps are performed using 2's complement negative numbers.

A LOCAL OFFSET of 0 is a shorthand means of jumping back to the start of the descriptor buffer, which is either the start of the job descriptor if there is no shared descriptor or the start of the shared descriptor, if there is one. (see Figure 7-2)

#### 7.20.1.2 Local conditional increment/decrement jump

The local conditional increment/decrement jump is simply a local conditional jump that either increments or decrements a specified register, updates the math conditions based upon the result, and then evaluates the selected math conditions to determine whether or not the jump should be taken:

- If the tested condition is satisfied, command execution continues at a new point within the descriptor buffer.
- If the tested condition is not satisfied, the jump is not taken and execution continues with the command that follows the JUMP command.

Note that the increment and decrement jump types use a different JUMP command format than the other jump types. The four most-significant bits of the TEST CONDITION field are replaced with a SRC_DST field that specifies the register that is to be incremented or decremented. The least significant four bits of the TEST CONDITION field constitute the MATH CONDITION field, which specifies the tested conditions that are evaluated to determine whether the jump is taken or not.

Any of the legal choices for the SRC0 field of the MATH command which are also legal choices for the DEST field of the MATH command may be selected as the register to increment or decrement. Use the same value to select the register as is used for the MATH command.

#### 7.20.1.3 Non-local conditional jump

The non-local conditional jump is just like the local conditional jump except that the target of the jump must be the header of a job descriptor or trusted descriptor. Note that the target descriptor may not be a shared descriptor nor may the target descriptor have a shared descriptor. The pointer to the target descriptor is in the one or two words following the JUMP command.

- If the tested condition evaluates to true, the jump is taken.
- If the tested condition evaluates to false, the jump is not taken and execution continues with the command following the pointer.

NOTE

It is permissible to JUMP from a job descriptor to another job descriptor or from a job descriptor or a trusted descriptor to another trusted descriptor, but jumping from a trusted descriptor to a job descriptor results in an error.

#### 7.20.1.4 Conditional halt

This JUMP command is actually a conditional halt, meaning it stops the execution of the current descriptor if the tested condition evaluates to true. In this case the PKHA/Math condition bits (see the "TEST CONDITION bits when JSL=0" column in the TEST CONDITION field in JUMP command format) are written out right-justified in the SSED field of the job termination status word (see Job termination status/error codes).

If the tested condition evaluates to false, the descriptor is not halted and execution instead continues with the command that follows the jump.

NOTE

If the specified conditions evaluate as true, this command will always result in a nonzero status being returned for this job. Therefore, such a job will always appear to have encountered an error. The 8-bit error code will, as described above, be the PKHA and Math status flags rather than one of the predefined error codes.

#### 7.20.1.5 Conditional halt with user-specified status

A JUMP command with the user-specified status option is another type of conditional halt. If the tested condition is true, it stops execution of the descriptor but instead of writing the PKHA/Math condition bits, this conditional halt writes out the value in the LOCAL OFFSET field (again, right-justified in the SSED field of the job termination status word). The interpretation of the code in the LOCAL OFFSET field is user-specified, so it could be used during debugging to indicate that execution reached a certain point in a particular descriptor. If the tested condition evaluates to false, execution continues with the command following the jump.

NOTE

If the specified conditions evaluate as true, and the LOCAL OFFSET field is nonzero, this command will result in a nonzero status being returned for this job. That is, it will appear that such a job encountered an error. The 8-bit error code will, as described above, be a copy of the LOCAL OFFSET field rather than one of the predefined error codes.

**NOTE**

If the specified conditions evaluate as true, and the LOCAL OFFSET is zero, this command will terminate execution of the descriptor with normal status. That is, it will appear that such a job terminated normally. This is a convenient way to terminate execution in the middle of a descriptor when it can be determined that all work is done rather than having to jump to the end of the descriptor.

#### 7.20.1.6 Conditional subroutine call

A JUMP command with the subroutine call option is another type of local conditional jump. If the tested condition is true, it jumps to the specified location in the descriptor buffer but also saves the return address. The return address is the location immediately following the JUMP command. If the tested condition evaluates to false, execution continues with the command following the JUMP.

Note that only one return address can be saved, so subroutine calls cannot be nested. The descriptor writer is responsible for enforcing this as no error will be thrown if subroutine calls are nested.

**NOTE**

A built-in protocol is, in fact, also a special subroutine call. The return address is used to note where execution should resume following the execution of the built-in protocol. Therefore, while a protocol may be called from within a subroutine, a subsequent subroutine return will return to the command following the protocol command rather than the command following the subroutine call.

Each time a conditional subroutine call is taken or a built-in protocol is started, the return address is saved. That return address will be maintained until it is overwritten by another conditional subroutine call or built-in protocol. Therefore, it is possible to have one subroutine call which corresponds to multiple subroutine returns. It is also possible to match subroutine returns with calls to built-in protocols.

### 7.20.1.7 Conditional subroutine return

A JUMP command with the subroutine return option is another type of local conditional jump. In this case the local offset is ignored because the target of the jump is taken from the previously saved return address. If the tested condition is true, the subroutine return jumps to the saved return address. This address is the location immediately following the most recently executed command which updated the return address. One command which updates the return address is a conditional subroutine call in which the tested condition evaluated as true. The other command which updates the return address is a built-in protocol. If the tested condition evaluates to false, execution continues with the command following the subroutine return command.

**NOTE**

See the previous section, Conditional subroutine call, for important details on the use of the conditional subroutine return.

## 7.20.2 Test type

The TEST TYPE field is used to specify when the conditional jump/halt tested condition is considered to be met. The test type options are:

- 00—All specified test conditions are true. (Logical AND of all conditions.)
- 01—All specified test conditions are false. (Logical NOR of all conditions.)
- 10—Any specified test condition is true. (Logical OR of all conditions.)
- 11—Any specified test condition is false. (Logical NAND of all conditions.)

To create an unconditional jump, use TEST TYPE = 00 (all specified conditions true) and clear all TEST CONDITION bits because the tested condition is considered to be true if no test condition bits are set.

To create an unconditional jump/halt with a JSL = 1 conditional wait condition, use TEST TYPE = 10 (any specified condition is true). This always jumps or halts once the wait is completed because the selected conditional wait condition(s) are always true after the wait is completed.

A local conditional jump with offset 1 (signifying a jump to the following command) is a no-op because the next command in sequence is executed whether or not the jump is taken. This is true regardless of the TEST TYPE and TEST CONDITION settings. However, a wait condition can be specified to prevent the next command from executing until the conditions are satisfied. This is a common use case for the local conditional jump.

### 7.20.3 JSL and TEST CONDITION fields

The JSL field selects between two different interpretations of the TEST CONDITION field:

- When JSL = 0, the conditional jump/halt bits select various MATH and PKHA status conditions. These are used to jump or halt if the tested condition is satisfied.
- When JSL = 1, the bits in the TEST CONDITION field can affect the action taken by the JUMP Command in two ways.
    - Some of the TEST CONDITION bits are conditional jump/halt bits. The JQP, SHRD, and SELF test conditions are typically used to avoid storing data that the next descriptor might change or to prevent reloading data that is already available because it was left by the previous descriptor.
    - The remainder of the TEST CONDITION bits are conditional wait bits. The CALM, NIP, NIFP, NOP, and NCP conditional wait bits are used to time loads, moves, and stores properly. If conditional wait bits are set the JUMP command stalls until all of the specified wait conditions become true. All of the conditional wait bits must evaluate to true independent of the TEST TYPE specified. In other words, you can't wait for one of two conditional wait conditions to become true; you must wait for both. Once all the contional wait conditions are true, the jump or halt either occurs or not, depending upon whether all of the specified conditions are satisfied. Note that once the wait has completed, the selected conditional wait conditions are always true; because they are evaluated as part of the tested condition, they can affect whether the jump or halt action is taken. Note that the CLASS bits are, in fact, conditional wait bits even though they are not used in the decision on whether to take the JUMP.

For example, if a JUMP command is executed with JSL = 1 and the TEST CONDITION bits NIP, NIFP, JQP, and SELF are set, the JUMP command stalls until both of the following are true:

- No input to the input data FIFO is pending (NIP).
- No input to the information FIFO is pending (NIFP).

Because these are conditional wait bits, the command waits until all of the wait conditions are true before evaluating the remaining conditions. The evaluation depends upon the test conditions that are selected, the state of the selected conditions, and the value in the TEST TYPE field:

- TEST TYPE = 00 (if all conditions are true): the jump or halt occurs if another job wants to share this shared descriptor (JQP) and this shared descriptor is running in the same DECO (SELF) as the one from which it was shared.

- TEST TYPE = 01 (if all conditions are false): the jump or halt never occurs because the NIP and NIFP conditions are true after the wait completes.
- TEST TYPE = 10 (if any condition is true): the jump or halt always occurs because the NIP and NIFP conditions are true after the wait completes.
- TEST TYPE = 11 (if any condition is false): the jump or halt occurs if no job wants to share this shared descriptor (JQP) or this shared descriptor is not running in the same DECO (SELF) as the one from which it was shared.

## 7.20.4 JUMP command format

**Table 7-89. JUMP command format**

| | 31-27 | 26-25 | 24 | 23-20 | 19-18 | 17-16 |
|---|---|---|---|---|---|---|
| | CTYPE = 10100 | CLASS | JSL | JUMP TYPE | Reserved | TEST TYPE |
| | 15-8 | | | 7-0 | | |
| Format used with all jump types except 0001 and 0011 | TEST CONDITION | | | LOCAL OFFSET | | |
| | 15-12 | 11-8 | | 7-0 | | |
| Format used with jump types 0001 and 0011 | SRC_DST | MATH CONDITION | | LOCAL OFFSET | | |
| | *Additional words of JUMP command* | | | | | |
| | Pointer (one or two words); see Address pointers) (this field is present for non-local JUMPs only) | | | | | |

**Table 7-90. JUMP command field descriptions**

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command type<br>If CTYPE=10100 : JUMP command |
| 26-25<br>CLASS | Class<br>Wait until specified class type CHA(s) is done before evaluating jump/halt conditions. For CLASS != 00, this makes the JUMP command a DONE checkpoint.<br>If CLASS=00 : None<br>If CLASS=01 : Class 1<br>If CLASS=10 : Class 2<br>If CLASS=11 : Both Class 1 and Class 2 |
| 24<br>JSL | Jump Select Type<br>Selects which definition of the TEST CONDITION field to use.<br>If JSL=0 : MATH and PKHA status conditions<br>If JSL=1 : Various jump/halt and wait conditions (Note that JSL=1 is prohibited with jump types 0001 and 0011 and such usage will result in an error.) |

*Table continues on the next page...*

## Table 7-90. JUMP command field descriptions (continued)

| Field | Description |
|---|---|
| 23-20<br>JUMP TYPE | Jump Type<br>Specifies the action taken by the JUMP Command. See Jump type for more information.<br>If JUMP TYPE=0000 : Local conditional jump. Evaluates the specified TEST CONDITION to determine whether the local jump should be taken.<br>If JUMP TYPE=0001 : Local conditional increment jump. Increments the register specified in SRC_DST before evaluating the specified MATH CONDITION.<br>If JUMP TYPE=0010 : Conditional subroutine call. Evaluates the specified TEST CONDITION to determine whether the local subroutine call should be taken.<br>If JUMP TYPE=0011 : Local conditional decrement jump. Decrements the register specified in SRC_DST before evaluating the specified MATH CONDITION.<br>If JUMP TYPE=0100 : Non-local conditional jump. Evaluates the specified TEST CONDITION to determine whether the non-local jump should be taken.<br>If JUMP TYPE=0110 : Conditional subroutine return. Evaluates the specified TEST CONDITION to determine whether the subroutine return should be taken.<br>If JUMP TYPE=1000 : Conditional Halt. If the specified TEST CONDITION is true, this returns the PKHA/ MATH bits as status. (see "TEST CONDITION bits when JSL=0" column in the TEST CONDITION field) and halts descriptor execution with error status.<br>If JUMP TYPE=1100 : Conditional Halt with user-specified status. If the specified TEST CONDITION is true, this returns the value in the LOCAL OFFSET field as status and halts descriptor execution with error status unless the LOCAL OFFSET is zero, in which case descriptor execution terminates normally.<br>All other codes are reserved, and will generate an error. |
| 19-18 | Reserved |
| 17-16<br>TEST TYPE | Test Type. This field defines how the condition code bits (see TEST CONDITION field) should be interpreted. See Test type for more information.<br>If TEST TYPE=00 : Jump/halt if ALL selected conditions are true. That is, jump or halt if all the status conditions are true for all TEST CONDITION bits that are 1. Note that if JSL = 1 and one or more conditional wait bits is set, the command waits for all selected conditional wait conditions to be true before the conditional jump/halt conditions are evaluated. The jump/halt then takes place if these conditions are all true.<br>If TEST TYPE=01 : Jump/halt if ALL selected conditions are false. That is, jump or halt if all the status conditions are false for all TEST CONDITION bits that are 1. Note that if JSL=1 and one or more Conditional Wait bits is set, the command will wait for all selected Conditional Wait conditions to be true and the jump/halt will not take place (since the Condition Wait condition(s) are now true). If no Conditional Wait bits are set, the jump/halt will take place if all of the selected Conditional Jump/Halt conditions are false.<br>If TEST TYPE=10 : Jump/halt if ANY selected condition is true. That is, jump or halt if any status condition is true for a TEST CONDITION bit that is 1. Note that if JSL=1 and one or more Conditional Wait bits is set, the command will wait for all selected Conditional Wait conditions to be true and then the jump/halt will take place (since the Condition Wait condition(s) are now true). If no Conditional Wait bits are set, the jump/halt will take place if any of the selected Conditional Jump/Halt conditions are true.<br>If TEST TYPE=11 : Jump/halt if ANY selected condition is false. That is, jump or halt if any status condition is false for a TEST CONDITION bit that is 1. Note that if JSL=1 and one or more Conditional Wait bits is set, the command will wait for all selected Conditional Wait conditions to be true and then Tested Condition will be evaluated. Whether a wait occurs or not, the jump/halt will take place if any selected Conditional Jump/Halt condition is false. |
| 15-8<br>TEST CONDITION | Test Condition.***This 8-bit field is used with all jump types except 0001 and 0011.*** The interpretation of the TEST CONDITION field depends upon the value of the JSL field, a shown in Table 7-91. See JSL and TEST CONDITION fields for more information. |

*Table continues on the next page...*

## Table 7-90. JUMP command field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15-12<br>SRC_DST</td><td>Source/Destination. <b><i>This four-bit field is used only with jump types 0001 and 0011. It replaces the most-significant four bits of the TEST CONDITION field.</i></b> This field is used to select the register that will be incremented (jump type 0001) or decremented (jump type 0011) before the selected math condition is evaluated to determine whether the local jump will be taken. For 8-byte registers, only the least-significant 4 bytes are used. (That is, the length of the math operation is restricted to 4 bytes.)<br>If SRC_DST=0000 : Math Register 0<br>If SRC_DST=0001 : Math Register 1<br>If SRC_DST=0010 : Math Register 2<br>If SRC_DST=0011 : Math Register 3<br>If SRC_DST=0101 : Math Register 4<br>If SRC_DST=0110 : Math Register 5<br>If SRC_DST=0111 : DECO Protocol Override Register<br>If SRC_DST=1000 : Sequence In Length (SIL)<br>If SRC_DST=1001 : Sequence Out Length (SOL)<br>If SRC_DST=1010 : Variable Sequence In Length (VSIL)<br>If SRC_DST=1011 : Variable Sequence Out Length (VSOL<br>If SRC_DST=1101 : Math Register 6<br>If SRC_DST=1110 : Math Register 7<br>All other values are reserved.</td></tr>
<tr><td>11-8<br>MATH CONDITION</td><td>Math condition. <b><i>This four-bit field is used only with jump types 0001 and 0011. It is identical with the least-significant four bits of the TEST CONDITION field and uses the same definitions.</i></b> This field is used to select the math conditions that will be evaluated to determine whether the local jump should be taken.
<table>
<tr><td>bit 11</td><td>bit 10</td><td>bit 9</td><td>bit 8</td></tr>
<tr><td>MATH N</td><td>MATH Z</td><td>MATH C</td><td>MATH NV</td></tr>
<tr><td>The result is negative.</td><td>The result is zero.</td><td>The operation resulted in a carry or borrow.</td><td>Used for signed compares. This is the XOR of the sign bit and 2's complement overflow.</td></tr>
</table></td></tr>
<tr><td>7-0<br>LOCAL OFFSET</td><td>For local jumps this field specifies the offset of the JUMP target from the JUMP command's address in the descriptor buffer. This field is ignored for non-local JUMPs. If the LOCAL OFFSET is 0, the target is the start of the Descriptor Buffer. For non-zero values, the target address is relative to the JUMP Command. That is, the field is interpreted as an 8-bit 2's complement number that is added to the index of the JUMP Command to yield the 32-bit word of the target. For Halt with status, the LOCAL OFFSET will be returned in Descriptor status. This will show up in the Output Job Status as the USTA field. If nonzero on halt with status, an error is reported.</td></tr>
<tr><td>31-0<br>POINTER</td><td>Pointer (32-bit or 64-bit). This field is present only for non-local jumps. This is the address of the Descriptor to which to jump if the jump is taken.</td></tr>
</table>

## Table 7-91. TEST CONDITION bit settings

| Bit # | TEST CONDITION bits when JSL=0 | | | TEST CONDITION bits when JSL=1 | | |
|---|---|---|---|---|---|---|
| 15 | PKHA IS_ZERO | For Finite Field operations the result of a PKHA operation is zero. For ECC operations, the result is a Point at Infinity. | Conditional Jump/Halt | JQP | Job Queue Pending. The Job Queue Controller has identified that another job wants to share this Shared Descriptor. This bit can be used to avoid storing data that the next Shared Descriptor would just refetch. This condition is false if this is not a Shared Descriptor. | Conditional Jump/Halt |
| 14 | PKHA GCD_1 | The greatest common divisor of two numbers is 1 (that is, the two numbers are relatively prime). | Conditional Jump/Halt | SHRD | SHARED. This Shared Descriptor was shared from a previously executed Descriptor. Depending on the type of sharing, this bit can be tested to conditionally jump over commands. For example, if the keys are shared they will already be in the Key Registers so decrypting and placing them in the Key Registers must be skipped. This condition is false if this is not a Shared Descriptor. | Conditional Jump/Halt |
| 13 | PKHA IS_PRIME | The given number is probably prime (that is, it passes the Miller-Rabin primality test). | Conditional Jump/Halt | SELF | The SELF bit indicates that this Shared Descriptor is running in the same DECO as the one from which it was shared. Hence, the Shared Descriptor may be able to assume that Context Registers, CHAs, and other items are still valid or available. This condition is false if this is not a Shared Descriptor. | Conditional Jump/Halt |
| 12 | Reserved | Must be 0. | — | CALM | All pending bus transactions for this DECO, whether internal or external, have completed. | Conditional Wait |
| 11 | MATH N | The negative math flag is set. | Conditional Jump/Halt | NIP | No input pending. No external loads, whether from LOAD, FIFO LOAD, SEQ LOAD, or SEQ FIFO LOAD, are pending. | Conditional Wait |
| 10 | MATH Z | The zero math flag is set. | Conditional Jump/Halt | NIFP | No iNformation FIFO entries pending. The NFIFO is empty and no data is waiting in the C1 or C2 alignment blocks. | Conditional Wait |
| 9 | MATH C | The carry/borrow math flag is set. | Conditional Jump/Halt | NOP | No output pending. No external stores, whether from STORE, FIFO STORE, SEQ STORE, or SEQ FIFO STORE, are pending. | Conditional Wait |
| 8 | MATH NV | The NV math flag is set. This is the XOR of the sign bit and 2's complement overflow. | Conditional Jump/Halt | NCP | No context load pending. There is no data in flight toward the context registers via the internal or external DMA. | Conditional Wait |

## 7.21 MATH and MATHI Commands

The MATH and MATHI commands compute simple mathematical functions of values in registers or specified via immediate data. The result can be written to a specified destination register or the result can be dropped. The commands set MATH condition bits that reflect the result of the mathematical operation (see MNV, MN, MC, and MZ). These condition bits can be tested with the JUMP commands, providing SEC with the flexibility to implement conditional processing constructs, including loops. In addition, the MC bit can be used to perform addition and subtraction of values larger than the math registers via borrow or carry.

Length must always be specified in the command, as it determines the size of the arguments used to set the MATH status bits. Note that the LENGTH field is used to mask off results after the math operation, not before, so the user must present properly sized data.

The MATHI command is useful when a one-byte immediate value is to be used. Since this immediate value is contained within the MATHI command word, this allows the MATHI command to be a single word rather than forcing the use of a two-word MATH command. This is useful since one-byte arguments are common. In some large descriptors, saving this one word several times can make the difference between fitting in the descriptor buffer and having to use multiple descriptors.

**Table 7-92. MATH and MATHI Commands, format**

<table>
<tr><th></th><th colspan="2">31–27</th><th>26</th><th>25</th><th>24</th><th>23–20</th><th>19–16</th></tr>
<tr><td>MATH:</td><td colspan="2">CTYPE = 10101</td><td>IFB</td><td>NFU</td><td>STL</td><td>FUNCTION</td><td>SRC0</td></tr>
<tr><td>MATHI:</td><td colspan="2">CTYPE = 11101</td><td>Reserved</td><td>NFU</td><td>SSEL</td><td>FUNCTION</td><td>SRC</td></tr>
<tr><th></th><th>15–12</th><th colspan="4">11–8</th><th>7-4</th><th>3–0</th></tr>
<tr><td>MATH:</td><td>SRC1</td><td colspan="4">DEST</td><td>Reserved</td><td>LEN</td></tr>
<tr><td>MATHI:</td><td>DEST</td><td colspan="5">IMM_VALUE</td><td>LEN</td></tr>
</table>

**Table 7-93. MATH command, field descriptions**

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command Type<br>If CTYPE=10101b : MATH command (fields defined in this table)<br>If CTYPE=11101b : MATHI command (fields defined in table Table 7-94 below) |
| 26<br>IFB | Immediate Four Bytes<br>If IFB=0 : Include full length immediate data in descriptor (length specified in LEN field)<br>If IFB=1 : Use only four bytes of immediate data even if LEN is 8. This shortens the descriptor by one word when 1, 2, or 4-byte immediate data is to be used in an 8-byte operation. The immediate data will automatically be zero padded out to 8 bytes. This bit has no effect if the LEN is less than 8. |

*Table continues on the next page...*

## Table 7-93. MATH command, field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>25<br>NFU</td><td>No Flag Update<br>If NFU=0 : Math flags will be updated as appropriate.<br>If NFU=1 : Preserve the existing math flag values by blocking all updates to math flags.</td></tr>
<tr><td>24<br>STL</td><td>Stall.<br>If STL=0 : Don't stall the execution of the MATH command.<br>If STL=1 : Stall MATH command. Causes the MATH command to take one extra clock cycle.</td></tr>
<tr><td>23-20<br>FUNCTION</td><td>This field specifies which function to perform, as listed in the table below titled Table 7-95. The operands are specified in the SRC0 and SRC1 fields and the result is written to the destination specified in the DEST field.</td></tr>
<tr><td>19-16<br>SRC0</td><td>The SRC0 field indicates the source of operand 0.
<table>
<tr><th>Source of Operand 0</th><th>SRC0 Field Value</th></tr>
<tr><td>Math Register 0</td><td>0h</td></tr>
<tr><td>Math Register 1</td><td>1h</td></tr>
<tr><td>Math Register 2</td><td>2h</td></tr>
<tr><td>Math Register 3</td><td>3h</td></tr>
<tr><td>Math Register 4</td><td>5h</td></tr>
<tr><td>Math Register 5</td><td>6h</td></tr>
<tr><td>Math Register 6</td><td>Dh</td></tr>
<tr><td>Math Register 7</td><td>Eh</td></tr>
<tr><td>Immediate data from descriptor words following the MATH command[1]</td><td>4h</td></tr>
<tr><td>Protocol Override (DPOVRD), left-extended with 0s</td><td>7h</td></tr>
<tr><td>Sequence In Length (SIL), left-extended with 0s</td><td>8h</td></tr>
<tr><td>Sequence Out Length (SOL), left-extended with 0s</td><td>9h</td></tr>
<tr><td>Variable Sequence In Length (VSIL)</td><td>Ah</td></tr>
<tr><td>Variable Sequence Out Length (VSOL)</td><td>Bh</td></tr>
<tr><td>ZERO (the value 0000 0000h) is used as operand 0</td><td>Ch</td></tr>
<tr><td>ONE (the value 0000 00001) is used as operand 0</td><td>Fh</td></tr>
</table></td></tr>
<tr><td>15-12<br>SRC1</td><td>The SRC1 field indicates the source of operand 1.
<table>
<tr><th>Source of Operand 1</th><th>SRC1 Field Value</th></tr>
<tr><td>Math Register 0</td><td>0h</td></tr>
<tr><td>Math Register 1</td><td>1h</td></tr>
<tr><td>Math Register 2</td><td>2h</td></tr>
<tr><td>Math Register 3</td><td>3h</td></tr>
<tr><td>Math Register 4</td><td>5h</td></tr>
</table></td></tr>
</table>

*Table continues on the next page...*

### Table 7-93. MATH command, field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Source of Operand 1</th><th>SRC1 Field Value</th></tr>
<tr><td>Math Register 5</td><td>6h</td></tr>
<tr><td>Math Register 6</td><td>Dh</td></tr>
<tr><td>Math Register 7</td><td>Eh</td></tr>
<tr><td>Immediate data from descriptor words following the MATH command[1]</td><td>4h</td></tr>
<tr><td>Protocol Override (DPOVRD), left-extended with 0s</td><td>7h</td></tr>
<tr><td>Variable Sequence In Length (VSIL)</td><td>8h</td></tr>
<tr><td>Variable Sequence Out Length (VSOL)</td><td>9h</td></tr>
<tr><td>Input Data FIFO[2,3]</td><td>Ah</td></tr>
<tr><td>Output Data FIFO[3,4]</td><td>Bh</td></tr>
<tr><td>ONE (the value 0000 0001h) is used as operand 1</td><td>Ch</td></tr>
<tr><td>ZERO (the value 0000 00000) is used as operand 1</td><td>Fh</td></tr>
</table>
</td></tr>
<tr><td>11-8<br>DEST</td><td>The DEST field specifies the destination for the result of the command as follows:
<table>
<tr><th>Destination for MATH operation result</th><th>DEST Field Value</th></tr>
<tr><td>Math Register 0</td><td>0h</td></tr>
<tr><td>Math Register 1</td><td>1h</td></tr>
<tr><td>Math Register 2</td><td>2h</td></tr>
<tr><td>Math Register 3</td><td>3h</td></tr>
<tr><td>Math Register 4</td><td>5h</td></tr>
<tr><td>Math Register 5</td><td>6h</td></tr>
<tr><td>Math Register 6</td><td>Dh</td></tr>
<tr><td>Math Register 7</td><td>Eh</td></tr>
<tr><td>Protocol Override</td><td>7h</td></tr>
<tr><td>Sequence In Length</td><td>8h</td></tr>
<tr><td>Sequence Out Length</td><td>9h</td></tr>
<tr><td>Variable Sequence In Length</td><td>Ah</td></tr>
<tr><td>Variable Sequence Out Length</td><td>Bh</td></tr>
<tr><td>No Destination. The result should not be written anywhere.[5]</td><td>Fh</td></tr>
<tr><td></td><td>All other values for this field are reserved.</td></tr>
</table>
</td></tr>
<tr><td>7-4</td><td>This field is reserved. All bits must be 0.</td></tr>
<tr><td>3-0<br>LEN</td><td>LEN denotes the length, in bytes, of the operation and the immediate value, if there is one.<br>1h : 1 byte<br>2h : 2 bytes</td></tr>
</table>

*Table continues on the next page...*

## Table 7-93. MATH command, field descriptions (continued)

| Field | Description |
|---|---|
| | 4h : 4 bytes<br>8h : 8 bytes<br>9h : 8 bytes, with word swapping performed prior to use if SEC STATUS REGISTER[PLEND]=0 (i.e. Little-Endian). LEN=9h is equivalent to LEN=8h (no word swapping) if SEC STATUS REGISTER[PLEND]=1 (i.e. Big-Endian).[6]<br>All other values are reserved.<br>**NOTE:** If the selected FUNCTION is shift_l or shift_r, a LEN value other than 8h may yield unexpected results. Also note that the IFB bit in the command can be used to override the LEN field for the immediate value. When set, the IFB (Immediate Four Bytes) bit allows the MATH command to use a 1, 2, or 4-byte immediate value (0 padded to the left) in the descriptor even though it is doing an 8-byte operation.<br>**NOTE:** If the Length is 8h but the destination is only 4 bytes, an error will be generated. The 4-byte destinations are SIL, SOL, and POVRD. |

1. If the data is less than 8 bytes, it is left-extended with 0s. If the data is less than 8 bytes it must be right-aligned. If SRC0 and SRC1 both specify Immediate data, the SRC0 data is in the first word following the MATH command and the SRC1 data is in the second word, and either the LEN field must be set to 4 bytes or the IFB field must be set to 1, else an error is generated.

2. The input data FIFO is popped when the MATH command executes unless the function is shld (shift and load). Note that this means a final pop may have to be done if the data consumed by the shld is the end of the data. If this is the last data to be consumed by DECO, then it is not necessary to pop the data, because leaving it there is not a problem if the input data FIFO is reset. The input FIFO is not automatically reset between job descriptors with the same shared descriptor unless the CIF bit in the Shared Descriptor is set. The input data FIFO is always reset betwen jobs without, or with different, shared descriptors. Note that the descriptor must have already created an NFIFO entry to get data to the DECO alignment block, from which the MATH command will pop it.

3. If SRC1 specifies either input data FIFO or output data FIFO, the MATH command does not execute until the corresponding FIFO has valid data. It is up to the user to ensure that a sufficient amount of data is present. The user must also realize that data comes out of the FIFOs left aligned. This means that if there are only five bytes, the data is in the left 5 bytes, not in the right 5 bytes, of the 8-byte source word.

4. The output data FIFO is popped when the MATH command executes unless the function is shld (shift and load). Note that this means a final pop may have to be done if the data consumed by the shld is the end of the data. If this is the last data to be consumed by DECO, then it is not necessary to pop the data, because leaving it there is not a problem if the output data FIFO is reset. The output FIFO is always cleared between descriptors whether shared or not.

5. No Destination is useful for setting flags when the actual result is not needed. An error will be generated if No Destination is selected when the FUNCTION is shift_l or shift_r.

6. An error will be generated for LEN=9h if IFB=1 or if both or neither of the operands is Immediate.

## Table 7-94. MATHI command, field descriptions

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command Type<br>If CTYPE=10101b : MATH command (see field definitions in table Table 7-93 above)<br>If CTYPE=11101b : MATHI command (fields defined in this table) |
| 26 | Reserved. Must be 0. |
| 25<br>NFU | No Flag Update<br>If NFU=0 : Math flags will be updated as appropriate.<br>If NFU=1 : Preserve the existing math flag values by blocking all updates to math flags. |

*Table continues on the next page...*

## Table 7-94. MATHI command, field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>24<br>SSEL</td><td>SSEL. Source Select. Selects the type and order of the operands to the math function: operand 0 &lt;function&gt; operand 1 -&gt; destination
<table>
<tr><th>SSEL value</th><th>operand 0 specified by</th><th>math function specified by</th><th>operand 1 specified by</th><th>destination specified by</th></tr>
<tr><td>0</td><td>SRC0 for MATH command</td><td>FUNCTION</td><td>IMM_VALUE</td><td>DEST</td></tr>
<tr><td>1</td><td>IMM_VALUE</td><td>FUNCTION</td><td>SRC1 for MATH command</td><td>DEST</td></tr>
</table>
<b>NOTE:</b> If FUNCTION=Ah (FBYT) it is illegal to set SSEL to 1.</td></tr>
<tr><td>23-20<br>FUNCTION</td><td>This field specifies which function to perform, as listed in the table below titled Table 7-95. The operands are specified in the SRC field and the IMM_VALUE field, and the result is written to the destination specified in the DEST field.</td></tr>
<tr><td>19-16<br>SRC</td><td>The SRC field indicates the source of one of the operands. The SRC field has two definitions, selected via the SSEL field:
<ul>
<li>If SSEL=0: the SRC field is defined the same as the MATH command's SRC0 field (see SRC0) except that IMM (4h) is not supported and will result in an error.</li>
<li>If SSEL=1: the SRC field is defined the same as the MATH command's SRC1 field (see SRC1) except that IMM (4h) is not supported and will result in an error.</li>
</ul></td></tr>
<tr><td>15-12<br>DEST</td><td>The destination for the result of the math operation. The MATHI command DEST field is defined the same as the MATH command's DEST field (see DEST), but is shifted to the left 4 bits to make room for the IMM_VALUE field.</td></tr>
<tr><td>11-4<br>IMM_VALUE</td><td>The IMM_VALUE field contains an 8-bit immediate value that is left-extended with 0s. This is used as either operand 0 or operand 1, as specified in the SSEL field.</td></tr>
<tr><td>3-0<br>LEN</td><td>LEN denotes the length, in bytes, of the operation (and the amount by which the IMM_VALUE is left-extended with 0s).<br>1h : 1 byte<br>2h : 2 bytes<br>4h : 4 bytes<br>8h : 8 bytes<br>All other values are reserved.<br><b>NOTE:</b> If the selected FUNCTION is shift_l or shift_r, a LEN value other than 8h may yield unexpected results.<br><b>NOTE:</b> If the Length is 8h but the destination is only 4 bytes, an error will be generated. The 4-byte destinations are SIL, SOL, and POVRD.</td></tr>
</table>

## Table 7-95. FUNCTION field values

| Value | Type | Description | Result |
|---|---|---|---|
| 0h | add | Perform addition operation on operand 0 and operand 1. | operand 0 + operand 1 |
| 1h | add_w_carry | Perform addition with a carry bit operation on operand 0 and operand 1. | operand 0 + operand 1 + MC |

*Table continues on the next page...*

Table 7-95. FUNCTION field values (continued)

| Value | Type | Description | Result |
|---|---|---|---|
| 2h | sub | Perform subtraction operation on operand 0 and operand 1. | operand 0 - operand 1 |
| 3h | sub_w_borrow | Perform subtraction with borrow operation on operand 0 and operand 1. | operand 0 - operand 1 - MC |
| 4h | or | Perform bitwise OR operation on operand 0 and operand 1 | operand 0 \| operand 1 |
| 5h | and | Perform bitwise AND operation on operand 0 and operand 1 | operand 0 & operand 1 |
| 6h | xor | Perform bitwise XOR operation on operand 0 and operand 1 | operand 0 ^ operand 1 |
| 7h | shift_l | Perform shift left operation. operand 0 should be shifted left by operand 1 bits; can't be used with "No Destination" | operand 0 << operand 1 |
| 8h | shift_r | Perform shift right operation. operand 0 should be shifted right by operand 1 bits; can't be used with "No Destination" | operand 0 >> operand 1 |
| 9h | shld | Perform 32-bit left shift of DEST and concatenate with left 32 bits of operand 1. shld is only meaningful when DEST specifies Math Registers 0-7. For all other destinations, this function will work like an ADD with operand 0 set to 0. (That is, operand 1 will be placed into DEST.) Note that if operand 1 and DEST are the same Math Register, then shld would do a word swap.<br>Function type shld is prohibited for the MATHI command. | {DEST[31:0], operand 1[63:32]} |
| Ah | zbyt or fbyt | MATH command: zbyt. Find zero bytes in operand 0. The function places into the destination seven bytes (if a 64-bit destination) or three bytes (if a 32-bit destination) of zeros followed by a single byte that contains a 1 in each bit position that corresponds to a byte of operand 0 that is all zeros.<br>MATHI command: fbyt. Find the immediate byte in operand 0. The function places into the destination seven bytes (if a 64-bit destination) or three bytes (if a 32-bit destination) of zeros followed by a single byte that contains a 1 in each bit position that corresponds to a byte of operand 0 that is equal to IMM_VALUE. For the fbyt function it is illegal to set SSEL=1. | result is shown at left |
| Bh | swap_bytes | Swap the order of the four bytes in the ms half of operand 0, and independently swap the order of the four bytes in the ls half of operand 0.<br>operand 0[39:32], operand 0[47:40], operand 0[55:48], operand 0[63:56],<br>operand 0[7:0], operand 0[15:8], operand 0[23:16], operand 0[31:24]<br>If this is used in conjunction with shld, the result of the two MATH operations will be an "end-for-end" swap of all 8 bytes.<br>Function swap_bytes is prohibited for the MATHI command. | result is shown at left |
| All other values for this field are reserved.<br>**NOTE:** A Compare operation is accomplished by selecting FUNCTION=sub, with DEST=No Destination and then doing a JUMP based on the CZ and/or CN flags. | | | |

All MATH and MATHI commands take one clock cycle to execute except for the shift_l and shift_r functions. For the shift_l and shift_r functions, the number of bit positions that the data is shifted is specified in operand 1.

- If the data is to be shifted 64 or more bit positions, the shift command takes two clocks. One clock decodes the command, and one clock stores all zeros in the DEST register. Because all bits are shifted off the end, the result is all zeros.
- If the data is to be shifted 63 or fewer bit positions, the shift_l and shift_r functions take at most two clocks more than the number of bits in operand that are 1. The shifter can shift any power-of-2 number of bit positions in one cycle, and there are up to two additional cycles of overhead. If an intermediate shift result is all 0, the remaining shifts are skipped, resulting in fewer clock cycles than the maximum.

Note that the shift_l and shift_r functions first copy the data specified by operand 0 into the register specified by DEST and then shift the data in the DEST register. If the source is 64 bits but the destination is a 32-bit register, the 64-bit source value is truncated to its least-significant 32 bits before the shifting begins. A shift_l works as expected, but a shift_r of data from a 64-bit source to a 32-bit destination shifts in 0s rather than shifts in bits from the most-significant 32-bits of the source.

When one source, operand 0 or operand 1, is immediate, then the length may be any legal value. If 1, 2, or 4 bytes, the value is right-aligned in the word following the command. If the value is 8 bytes, then the value is in the two words that follow the command. Note that the immediate data can be 4 bytes even if the LEN is 8 bytes if the IFB bit is set.

## 7.22 SEQ IN PTR command

The Sequence In Pointer (SEQ IN PTR) command is used to specify the starting address for an input sequence and the length of that sequence (see SEQ vs non-SEQ commands). Only one input sequence may be active within the DECO at any one time. An input sequence is initiated by executing a SEQ IN PTR command with PRE = 0. This causes the following:

- Starting address of the input sequence to be set to the value in the Pointer field or to the original pointer if RTO=1 or to the original output sequence pointer if SOP=1.
- The Sequence In Length register to be set to the value in the LENGTH field (if EXT = 0) or the EXT_LENGTH field (if EXT = 1). If rewinding, the LENGTH or EXT_LENGTH field is added to the current length.

Note that if the EXT bit is 0, the EXT_LENGTH field is omitted from the SEQ IN PTR command.

The input sequence terminates when one of the following occurs:

- All input data is utilized.

- An error occurs.
- A new input sequence is started by executing a SEQ IN PTR command with PRE = 0.

An error is flagged if a SEQ command attempts to input data if the execution of that command would cause the remaining length to go below 0. To extend the length of the sequence any number of additional SEQ IN PTR commands may be executed with PRE = 1. If PRE = 1, the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1) is added to the current Sequence In Length register value, but the address for the input sequence is unaffected. In this case the SEQ IN PTR command does not include a Pointer field. Additional length may also be added via the MATH and MATHI commands.

If the same input data needs to be processed again, the input pointer can be restored to the original starting address by executing a SEQ IN PTR with RTO = 1. However, if the job descriptor using this input sequence was submitted through the Queue Manager Interface, scatter tables are active, and input buffers are being released, it is not possible to back up if buffers have already been released. The SEQ IN PTR command does not include a Pointer field in this case.

**Table 7-96. SEQ IN PTR command, format**

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CTYPE = 11110 | | | | | RBS | INL | SGF | PRE | EXT | RTO | RJD | SOP | CTRL | Reserved | Reserved |
| | 15–0 | | | | | | | | | | | | | | | |
| | LENGTH (used if EXT = 0) | | | | | | | | | | | | | | | |
| *Optional words of SEQ IN PTR command:* | | | | | | | | | | | | | | | | |
| This pointer is omitted if PRE=1 or RTO=1 | Pointer (one or two words; see Address pointers) | | | | | | | | | | | | | | | |
| This field is omitted if EXT=0 | EXT_LENGTH (present if EXT = 1) (one word) | | | | | | | | | | | | | | | |

**Table 7-97. SEQ IN PTR command, field descriptions**

| Field | Description |
|---|---|
| 31-27<br>CTYPE | Command Type<br>If CTYPE=11110b : SEQ IN PTR command |
| 26<br>RBS | Release Buffers<br>If RBS=0 : Do not release buffers. |

*Table continues on the next page...*

## Table 7-97. SEQ IN PTR command, field descriptions (continued)

| Field | Description |
|---|---|
| | If RBS=1 : Release buffers from input frame. DECO releases buffers from the input frame after using them. It is an error if RBS = 1 when the Queue Manager Interface is not the job source. DECO releases only data buffers and scatter/gather tables referenced, directly or indirectly, by a top-level scatter/gather table but does not release the top-level scatter/gather table. If the table was specified by the original job descriptor's SEQ IN PTR command, then the top-level scatter/gather table will be released following the completion of the job by the Queue Manager Interface. If the input frame is not from a scatter/gather table, it is handled in the same manner as the top-level scatter/gather table. |
| 25<br>INL | In-Line Descriptor. This specifies that a new descriptor is to be found at the start of the data pointed to by the sequence.<br>If INL=0 : No in-line descriptor<br>If INL=1 : In-line descriptor present. An in-line descriptor is found at the start of the data pointed to by the sequence. DECO reads that descriptor (which must not have a shared descriptor) and then executes it. Therefore, a SEQ IN PTR with INL = 1 is the last command that is executed in the current descriptor.<br>If the INL bit is 1 and the current Input Sequence length is not as large as the in-line descriptor, an error is flagged. Note that it is an error for INL and RJD to both be 1.<br>See Using in-line descriptors for more information. |
| 24<br>SGF | Scatter/Gather Table Flag.<br>If SGF=0 : Pointer points to actual data.<br>If SGF=1 : Pointer points to a scatter/gather table. SGF is ignored if RTO=1. |
| 23<br>PRE | Previous. Add more length to the previously specified length of the input sequence.<br>If PRE=0 : The sequence pointer is set to the value of the Pointer and the input sequence length is set to the value specified in the LENGTH or EXT LENGTH field.<br>If PRE=1 : Command has no pointer field, and the specified length (LENGTH or EXT LENGTH) is added to the current input sequence length.<br>It is an error for the PRE bit and the RTO bit to both be set. |
| 22<br>EXT | Extended Length<br>If EXT=0 : Input data length value is in the 16-bit LENGTH field in the first word of the command (before the pointer). The EXT LENGTH field is omitted from the command.<br>If EXT=1 : Input data length value is in the 32-bit EXT LENGTH following the pointer. The 16-bit LENGTH field is ignored. |
| 21<br>RTO | Restore. Used to restore an input sequence.<br>If RTO=0 : Do not restore.<br>If RTO=1 : Restore. This command has no POINTER field. The length specified in LENGTH or EXT LENGTH is added to the current Input Sequence length. The original sequence address and RBS and SGF bits are automatically restored. The intended use is to be able to go back to the beginning of a sequence to reprocess some or all of the data.<br>It is an error for the PRE bit and the RTO bit to both be set. SGF is ignored if RTO=1. |
| 20<br>RJD | Replacement Job Descriptor<br>If RJD=0 : Don't replace job descriptor<br>If RJD=1 : Replace job descriptor. If there is no shared descriptor, and CTRL=0, this is synonymous with the INL bit (that is, setting either bit yields the same result). However, if there is a shared descriptor, setting the RJD bit causes the job descriptor to be replaced without affecting the shared descriptor, which will have already been loaded. See Using replacement job descriptors for more information. It is an error if both RJD = 1 and INL = 1. |

*Table continues on the next page...*

Table 7-97. SEQ IN PTR command, field descriptions (continued)

| Field | Description | |
|---|---|---|
| | **NOTE:** See the description below for the CTRL bit to understand how that bit can modify the above behavior. | |
| 19<br>SOP | Sequence Out Pointer<br>If SOP=0 : This bit has no effect.<br>If SOP=1 : Start a new input sequence using the pointer and SGF bit used when the current output sequence was defined. (If there was no previous sequence, behavior is undefined.) The length used is the length that has already been written to the current output sequence. This functionality is used when a multi-pass operation is required. The results of the first pass are stored in the output frame. By using the SOP bit, the SEQ IN PTR command allows the second pass to reference the results of the first pass.<br>It is an error to assert SOP if RBS, PRE, EXT or RTO are set. SGF and LENGTH are ignored. | |
| 18<br>CTRL | CTRL. This bit is used in conjunction with the RJD bit to differentiate between a normal RJD and a control RJD. See Using replacement job descriptors for more information.<br>If CTRL=0 and RJD=0 : This bit has no effect.<br>If CTRL=0 and RJD=1 : The new descriptor is the next data to be read from the input frame.<br>If CTRL=1 and RJD=0 : An error will be thrown.<br>If CTRL=1 and RJD=1 : The new descriptor is found following the shared descriptor in memory. If there is no shared descriptor, an error will be thrown. | |
| 17 | Reserved | |
| 16 | Reserved | |
| 15-0<br>LENGTH | LENGTH. This is the length of the input frame.<br>If EXT = 0 : The LENGTH field specifies the number of bytes in (or to be added to) the input sequence. The Extended Length word is omitted.<br>If EXT = 1 : The number of bytes in (or to be added to) the input sequence is specified in the Extended Length field. The LENGTH field is ignored. | |
| *Optional words of SEQ IN PTR command:* | | |
| 31-0<br>POINTER | Pointer<br>Specifies the starting address for an Input Sequence. See Address pointers. | *If PRE = 1,RTO = 1, or SOP = 1, this field is omitted.* |
| 31-0<br>EXT_LENGTH | Extended Length Field<br>If EXT = 0 : This field not present.<br>If EXT = 1 : The EXT LENGTH field specifies the number of bytes in (or to be added to) the Input Sequence. | *If EXT = 0, this field is omitted.* |

## 7.23 SEQ OUT PTR command

The Sequence Out Pointer (SEQ OUT PTR) command is used to specify the starting address for an output sequence and the length of that sequence (see SEQ vs non-SEQ commands). Only one output sequence may be active within the DECO at any one time.

An output sequence is initiated by executing a SEQ OUT PTR command with PRE = 0. This causes the following:

- The starting address of the output sequence to be set to the value in the Pointer field or to the original pointer if rewinding.
- The Sequence Out Length register to be set to the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1). If rewinding, the LENGTH or EXT_LENGTH field is added to the current length if REW = 10b and is ignored if REW = 11b.

If the EXT bit is 0, the EXT LENGTH field is omitted from the SEQ OUT PTR command.

The output sequence terminates when one of the following occurs:

- An error
- A new output sequence is started by executing a SEQ OUT PTR command with PRE = 0.

To extend the length of the sequence, any number of additional SEQ OUT PTR commands may be executed with PRE = 1. If PRE = 1, the value in the LENGTH field (if EXT = 0) or the EXT LENGTH field (if EXT = 1) is added to the current value in the Sequence Out Length register, but the address for the output sequence is unaffected. In this case, the SEQ OUT PTR command does not include a Pointer field. Additional length may also be added via the MATH and MATHI commands.

If the same output data needs to be processed again, the output pointer can be restored to the original starting address by executing a SEQ OUT PTR using the REW field. The SEQ OUT PTR command does not include a Pointer field in this case.

**Table 7-98. SEQ OUT PTR command, format**

| | 31–27 | 26 | 25 | 24 | 23 | 22 | 21-20 | 19 | 18-17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|
| | CTYPE = 11111 | Reserved | | SGF | PRE | EXT | REW | EWS | Reserved | Reserved |
| | 15–0 | | | | | | | | | |
| | LENGTH (used if EXT = 0) | | | | | | | | | |
| *Additional words of SEQ OUT PTR command:* | | | | | | | | | | |
| *This pointer is omitted if PRE=1 or if rewinding* | Pointer (one or two words; see Address pointers) | | | | | | | | | |
| *This word is omitted if EXT=0* | EXT LENGTH (used if EXT = 1) | | | | | | | | | |

### Table 7-99. SEQ OUT PTR command, field descriptions

| Field | Description | |
|---|---|---|
| 31-27<br>CTYPE | Command Type.<br>If CTYPE=11111b : SEQ OUT PTR command | |
| 26-25 | Reserved | |
| 24<br>SGF | If SGF=0 : Pointer points to actual data.<br>If SGF=1 : Pointer points to a scatter/gather table. | |
| 23<br>PRE | Previous. Add more length to the previously specified length of the Output Sequence.<br>If PRE=0 : The sequence pointer is set to the value of the pointer and the Output Sequence Length is set to the value specified in the LENGTH or EXT LENGTH field.<br>If PRE=1 : The SEQ OUT PTR command has no pointer field, and the specified length (LENGTH or EXT LENGTH) is added to the current Output Sequence Length.<br>Note that it is an error if PRE = 1 and REW = 10b or 11b. | |
| 22<br>EXT | Extended Length<br>If EXT=0 : The output data length value is in the 16-bit LENGTH field in the first word of the command (before the pointer). The EXT LENGTH field is omitted from the command.<br>If EXT=1 : The output data length value is in the 32-bit EXT LENGTH field following the pointer. The 16-bit LENGTH field is ignored. | |
| 21-20<br>REW | Rewind. Used to rewind an Output Sequence.<br>If REW = 00b : Do not rewind.<br>If REW = 01b : Error<br>If REW = 10b : Rewind. This command has no POINTER field. The length specified in LENGTH or EXT LENGTH is added to the current sequence output length. The original sequence address and SGF bit are automatically restored. This allows returning to the beginning of a sequence to reprocess some or all of the data. DECO automatically disables the counting of bytes written to the output frame. In order to re-enable counting, use a write to the DECO CTRL Register.<br>If REW = 11b : Rewind and Reset. The same as 10b, except that any length provided is ignored, the current output frame length is added back to the SOL (sequence output length) register and the tracking length [1] of bytes written to the output frame is reset to 0. Care must be taken if the descriptor has modified the SOL register other than as a result of decrements caused by SEQ STORE and SEQ FIFO STORE commands. Since the number of bytes written to the output frame has been reset, counting such bytes remains enabled in this case.<br>The REW = 10b or 11b functionality is used when a multi-pass operation is required. The results of the first pass are stored in the output frame. Executing the SEQ OUT PTR command with REW = 11b allows the second pass to start from the beginning of the output frame as if this were the original output stream. That way the final status reported back contains the correct length. | |
| 19<br>EWS | Enable Write Safe.<br>When this bit is set, write-safe bus transactions are permitted for this output sequence. See AXI master (DMA) interface. | |
| 18-16 | Reserved | |
| 15-0<br>LENGTH | If EXT = 0 : The LENGTH field specifies the number of bytes in (or to be added to) the output sequence.<br>If EXT = 1 : The LENGTH field is ignored. | |
| | *Optional words of SEQ OUT PTR command:* | |
| POINTER | Pointer. Specifies the starting address for an Output Sequence. See Address pointers. | If PRE = 1 or REW != 00b, this field is omitted. |

*Table continues on the next page...*

## Table 7-99. SEQ OUT PTR command, field descriptions (continued)

| Field | Description | |
|---|---|---|
| One word<br>EXT LENGTH | If EXT = 0 : The EXT LENGTH field is omitted.<br>If EXT = 1 : The EXT LENGTH field specifies the number of bytes in (or to be added to) the output sequence. | If EXT = 0, this field is omitted. |

1. DECO tracks how many bytes have been written to the output frame so that this number can be part of the status reported when a job completes.

# Chapter 8
# Public Key Cryptography Operations

SEC implements, through protocol commands, a number of public (and private) key functions. These are:

- DSA and ECDSA sign/verify
- Diffie-Hellman (DH) and ECDH key agreement
- ECC key generation (for ECDH, ECDSA, etc.)
- ECC public key validation
- DLC key generation for DH, DSA
- RSA public-key and private-key primitives, for use with RSA encryption/decryption and RSA signature generation/verification
- RSA key-generation filnalization

SEC also contains a hardware block, Public-key hardware accelerator (PKHA) functionality, which can be programmed directly for public key calculations.

## 8.1 Conformance considerations

The DSA and ECDSA key-generation, signing, verification, and Diffie-Hellman functions described are intended to conform to the following specifications (except where noted). For more information refer to the NIST Cryptographic Algorithm Validation Program (CAVP) Certifications whitepaper, www.nxp.com/security, or consult these standards:

- FIPS PUB 186-4, *Digital Signature Standard (DSS)*, July 2013
- NIST SP800-90A, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, January 2012
- IEEE1363-2000, *IEEE Standard Specifications for Public-Key Cryptography*, January 30, 2000
- ANSI X9.42-2003, *Public Key Cryptography for the Financial Services Industry, Agreement of Symmetric Keys Using Discrete Logarithm Cryptography*, November 19, 2003

- ANSI X9.63-2001, *Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography*, November 20, 2001
- ANSI X9.62-2005, *Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, November 16, 2005

The notation used is from IEEE 1363-2000 because only that document provides a set of variable names and definitions consistent between both DSA and ECDSA.

Private keys for DSA and ECDSA, (as well as per-message secrets), are generated using the method of extra random bits, equivalent to that described in FIPS 186-4, (Appendix B.1.1). In B.1.1, $c$ is a string of random bits, 64 bits longer than requested.

Then $x = (c \bmod (q - 1)) + 1$

SEC uses the following equivalent version.

$x = c \bmod q$; if ($x = 0$), choose another $c$

In both cases, $x$ is uniformly distributed in the range [1, $q$-1].

Binary (aka Characteristic 2 or $F_{2^m}$) Elliptic Curves inputs and outputs are in polynomial basis and in affine (x, y) coordinates.

Assurances for the validity of all domain parameters and public keys must be obtained before invoking any of these functions. These functions assume that all domain parameters and public keys are valid and are associated with each other.

## 8.2 Discrete-log key-pair generation

Some important characteristics and requirements of discrete-log key-pair generation is as follows:

- DL KEY PAIR GEN is used to generate public key-pairs. There are four variations to generate either prime field or binary field keys for either DSA or ECDSA.
- Each of the public key functions writes out the private key, followed by the public key.
- DL KEY PAIR GEN requires the parameters listed in the following table. Note that $G_{x,y}$ and $W_{x,y}$ are pointers to input buffers containing both an $x$ and $y$ coordinate. The two coordinates must be the same length.
- There are two parameter lengths, size of the field (L), and size of the group or private key modulus (N). These represent the size of the buffers, in bytes, required to hold the input and output data, (not the bit lengths of the various parameters). Note that

the size of the buffers for $G_{x,y}$, $W_{x,y}$ and *a,b* must be twice L, as each holds two values of size L.

**Table 8-1. Public key-generation parameters**

| Parameter | Input/output | Length | Definition |
|---|---|---|---|
| *q* | Input | L | Prime number or irreducible polynomial that creates the field |
| *r* | Input | N | Order of the field of private keys or modulus for creating private keys |
| *a,b* | Input | 2L | ECC curve parameters. For binary field curves, b' rather than b is used. (ECC only.) |
| *g* or $G_{x,y}$ | Input | L or 2L | Generator or generator point (ECC) |
| *s* | Output | N | Private key |
| *w* or $W_{x,y}$ | Output | L or 2L | Public key |

## 8.2.1 Inputs to the discrete-log key-pair generation function

- For DSA, the domain parameters *q*, *r*, and *g*
- For ECDSA, the domain parameters *q*, *a, b or b',r*, and $G_{x,y}$

## 8.2.2 Assumptions of the discrete-log key-pair generation function

- The domain parameters are valid and are associated with each other (that is, parameter validation must be done prior to using this function).
- If the ENC bit of the Protocol Command register is set, s is treated as an encrypted key and is encrypted before being written out. When generating an encrypted key, the buffer must be large enough to hold the black key, i.e., the encrypted version of the key.

## 8.2.3 Outputs from the discrete-log key-pair generation function

- The signer's private key *s*
- For DSA, the signer's public key w
- For ECDSA, the signer's public key $W_{x,y}$

## 8.2.4 Operation of the discrete-log key-pair generation function

- Generate a private key *s*, in the range $1 \le s < r$. (Generate a random number *k*, 64 bits larger than *r*, and find $s = k \bmod r$. If $s = 0$, generate a new *k*.)

- Compute $w = g^s \bmod q$, or $W_{x,y} = sG_{x,y}$.
- Output ($s$, $w$) or ($s$, $W_{x,y}$) as the private and public keys.

### 8.2.5 Notes associated with the discrete-log key-pair generation function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} mod\ q$ must be given, rather than $b$.

For IETF DH involving domains like MODP Groups 5, 14, 15, and 16, there is no published $r$ value. However, a value is necessary for this function, as it is the modulus used to create the private key, where 1 (< private_key < mod). The value of $N$ should be determined based upon the desired strength of the private key; there are recommendations in the IETF RFCs and elsewhere. Both $r$ and the private key will be $N$ bytes long. A typical value for $r$ would be a string containing $N$ bytes of FFh.

When the PD (Predefined Domain) bit in the PDB is 1, the ECDSEL (Elliptic Curve Domain Selection) field is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in Table 8-5.

**Table 8-2. Public-key generation protocol data block (PD=0 version)**

| SGF<br>(6 bits) | PD=0<br>(1 bit) | Reserved<br>(8 bits) | L<br>(10 bits) | N<br>(7 bits) |
|---|---|---|---|---|
| Pointer to q | | | | |
| Pointer to r | | | | |
| Pointer to g (DSA) or Gx,y (ECC) | | | | |
| Pointer to s | | | | |
| Pointer to w (DSA) or Wx,y (ECC) | | | | |
| Pointer to a,b (ECC only)<br>(The protocol data block for DSA is shorter than for ECDSA, because the pointer to a,b is not required.) | | | | |

**Table 8-3. Public-key generation protocol data block (PD=1 version)**

| SGF<br>(6 bits) | PD=1<br>(1 bit) | Reserved<br>(8 bits) | Reserved<br>(3 bits) | ECDSEL<br>(7 bits) | Reserved<br>(7 bits) |
|---|---|---|---|---|---|
| Pointer to s | | | | | |
| Pointer to w or Wx,y | | | | | |

For both PDB versions the format of the SGF field is illustrated in this figure.

Table 8-4. Public key-generation PDB - format of the SGF field

| | 31 | 30 | 29 | 28 | 27 | 26 |
|---|---|---|---|---|---|---|
| Version when PD=0 | ref q | ref r | ref Gx,y | ref s | ref w or Wx,y | a,b (ECC)<br>reserved (DSA) |
| Version when PD=1 | Reserved | | | ref s | ref Wx,y | Reserved |
| **NOTE:** If the SGF bit is set the argument is referenced via a scatter/gather table. If the SGF bit is not set the argument is referenced via a direct-address pointer. | | | | | | |

When PD=1, a valid value must appear in the ECDSEL (Elliptic Curve Domain Selection) field. A list of the values that may be used in the ECDSEL field, and their meanings, is given in the table below.

The following variable definitions apply to the following table. Variable names (*q, r, b, c*) follow the conventions of IEEE Std 1363.

**Name**

The names in this table are associated with, or named in, various published standards. Neither the names nor the domains are guaranteed to be complete. Two values of the domain parameters are provided for purposes of identification.

- Those beginning with "P-", "K-", and "B-" are in FIPS 186 from NIST, found at www.csrc.nist.gov
- Those beginning with "ansix9" are names from ANS X9.62-2005; those beginning with "prime" or "c2pnb" are from an earlier ANSI document
- Those beginning with "sec" are from SEC 2 from the Standards for Efficient Cryptography group, found at www.secg.org
- Those beginning with "wtls" are taken from Wireless Transport Layer Security / Wireless Access Protocol, Version 06-Apr-2001, WAP-261-WTLS-20010406-a. Not all software libraries agree with the mapping of these names to values; care has been taken to identify the values based upon the source documentation.
- Those beginning with "ECDSA", "ECP", "EC2N", "ecp_group", and "Oakley" are from various RFCs found at www.ietf.org
- Those beginning with "GOST" are from the Russian standard GOST R 3410-2001
- Those beginning with "brainpool" are from ECC Brainpool, found at www.ecc-brainpool.org and republished in RFC 5639

**q**

This is the *field-defining* value for the elliptic curve. For $F_p$ curves, it is the prime number used as the modulus for all point arithmetic; it is named *p* in some other publications. For $F_{2^m}$ curves, it is the irreducible binary polynomial used as the modulus for all point arithmetic. It is not, as usually defined, $q = 2^m$, i.e. the size of the field.

**L**

This is the number of bytes needed to hold *q* and each of its associated values:, *a*,*b*,*c*, the point coordinates *x* and *y*, the result of an ECDH key agreement, etc.

**r**

This is the (usually prime) number which is the order of *G*, the generator point. It is also usually used as the modulus for the non-ECC-related arithmetic in an ECC primitive. This variable is named *n* in some other publications.

**N**

This is the number of bytes needed to hold *r* and each of its associated values: private keys, each of the two components of an ECDSA signature, etc.

**a**

This variable, along with *q* and *b*, define the elliptic curve. For Fp, *a* is the coefficient for the *x* term. For $F_{2^m}$, it is the coefficient for the $x^2$ term.

**b / c (b')**

*b* is the coefficient for the $x^0$ (ones) term in an $F_{2^m}$ elliptic curve equation. Its relationship with *c* is $b = c^4$. *c* is sometimes referred to as *b'* in NXP documentation.

### Table 8-5. ECDSEL field values for built-in ECC $F_p$ domains

| When PD=1 in the first word of the PDB, the ECDSEL field specifies one of the built-in ECC domains. The valid values for the ECDSEL field and the name of the ECC domain are listed in this table. The domains are ordered by size. | |
|---|---|
| **Value** | **Name(s)** |
| **ECC $F_p$ domains** | |
| 00h | P-192, secp192r1, ansix9p192r1, prime192v1, ECPRGF192Random |
| 01h | P-224, secp224r1, ansix9p224r1, wtls12, ECPRGF224Random |
| 02h | P-256, secp256r1, ansix9p256r1, prime256v1, ECDSA-256, ecp_group_19, ECPRGF256Random |
| 03h | P-384, secp384r1, ansix9p384r1, ECDSA-384, ecp_group_20, ECPRGF384Random |
| 04h | P-521, secp521r1, ansix9p521r1, ECDSA-521, ecp_group_21, ECPRGF521Random |
| 05h | brainpoolP160r1 |
| 06h | brainpoolP160t1 |
| 07h | brainpoolP192r1 |
| 08h | brainpoolP192t1 |
| 09h | brainpoolP224r1 |
| 0Ah | brainpoolP224t1 |
| 0Bh | brainpoolP256r1 |
| 0Ch | brainpoolP256t1 |
| 0Dh | brainpoolP320r1 |
| 0Eh | brainpoolP320t1 |
| 0Fh | brainpoolP384r1 |
| 10h | brainpoolP384t1 |
| 11h | brainpoolP512r1 |
| 12h | brainpoolP512t1 |
| 13h | prime192v2 |

*Table continues on the next page...*

### Table 8-5. ECDSEL field values for built-in ECC $F_p$ domains (continued)

| When PD=1 in the first word of the PDB, the ECDSEL field specifies one of the built-in ECC domains. The valid values for the ECDSEL field and the name of the ECC domain are listed in this table. The domains are ordered by size. | |
|---|---|
| **Value** | **Name(s)** |
| **ECC $F_p$ domains** | |
| 14h | prime192v3 |
| 15h | prime239v1 |
| 16h | prime239v2 |
| 17h | prime239v3 |
| 18h | secp112r1, wtls6 |
| 19h | wtls8 |
| 1Ah | wtls9 |
| 1Bh | secp160k1, ansix9p160k1 |
| 1Ch | secp160r1, ansix9p160r1, wtls7 |
| 1Dh | secp160r2, ansix9p160r2 |
| 1Eh | secp192k1, ansix9p192k1 |
| 1Fh | secp224k1, ansix9p224k1 |
| 20h | secp256k1, ansix9p256k1 |
| ECC $F_{2^m}$ domains | |
| 40h | B-163, ansix9t163r2, sect163r2, EC2NGF163Random |
| 41h | B-233, sect233r1, ansix9t233r1, EC2NGF233Random, wtls11 |
| 42h | B-283, sect283r1, ansix9t283r1, EC2NGF283Random |
| 43h | B-409, sect409r1, ansix9t409r1, EC2NGF409Random |
| 44h | B-571, sect571r1, ansix9t571r1, EC2NGF571Random |
| 45h | K-163, ansix9t163k1, sect163k1, EC2NGF163Koblitz, wtls3 |
| 46h | K-233, sect233k1, ansix9t233k1, EC2NGF233Koblitz, wtls10 |
| 47h | K-283, sect283k1, ansix9t283k1, EC2NGF283Koblitz |
| 48h | K-409, sect409k1, ansix9t409k1, EC2NGF409Koblitz |
| 49h | K-571, sect571k1, ansix9t571k1, EC2NGF571Koblitz |
| 4Ah | wtls1 |
| 4Bh | sect113r1, wtls4 |
| 4Ch | c2pnb163v1, wtls5 |
| 4Dh | c2pnb163v2 |
| 4Eh | c2pnb163v3 |
| 4Fh | sect163r1, ansix9t163r1 |
| 50h | sect193r1, ansix9t193r1 |
| 51h | sect193r2, ansix9t193r2 |
| 52h | sect239k1, ansix9t239k1 |
| 53h | Oakley 3, ec2n_group_3 |
| 54h | Oakley 4, ec2n_group_4 |

## 8.3 Using the Diffie_Hellman function

Diffie-Hellman is used in key exchange and key agreement schemes. For example, Internet Key Exchange (IKE) specifies a pseudo-random function (PRF) that takes the result of a Diffie-Hellman operation as an input. Because the output of Diffie-Hellman is a secret value, it is advisable to store the output in encrypted form. SEC's Diffie-Hellman protocol provides this option, and SEC's IKE PRF protocol can read the secret in encrypted form.

Diffie-Hellman is defined for both discrete log (DH) and elliptic-curve (ECDH) forms. SEC provides acceleration support for both forms.

### 8.3.1 Diffie_Hellman requirements

Diffie-Hellman requires the parameters listed in this table.

**Table 8-6. Required Diffie-Hellman parameters**

| Parameter | Input/Output | Length | Definition |
|---|---|---|---|
| L | input | 10 bits | Number of bytes of the the field |
| N | input | 7 bits | Number of bytes of the private key |
| *q* | input | L | Prime number or irreducible polynomial that creates the field |
| *r* | input | - | Unused for Diffie-Hellman |
| *a,b* | input | 2L | ECC curve parameters. For binary field curves, b' rather than b is used. (ECC only.) |
| *w´* or $W'_{x,y}$ | input | L (DH) or 2L (ECDH) | Other party's public key |
| *s* | input | N | Own private key |
| *z* | output | L | Shared secret value |
| **NOTE:** Wx,y is a pointer to an input buffer containing both an x and a y coordinate. The two coordinates must be the same length.<br>There are two parameter lengths, size of the field (L), and the size of the private key (N). These represent the size of the buffers, in bytes, required to hold the input and output data.<br>The size of the buffers for $G_{x,y}$, $W_{x,y}$, and a,b must be twice L, as each holds two values of size L. | | | |

### 8.3.2 Inputs to the Diffie-Hellman function

- For discrete logs, the domain parameters q, s (own private key), and w' (other's public key).
- For elliptic curve, the domain parameters q, s (own private key), and $W'_{x,y}$ (other's public key), a and b (or b').

Note that the domain parameters r and g (or $G_{x,y}$) are not used.

### 8.3.3 Assumptions of the Diffie-Hellman function

- The domain parameters are valid and are associated with each other (that is, parameter validation must be done prior to using this function).
- If the ENC_PRI bit of the Protocol Information register is set, s is treated as an encrypted key and is decrypted after being read. If the ENC_PUB bit of the protocol information is set, then z is encrypted before being written.

### 8.3.4 Outputs from the Diffie-Hellman function

The shared secret value z

### 8.3.5 Operation of the Diffie-Hellman function

- Read in the private key pointed to by s.
- For DL, compute $z = w^s \bmod q$.
- For ECC, compute new_point = s * W, and output z = x coordinate of new_point
- Output z as the shared secret.

### 8.3.6 Notes associated with the Diffie-Hellman function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \bmod q$ must be given rather than $b$. For a detailed explanation, see Point math over a binary field ($F_{2^m}$)

**Table 8-7. Diffie-Hellman protocol data block**

| SGF<br>(6 bits) | Reserved<br>(9 bits) | L<br>(10 bits) | N<br>(7 bits) |
|---|---|---|---|
| Pointer to *q* | | | |
| Pointer to *r* (unused) | | | |

*Table continues on the next page...*

**Table 8-7. Diffie-Hellman protocol data block (continued)**

| |
|---|
| Pointer to *w* or *Wx,y* |
| Pointer to *s* |
| Pointer to *z* |
| Pointer to *a,b* (ECC only) |

For discrete log Diffie-Hellman, the pointer to a,b is not required. The following figure illustrates the format of the SGF field.

**Table 8-8. Diffie-Hellman PDB-format of the SGF field**

| 31 | 30 | 29 | 28 | 27 | 26 |
|---|---|---|---|---|---|
| ref q | ref r (unused) | ref w or Wx,y | ref s | ref z | a,b (ECC only) |
| **NOTE:** If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer. | | | | | |

## 8.4 Generating DSA and ECDSA signatures

DSA_SIGN is SEC's hardware implementation of NIST's DSA (Digital Signature Algorithm) and ECDSA digital signing functions. It supports DSA and ECDSA in both prime fields and binary fields. These functions can take either a message or a message representative as input, controlled by the MSG_REP bit in the OPERATION Command register.

There are two parameter lengths: size of the field (L), and size of the group (N). These represent the size of the buffers, in bytes, required to hold the input and output data, (not the bit lengths of the various parameters). Note that the size of the buffers for $G_{x,y}$ and *a,b* must be twice L, as each holds two values of size L.

This table lists the DSA and ECDSA sign protocol parameters.

**Table 8-9. DSA and ECDSA sign parameters**

| Parameter | Input/Output | Length | Definition |
|---|---|---|---|
| *q* | input | L | Prime number or irreducible polynomial that creates the field |
| *r* | input | N | Order of the field of private keys |
| *a, b* | input | 2L | ECC curve parameters. For binary field curves, b' rather than b is given. (ECC only.) |
| *g* or $G_{x,y}$ | input | L (DSA),<br>2L (ECDSA) | Generator or generator point (ECC) |
| *s* | input | N | Private key |

*Table continues on the next page...*

Table 8-9. DSA and ECDSA sign parameters (continued)

| Parameter | Input/Output | Length | Definition |
|---|---|---|---|
| *f (or m)* | input | N | Message representative (typically the hash of the message) or the actual message |
| *c* | output | N | First part of digital signature |
| *d* | output | N | Second part of digital signature. The buffer for *d* must be a multiple of 16 bytes, as it is used to store an encrypted intermediate result, which may include padding. |
| u | output | N | Per message random number, only in TEST mode |

### 8.4.1 Inputs to the DSA and ECDSA signature generation function

- For DSA, the domain parameters *q*, *r*, and *g* associated with key *s*.
- For ECDSA, the domain parameters *q*, *r*, *g*, *a* and *b* associated with key *s*.
- The signer's private key *s*.
- The message representative, which is an integer $f \geq 0$, or the message itself (which is hashed to form a message representative).

### 8.4.2 Assumptions of the DSA and ECDSA signature generation function

- The private key *s* is in the range $1 \leq s < r$, and the domain parameters are valid and are associated with each other, (that is, parameter validation must be done prior to using this function).
- The message representative, *f*, is generated using an approved hashing function of the appropriate security strength.
- If the ENC bit of the Protocol Command is set, then *s* is treated as an encrypted key, and is decrypted before it is used.

### 8.4.3 Outputs from the DSA and ECDSA signature generation function

When running the full signature opeartion, the output is a pair of integers (*c*, *d*), where $1 \leq c < r$ and $1 \leq d < r$

When just the first part of the signature is generated the output is the integer $c$, as above, and the encrypted version of the (inverted) temporary key used in the creation of the signature. It is stored at $d$, and the memory there must have space for the ECB-encrypted version (i.e, rounded up to the nearest 16 bytes).

When just the second part of the signature is generated the output is $d$, as in the output of the complete signature operation.

## 8.4.4 Operation of the DSA and ECDSA signature generation function

- Generate a per message private key $u$, in the range $1 \le u < r$. (Generate a random number $k$, 64 bits larger than $r$, and find $u = k \bmod r$. If $u = 0$, generate a new $k$.)
- Compute $c = (g^u \bmod q) \bmod r$, or $V_{x,y} = uG_{x,y}$, $c = V_x \bmod r$. If $c = 0$, try again with a new $u$.
- Compute $d = u^{-1}(f + sc) \bmod r$. If $d = 0$, try again with a new $u$.
- Output $(c, d)$ as the signature.
- If the TEST bit of the Protocol Command is set, then also output $u$. This test mode is not accessible in the Trusted or Secure states.

## 8.4.5 Notes associated with the DSA and ECDSA Signature Generation function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \bmod q$ must be given, rather than $b$.

The beginning of the descriptor contains a protocol data block that specifies the sizes of arguments to the Signature Generation function and pointers to those arguments. Each pointer occupies one word of the PDB if MCFGR[PS]=0, or two words of the PDB if MCFGR[PS]=1.

When the PD (Predefined Domain) bit in the PDB is 1, the ECDSEL (Elliptic Curve Domain Selection) field is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in Table 8-5. Note that if PD=1 for a DSA operation, a PDB error will be generated.

**Table 8-10. DSA and ECDSA Signature Generation protocol data block**

| The format of PDB Word 1 depends on the value in the PD field. | | | | | |
|---|---|---|---|---|---|
| PDB Word 1 | SGF (see format in table below)<br>(9 bits) | **PD=0**<br>(1 bit) | Reserved<br>(5 bits) | L<br>(10 bits) | N<br>(7 bits) |

Table continues on the next page...

## Table 8-10. DSA and ECDSA Signature Generation protocol data block (continued)

| | | | | | | |
|---|---|---|---|---|---|---|
| | SGF (see format in table below)<br>(9 bits) | **PD=1**<br>(1 bit) | Reserved<br>(5 bits) | Reserved<br>(3 bits) | ECDSEL<br>(7 bits) | Reserved<br>(7 bits) |

## Table 8-11. DSA and ECDSA Signature Generation protocol data block, continued

| Following PDB Word 1 the PDB contains a series of address pointers for arguments to the Sign Protocol operation. Each pointer occupies two words. If the pointer's SGF bit is 0 this is the 40-bit address of the argument data itself. If the pointer's SGF bit is 1 this is the 40-bit address of a Scatter Gather Table that references the argument data. The format of the SGF field is illustrated in the table below. Note that different arguments pointers are included or omitted depending upon the value in the PD field and the type of Sign operation (full, 2nd half, 1st half). The six possible arrangements of the pointers are shown in the rightmost six columns below. | | | | | | |
|---|---|---|---|---|---|---|
| | **PD=0** | | | **PD=1** | | |
| | Full Sign operation (1st Half Sign **and** 2nd Half Sign) | 2nd Half Sign operation<br>but not 1st Half Sign | 1st Half Sign operation<br>but not 2nd Half Sign | Full Sign operation (1st Half Sign **and** 2nd Half Sign) | 2nd Half Sign operation<br>but not 1st Half Sign | 1st Half Sign operation<br>but not 2nd Half Sign |
| PDB pointer 1 | *q* | *r* | *q* | *s* | *s* | *c* |
| PDB pointer 2 | *r* | *s* | *r* | *f* (MSG_REP=0)<br>or *m* (MSG_REP=1) | *f* (MSG_REP=0)<br>or *m* (MSG_REP=1) | *d* |
| PDB pointer 3 | *g* (DSA)<br>or *Gx,y* (ECDSA) | *f* (MSG_REP=0)<br>or *m* (MSG_REP=1) | *g* (DSA)<br>or *Gx,y* (ECDSA) | *c* | *c* | See note 4 |
| PDB pointer 4 | *s* | *c* | *c* | *d* | *d* | |
| PDB pointer 5 | *f* (MSG_REP=0)<br>or *m* (MSG_REP=1) | *d* | *d* | See notes 2 and 7 | See notes 3 and 7 | |
| PDB pointer 6 | *c* | See notes 3 and 7 | See note 5 | See notes 3 and 7 | | |
| PDB pointer 7 | *d* | | See note 6 | | | |
| PDB pointer 8 | See notes 1 and 7 | | | | | |
| PDB pointer 9 | See notes 8 and 7 | | | | | |
| PDB pointer 10 | See notes 9 and 7 | | | | | |
| Note 1: If ECDSA, this is a pointer to *a,b*, else if TEST=1, this is a pointer to *u*, else if MSG_REP=1, this is the message length, else the previous word is the last word of the PDB.<br>Note 2: If TEST=1, this is a pointer to *u*, else if MSG_REP=1, this word is the 32-bit message length, else the previous word is the last word of the PDB.<br>Note 3: If MSG_REP=1, this word is the 32-bit message length, else the previous word is the last word of the PDB.<br>Note 4: If TEST=1, this is a pointer to *u*, else the previous word is the last word of the PDBt. | | | | | | |

### Table 8-11. DSA and ECDSA Signature Generation protocol data block, continued

| |
|---|
| Note 5: If ECDSA, this is a pointer to *a,b*, else if TEST=1, this is a pointer to *u*, else the previous word is the last word of the PDB. |
| Note 6: If ECDSA **and** TEST=1, this is a pointer to *u*, else the previous word is the last word of the PDB. |
| Note 7: MSG_REP=1 means calculate the message representative from the message *m*. In this case the message length must be provided in one word of the PDB. MSG_REP=0 means don't calculate a message representation, use *f* as the message representative. In this case the word that would contain the message length is omitted from the PDB. |
| Note 8: If ECDSA **and** TEST=1, this is a pointer to *u*, else if (ECDSA **or** TEST=1) **and** MSG_REP=1, this word contains the message length, else the previous word is the last word of the PDB. |
| Note 9: If ECDSA **and** TEST=1 **and** MSG_REP=1, this word contains the message length, else the previous word is the last word of the PDB. |

This table shows the format of the SGF field.

### Table 8-12. DSA and ECDSA Signature Generation protocol data block - Format of the SGF Field

| **Formats for** | **Format for** | | bit number | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |
| **DSA or for ECDSA with PD=0** | **Full Sign** | **SGF bit for:** | q | r | g (DSA)<br>or<br>Gx,y (ECDSA) | s | f (MSG_REP=0)<br>or<br>m (MSG_REP=1) | c | d | a,b (ECDSA)<br>or<br>reserved (DSA) | u (TEST=1)<br>or<br>reserved (TEST=0) |
| | **1st Half Sign** | | q | r | g (DSA)<br>or<br>Gx,y (ECDSA) | reserved | reserved | c | d | a,b (ECDSA)<br>or<br>reserved (DSA) | u (TEST=1)<br>or<br>reserved (TEST=0) |
| | **2nd Half Sign** | | reserved | r | reserved | s | f (MSG_REP=0)<br>or<br>m (MSG_REP=1) | c | d | reserved | reserved |
| **ECDSA with PD=1** | **Full Sign** | **SGF bit for:** | reserved | reserved | reserved | s | f (MSG_REP=0)<br>or<br>m (MSG_REP=1) | c | d | reserved | u (TEST=1)<br>or<br>reserved (TEST=0) |

*Table continues on the next page...*

Table 8-12. DSA and ECDSA Signature Generation protocol data block - Format of the SGF Field (continued)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1st Half Sign** | | reserved | reserved | reserved | s | reserved | c | d | reserved | u (TEST=1)<br>or<br>reserved (TEST=0) |
| | **2nd Half Sign** | | reserved | reserved | reserved | reserved | f (MSG_REP=0)<br>or<br>m (MSG_REP=1) | c | d | reserved | reserved |
| | | | *If the SGF bit for an argument is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.* | | | | | | | | |

## 8.5 Verifying DSA and ECDSA signatures

DSA_VERIFY is the digital signature algorithm (DSA) verification function. It supports both DSA and ECDSA, in both prime fields and binary fields. These functions can take either a message or a message representative as input, controlled by the MSG_REP bit in the OPERATION command.

There are two parameter lengths:

- Size of the field (L)
- Size of the subgroup (N)

These are given in bytes, and denote the size of the buffer required to hold each parameter. Note that the size of the buffers for $G_{x,y}$, $W_{x,y}$ and *a,b* must be twice L, as each holds two values of size L.

Table 8-13. DSA and ECDSA Verify parameters

| Parameter | Input/Output | Length (bytes) | Definition |
|---|---|---|---|
| *q* | input | L | Prime number or irreducible polynomial that creates the field |
| *r* | input | N | Order of the subgroup of private keys |
| *a, b* | input | 2L | ECC curve parameters. For binary field curves, b' rather than b is used. (ECDSA only.) |
| *g* or $G_{x,y}$ | input | L (DSA),<br>2L (ECDSA) | Generator or generator point (ECDSA) |
| *w* or $W_{x,y}$ | input | L (DSA), | Public key |

*Table continues on the next page...*

**Table 8-13. DSA and ECDSA Verify parameters (continued)**

| Parameter | Input/Output | Length (bytes) | Definition |
|---|---|---|---|
| | | 2L (ECDSA) | |
| *f (or m)* | input | N | Message representative (typically the hash of the message) or the actual message |
| *c* | input | N | First part of digital signature |
| *d* | input | N | Second part of digital signature |
| Temp | input/output | L (DSA)<br>2L (ECDSA) | Temporary storage for intermediate results |

## 8.5.1 Inputs to the DSA and ECDSA signature verification function

- For DSA, the domain parameters $q$, $r$, and $g$ associated with key $w$
- For ECDSA, the domain parameters $q$, $r$, $G_{x,y}$, $a$ and $b$ associated with key $W_{x,y}$
- The signer's public key $w$ or $W_{\mathrm{x,y}}$
- The received message representative, which is an integer $f \geq 0$, or the message itself (which is hashed to form a message representative)
- The received signature to be verified, which is a pair of integers ($c$, $d$)

## 8.5.2 Assumptions of the DSA and ECDSA signature verification function

- The public key ($w$ or $W_{\mathrm{x,y}}$) and the domain parameters are valid and are associated with each other (that is, parameter validation must be done prior to using this function).
- The message representative, $f$, is generated using an approved hashing function of the appropriate security strength.

## 8.5.3 Outputs from the DSA and ECDSA signature verification function

- If the signature is correct, this function terminates normally.
- If the signature is not correct, this function terminates with an error code.

### 8.5.4 Operation of the DSA and ECDSA signature verification function

- Check that $c$ is in the range [1, $r$-1]. If not, terminate with error code invalid signature.
- Check that d is in the range [1, $r$-1]. If not, terminate with error code invalid signature.
- For DSA, compute $c' = ((G^{d^{-1}f} \bmod q)(w^{d^{-1}c} \bmod q)) \bmod r$.
- For ECDSA, compute $P_{x,y} = d^{-1}fG_{x,y} + d^{-1}cW_{x,y}$, and then if $P_{x,y}$ is the point at infinity, terminate with error code invalid signature, else let $c' = P_x \bmod r$.
- If $c' \neq c$, then terminate with error code invalid signature.
- Continue as valid.

### 8.5.5 Notes associated with the DSA and ECDSA Signature Verification function

For ECC binary field (F2M) functions, $b' = b^{2^{m-2}} \bmod q$ must be given, rather than $b$.

The beginning of the descriptor contains a protocol data block that specifies the sizes of arguments to the Signature Verification function and pointers to those arguments. Each pointer occupies one word of the PDB if MCFGR[PS]=0, or two words of the PDB if MCFGR[PS]=1.

Parameter information is as follows:

- L is the number of bytes in various data buffers. L the size of the prime number or irreducible polynomial representing the cryptographic field.
- N is another length, in number of bytes in data buffers. N is the size of the number representing the order of the subgroup of private keys within the field.
- All parameters are pointers to data buffers of size L or N, (or 2L for elliptic curve points and a,b).

The protocol data block for DSA is shorter than for ECDSA, as the pointer to a,b is absent.

A temporary buffer is required during the verification of the signature.

- For DSA, the temporary buffer must be at least L bytes.
- For ECDSA, the temporary buffer must be at least 2L bytes.

The ENC bit of PROTOINFO is ignored for signature verification because only public keys are used. If the MSG_REP bit of the OPERATION command is set to 1, the pointer to f (or m) points to the message to be signed, rather than to a message representative. The message length field of the protocol data block (only used when the MSG_REP bit of the OPERATION command is set to 1) defines the length of the message to be signed.

When the PD (Predefined Domain) bit in the PDB is 1, the ECDSEL (Elliptic Curve Domain Selection) field is used to select one of the built-in ECC domains. In this case most of the curve parameters are supplied by the hardware. The valid values for the ECDSEL field and their meanings are listed in Table 8-5. Note that if PD=1 for a DSA operation, a PDB error will be generated.

### Table 8-14. DSA and ECDSA Signature Verification protocol data block

| The format of PDB Word 1 depends on the value in the PD field. | | | | | | |
|---|---|---|---|---|---|---|
| PDB Word 1 | SGF (see format in table below)<br>(9 bits) | **PD=0**<br>(1 bit) | Reserved<br>(5 bits) | L<br>(10 bits) | | N<br>(7 bits) |
| | SGF (see format in table below)<br>(9 bits) | **PD=1**<br>(1 bit) | Reserved<br>(5 bits) | Reserved<br>(3 bits) | ECDSEL<br>(7 bits) | Reserved<br>(7 bits) |

### Table 8-15. DSA and ECDSA Signature Verification protocol data block, continued

| Following PDB Word 1 the PDB contains a series of address pointers for arguments to the Signature Verification function. Each pointer occupies two words. If the pointer's SGF bit is 0 this is the 40-bit address of the argument data itself. If the pointer's SGF bit is 1 this is the 40-bit address of a Scatter Gather Table that references the argument data. The format of the SGF field is illustrated in the table below. Note that different arguments pointers are included or omitted depending upon the value in the PD field and the type of Signature Verification operation (verification with public key, verification with private key). The four possible arrangements of the pointers are shown in the rightmost four columns below. | | | | |
|---|---|---|---|---|
| | **PD=0**(don't use a predefined domain) | | **PD=1** (use a predefined domain) | |
| | Signature Verification with public key | Signature Verification with private key | Signature Verification with public key | Signature Verification with private key |
| PDB pointer 1 | *q* | *q* | *Wx,y* | *s* |
| PDB pointer 2 | *r* | *r* | *f* (MSG_REP=0)<br>or *m* (MSG_REP=1) | *f* (MSG_REP=0)<br>or *m* (MSG_REP=1) |
| PDB pointer 3 | *g* (DSA)<br>or *Gx,y* (ECDSA) | *g* (DSA)<br>or *Gx,y* (ECDSA) | *c* | *c* |
| PDB pointer 4 | *w* (DSA)<br>or *Wx,y* (ECDSA) | *s* | *d* | *d* |
| PDB pointer 5 | *f* (MSG_REP=0)<br>or *m* (MSG_REP=1) | *f* (MSG_REP=0)<br>or *m* (MSG_REP=1) | Temp | See notes 2 and 3 |
| PDB pointer 6 | *c* | *c* | See notes 2 and 3 | |
| PDB pointer 7 | *d* | *d* | | |
| PDB pointer 8 | Temp | See notes 1 and 3 | | |

*Table continues on the next page...*

## Table 8-15. DSA and ECDSA Signature Verification protocol data block, continued (continued)

| | | | | |
|---|---|---|---|---|
| PDB pointer 9 | See notes 1 and 3 | See notes 2 and 3 | | |
| PDB pointer 10 | See notes 2 and 3 | | | |
| Note 1: If ECDSA, this is a pointer to *a,b*, else if MSG_REP=1, this is the message length, else the previous word is the last word of the PDB.<br>Note 2: If ECDSA and MSG_REP=1, this word is the 32-bit message length, else the previous word is the last word of the PDB.<br>Note 3: MSG_REP=1 means calculate the message representative from the message m. In this case the message length must be provided in one word of the PDB. MSG_REP=0 means don't calculate a message representation, use f as the message representative. In this case the word that would contain the message length is omitted from the PDB. | | | | |

This table shows the format of the SGF field.

## Table 8-16. DSA and ECDSA Signature Verification protocol data block - Format of the SGF Field

| **Formats for** | **Format for** | | bit number | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |
| **DSA or ECDSA with PD=0** | **Verify with public key** | **SGF bit for:** | q | r | g (DSA)<br>or<br>Gx,y (ECDSA) | w (DSA)<br>or<br>Wx,y (ECDSA) | f (MSG_REP=0)<br>or<br>m (MSG_REP=1) | c | d | Temp | a,b (ECDSA)<br>or<br>reserved (DSA) |
| | **Verify with private key** | | q | r | g (DSA)<br>or<br>Gx,y (ECDSA) | s | f (MSG_REP=0)<br>or<br>m (MSG_REP=1) | c | d | Temp | a,b (ECDSA)<br>or<br>reserved (DSA) |
| **ECDSA with PD=1** | **Verify with public key** | **SGF bit for:** | reserved | reserved | reserved | Wx,y (ECDSA) | f (MSG_REP=0)<br>or<br>m (MSG_REP=1) | c | d | Temp | reserved |
| | **Verify with private key** | | reserved | reserved | reserved | s | reserved | c | d | Temp | reserved |
| | | | *If the SGF bit for an argument is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer.* | | | | | | | | |

## 8.6 RSA Finalize Key Generation (RFKG)

SEC is able to complete RSA key generation given primes p and q and the public exponent e. It can be configured to compute, or skip computation of, the remaining elements of the public and private key.

The computations performed are:

- $n = p*q$
- $d = 1/e \bmod \mathrm{LCM}(p-1, q-1)$
- $d1 = d \bmod (p-1)$
- $d2 = d \bmod (q-1)$
- $c = 1/q \bmod p$

it will also

- check p and q to determine whether they are 'too close' (per FIPS 186-3). This will occur if they are long enough. It will not be effective if they are not of the same bit length (that is, high order bits of p and q are not the same).
- Compute #d if d is being computed
- Check that #d > #p.
- Check that #n and the computed #n are the same

RSA Finalize Key Generation PDB

**Table 8-17. RSA Finalize Key Generation PDB**

<table>
<tr><th colspan="4">Descriptor header (one or two words)</th></tr>
<tr><td>SGF<br>(9 bits)</td><td colspan="3">Reserved<br>(23 bits)</td></tr>
<tr><td colspan="3">Reserved<br>(23 bits)</td><td>#p<br>(9 bits)</td></tr>
<tr><td>Rsv<br>(6 bits)</td><td>#n<br>(10 bits)</td><td>Reserved<br>(6 bits)</td><td>#e<br>(10 bits)</td></tr>
<tr><td colspan="4">Reference to p</td></tr>
<tr><td colspan="4">Reference to q</td></tr>
<tr><td colspan="4">Reference to e</td></tr>
<tr><td colspan="4">Reference to n</td></tr>
<tr><td colspan="4">Reference to d</td></tr>
<tr><td colspan="4">Reference to #d</td></tr>
<tr><td colspan="4">Reference to d1 (not required if FUNCTION=10b)</td></tr>
<tr><td colspan="4">Reference to d2 (not required if FUNCTION=10b)</td></tr>
<tr><td colspan="4">Reference to c (not required if FUNCTION=10b)</td></tr>
</table>

The fields #d and #n contain right-aligned 10-bit values that indicate the size (plaintext size, in bytes) of the d and n inputs, respectively. Note that the size of d must be at least as large as the size of an encrypted n.

A reference is a pointer, either to the data or to a scatter-gather table. The pointer is one or two words long, depend upon the platform.

The references to d1, d2, and c may be omitted if those values are not to be generated.

This figure shows the format of the SGF field.

**Table 8-18. RSA Finalize Key Generation PDB - SGF field**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 |
|---|---|---|---|---|---|---|---|---|---|---|
| ref p | ref q | ref e | ref n | ref d | ref #d | ref d1 | ref d2 | ref c | Reserved | Reserved |
| If the SGF bit for a particular reference is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer. | | | | | | | | | | |

## 8.7 Implementation of the RSA encrypt operation

SEC implements an RSA encrypt operation that can be used for various purposes, including support for RSA-Based IKEv1 for IPsec and SSL-TLS. It is the "RSA public key primitive" and it is commonly used to encrypt a secret or to verify a signature.

When used for signature verification, it is actually "unscrambling" the signature so that its contents may be verified. The input must be passed "raw" to the RSA function.

SEC implements the RSA encrypt operation in the following form:

g = RSA-Encrypt(n, e, FORMAT, #f, f, fff)

The variables have the following definitions:

- n, e represent the public key
- Before the RSA math is performed, FORMAT specifies the format to be used for encoding f (none or PKCS #1 v1.5 encryption)
- f is the value to be RSA-encrypted (input value; will be output value if random data ('f out')is selected)
- #f is the size in bytes of f
- fff represents the type of encryption applied to f if it is output by SEC
- g is the RSA-encrypted value of (the possibly formatted version of) f

The RSA Encrypt function is implemented with the OPERATION command. See PROTOCOL OPERATION commands for details on selecting this operation. See Table 7-63 and Table 7-64 for details about the PROTINFO field in the OPERATION command.

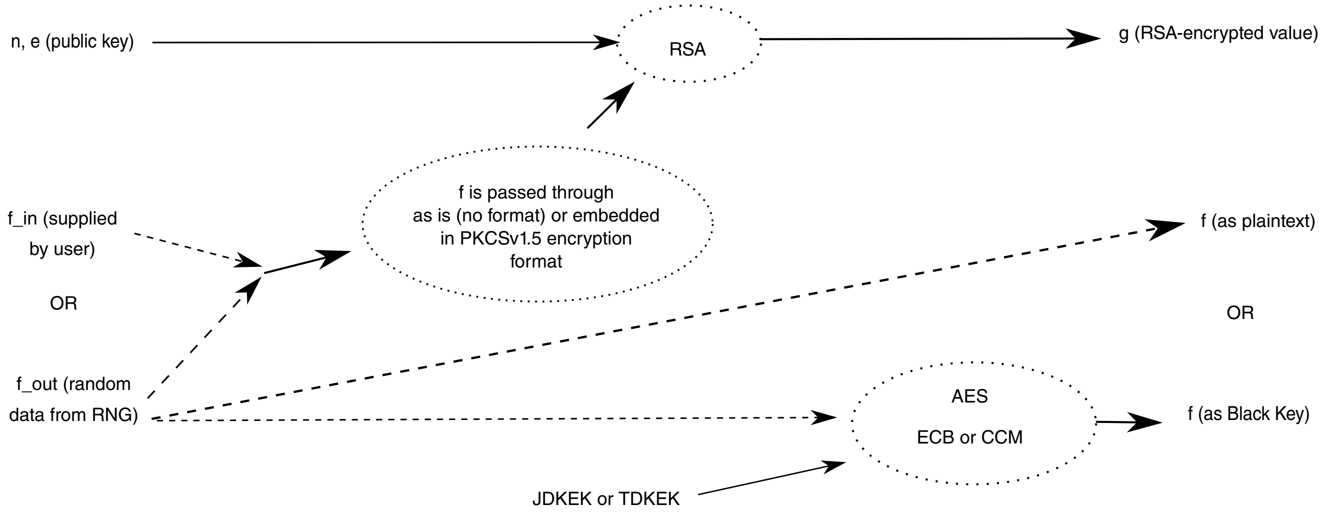


**Figure 8-1. RSA encrypt operation**

The user may either supply a plaintext value to be RSA encrypted (f_in) or may opt to have SEC generate #f bytes of random data from the RNG (f_out). The latter option allows f to be stored encrypted as a black key, It is then protected; encrypted or not, it can be used as an input to a PRF operation.

Once the value of f is known and possibly wrapped in PKCSv1.5 encoding, it is RSA-encrypted and the result stored as g.

The PDB for the RSA encrypt operation is shown below.

**Table 8-19. RSA Encrypt PDB**

<table>
<tr><td>SGF<br>(4 bits)</td><td>Rsv<br>(4 bits)</td><td>#e<br>(12 bits)</td><td>#n<br>(12 bits)</td></tr>
<tr><td colspan="4">Reference to f</td></tr>
<tr><td colspan="4">Reference to g</td></tr>
<tr><td colspan="4">Reference to n</td></tr>
<tr><td colspan="4">Reference to e</td></tr>
<tr><td colspan="3">Reserved<br>(20 bits)</td><td>#f<br>(12 bits)</td></tr>
</table>

All references are either 32-bit, 36-bit or 40-bit address pointers.

The fields #e, #n and #f contain right-aligned 12-bit values that indicate the size of the e, n and f inputs, respectively. The format of the SGF field is shown below.

**Table 8-20. RSA Encrypt PDB; SGF field**

| 31 | 30 | 29 | 28 |
|---|---|---|---|
| ref f | ref g | ref n | ref e |
| If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer. | | | |

## 8.8 Implementation of the RSA decrypt operation

SEC implements an RSA decrypt operation that can be used for various purposes, including support for RSA-Based IKEv1 for IPsec and SSL-TLS. This is the "RSA private key primitive" and it is commonly used either to decrypt a secret or to create a signature (sign a message).

When used for signing a message, it is actually "scrambling" the signature; the output must be allowed to pass "raw" from the RSA function.

SEC implements the RSA decrypt operation in the form:

f = RSA-Decrypt((private key), FORMAT, g, fff)

The variables have the following definitions:

- (private key) represents the private key, in one of three forms
- After the RSA math is performed on g, FORMAT specifies the format to be used for decoding f (none or PKCS #1 v1.5 encryption)..
- g is the input value
- fff represents the type of encryption applied to f when it is output by SEC
- f is the RSA-decrypted output value.

This operation leaves #f (size of the plaintext f, in bytes) in the MATH0 register. This may be important if using a FORMAT of PKCS #1 v1.5 encryption.

The RSA Decrypt function is implemented via the OPERATION Command. See PROTOCOL OPERATION commands for details on selecting this operation. See Table 7-65 and Table 7-65 for details concerning the PROTINFO field in the OPERATION Command.

As the private key is an input and is considered sensitive, it may be supplied in Black Key form. The components, individually, of the private key would then be decrypted using the appropriate key encryption key and cryptographic mode prior to use. Note that n is never encrypted. SEC allows the private key input to be provided in three different forms: #1 (n, d), #2 (p, q, d), and #3 (p, q, dp, dq, c).

The RSA Decrypt function is implemented via the OPERATION Command. See PROTOCOL OPERATION commands for details on selecting this operation. See Table 7-65 and Table 7-66 for details concerning the PROTINFO field in the OPERATION Command.

The operation of RSA decrypt when using form #1, in which the private key is input as (n, d), is illustrated below.

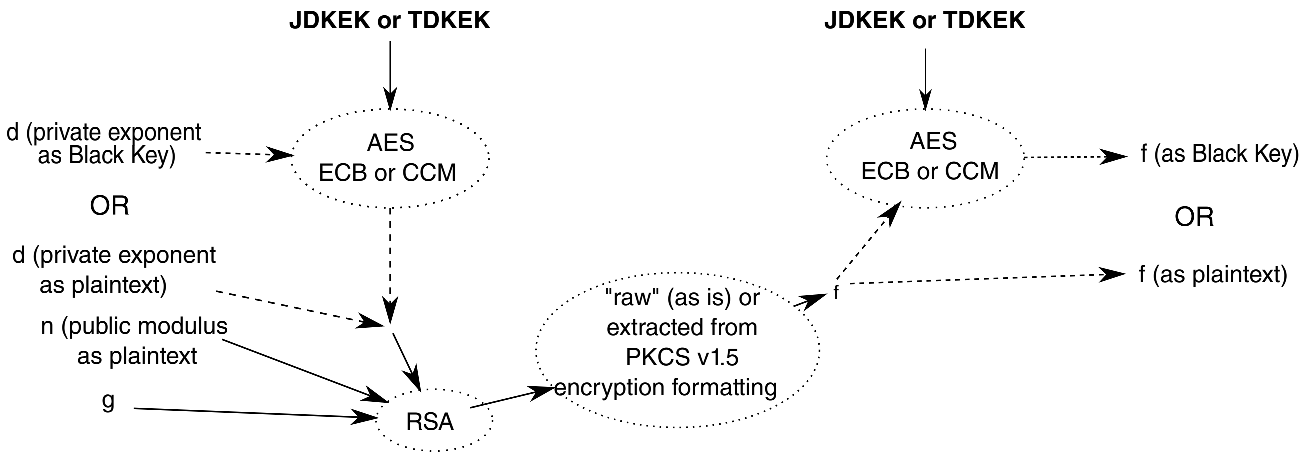

JDKEK or TDKEK
JDKEK or TDKEK
d (private exponent as Black Key)
AES ECB or CCM
OR
d (private exponent as plaintext)
n (public modulus as plaintext
g
RSA
"raw" (as is) or extracted from PKCS v1.5 encryption formatting
f
AES ECB or CCM
f (as Black Key)
OR
f (as plaintext)

**Figure 8-2. RSA decrypt operation - private key form #1**

The PDB for private key form #1 is shown below. All references are either 32-bit, 36-bit or 40-bit address pointers.

**Table 8-21. RSA decrypt PDB - private key form #1**

| SGF<br>(4 bits) | Rsv<br>(4 bits) | #d<br>(12 bits) | #n<br>(12 bits) |
|---|---|---|---|
| Reference to g | | | |
| Reference to f | | | |
| Reference to n | | | |
| Reference to d | | | |

The fields #d and #n contain right-aligned 12-bit values that indicate the size (plaintext size, in bytes) of the d and n inputs, respectively. This figure shows the format of the SGF field.

Table 8-22. RSA decrypt PDB - private key form #1; SGF field

| 31 | 30 | 29 | 28 |
|---|---|---|---|
| ref g | ref f | ref n | ref d |
| If the SGF bit for a particular reference is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer. | | | |

The RSA decrypt operation also accepts the private key in the form (p, q, d). This form (form #2) of the RSA decrypt operation is illustrated in this figure.

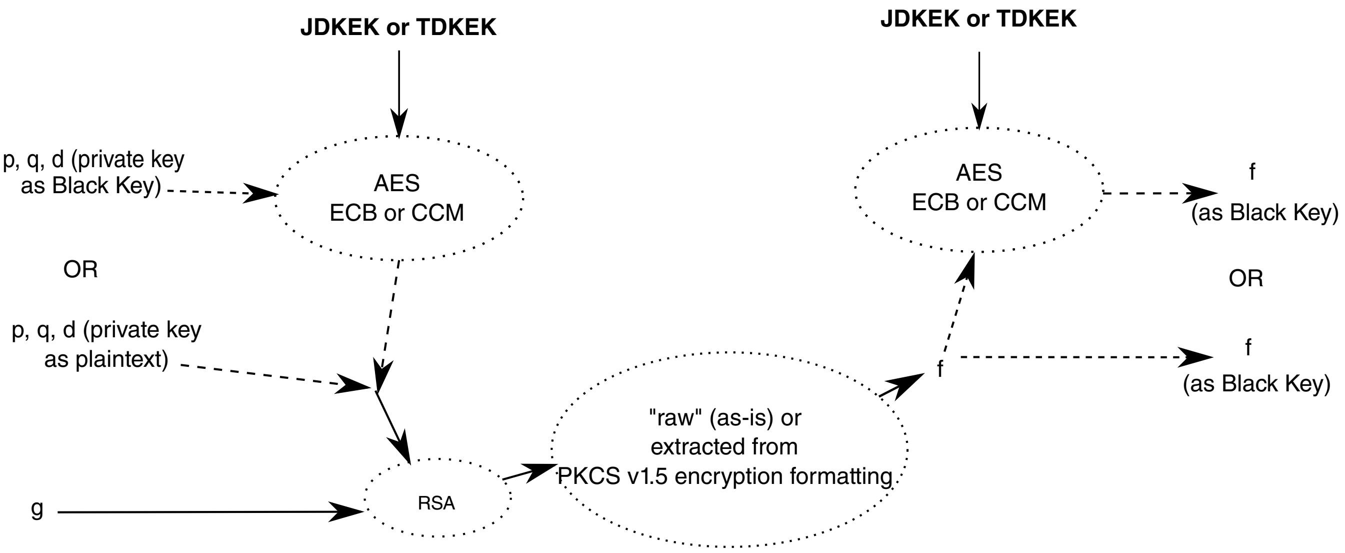

JDKEK or TDKEK
JDKEK or TDKEK
p, q, d (private key as Black Key)
AES ECB or CCM
AES ECB or CCM
f (as Black Key)
OR
OR
p, q, d (private key as plaintext)
f
f (as Black Key)
"raw" (as-is) or extracted from PKCS v1.5 encryption formatting
g
RSA

Figure 8-3. RSA decrypt operation - private key form #2

This figure shows the PDB for private key form #2. All references are either 32-bit, 36-bit or 40-bit address pointers.

Table 8-23. RSA decrypt PDB - private key form #2

| SGF<br>(7 bits) | Rsv<br>(1 bit) | #d<br>(12 bits) | #n<br>(12 bits) |
|---|---|---|---|
| Reference to g | | | |
| Reference to f | | | |
| Reference to d | | | |
| Reference to p | | | |
| Reference to q | | | |
| Reference to tmp1 | | | |
| Reference to tmp2 | | | |
| Reserved<br>(8 bits) | | #q<br>(12 bits) | #p<br>(12 bits) |

The fields #d, #n, #q and #p contain right-aligned 12-bit values that indicate the size (plaintext sizes, in bytes) of d, n, q and p, respectively. Note that even though there is no n input, #n is still needed, as it is not just #p + #q. tmp1 needs to be as long as p (either #p, or, if p is encrypted, as big as the encrypted value of p). tmp2 needs to be as long as q (either #q, or. if q is encrypted, as big as the encrypted value of q). This figure shows the format of the SGF field.

**Table 8-24. RSA decrypt PDB - private key form #2; SGF field**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
|---|---|---|---|---|---|---|
| ref g | ref f | ref d | ref p | ref q | ref tmp1 | ref tmp2 |
| If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer. | | | | | | |

The RSA decrypt operation also accepts the private key in form #3, (p, q, dp, dq, c). dp, dq, and c are

- $dp = d \bmod p\text{-}1$
- $dq = d \bmod q\text{-}1$
- $c = q^{-1} \bmod p$

The operation this form of RSA decrypt is illustrated in this figure.

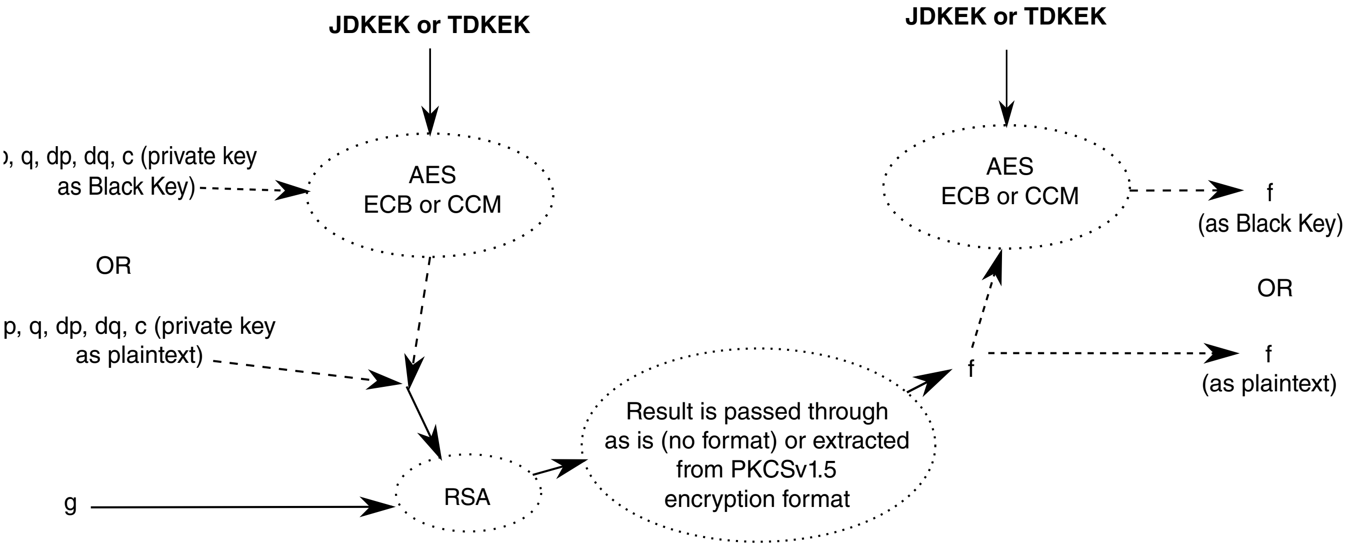

JDKEK or TDKEK
JDKEK or TDKEK
p, q, dp, dq, c (private key as Black Key)
AES
ECB or CCM
AES
ECB or CCM
f
(as Black Key)
OR
OR
p, q, dp, dq, c (private key as plaintext)
f
f
(as plaintext)
Result is passed through as is (no format) or extracted from PKCSv1.5 encryption format
g
RSA

**Figure 8-4. RSA Decrypt Operation - private key form #3**

This figure shows the PDB for private key form #3. All references are either 32-bit, 36-bit or 40-bit address pointers.

**Table 8-25. RSA decrypt PDB - private key form #3**

| | | |
|---|---|---|
| SGF | Reserved | #n |

*Table continues on the next page...*

### Table 8-25. RSA decrypt PDB - private key form #3 (continued)

| (9 bits) | (11 bits) | (12 bits) |
|---|---|---|
| Reference to g | | |
| Reference to f | | |
| Reference to c | | |
| Reference to p | | |
| Reference to q | | |
| Reference to $d_p$ | | |
| Reference to $d_q$ | | |
| Reference to tmp1 | | |
| Reference to tmp2 | | |
| Reserved<br>(8 bits) | #q<br>(12 bits) | #p<br>(12 bits) |

The fields #n, #q and #p contain right-aligned, 12-bit values that indicate the size (plaintext sizes, in bytes) of n, q and p, respectively. Note that even though there is no n input, #n is still needed, as it is not just #p + #q. Note that #dp and #c are assumed to be #p, and #dq is assumed to be #q. tmp1 needs to be as long as p (either #p, or, if p is encrypted, as big as the encrypted value of p). tmp2 needs to be as long as q (either #q, or, if q is encrypted, as big as the encrypted value of q).

### Table 8-26. RSA Decrypt PDB - private key form #3; SGF Field

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |
|---|---|---|---|---|---|---|---|---|
| ref g | ref f | ref c | ref p | ref q | ref dp | ref dq | ref tmp1 | ref tmp2 |
| **NOTE:** If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct address pointer. | | | | | | | | |

# Chapter 9
# Protocol acceleration

SEC is designed to accelerate the cryptographic operations associated with various network protocols. These cryptographic operations can be implemented using the descriptor commands described in sections KEY commands through SEQ OUT PTR command, but SEC also implements specialized descriptor commands for particular networking protocols. Each such command performs a sequence of operations that are equivalent to a series of the more general descriptor commands; for example, all the protocols in this section manage the input data and output FIFOs directly -- a SEQ FIFO LOAD command is required in a descriptor only if there is data in the input frame that is not to be handled by the protocol. These protocols often require that state information (for example, sequence numbers) be maintained per security association.

The specialized protocol commands implemented by SEC use data structures called protocol data blocks (PDBs) embedded within the descriptor to specify protocol options and hold state information. Typically these protocol commands and their associated PDBs are contained in shared descriptors, so that the same protocol options and state information can be shared among all the job descriptors that identify the PDUs within a particular security association. The PDB is embedded within the shared descriptor immediately following the header, and the START INDEX field in the header is used to skip over the PDB to continue executing the commands within the shared descriptor.

If the protocol requires that state information be updated, SEC writes the updated information back to the PDB in the shared descriptor located in system memory.

Sharing is described in Shared descriptors. Sharing significantly impacts Protocol operation in particular, because SEC protocols tend to use a shared descriptor PDB to share state across many jobs within a flow. One example is a sequence or packet number -- it is important that only one packet be encapsulated with a given sequence number. Therefore sharing type as described in Table 7-1 is to be carefully considered when crafting a descriptor.

SEC is most efficent when using serial sharing -- there is extra time associated with moving shared descriptor material from one DECO to another. On the other hand, an idle DECO resulting from overuse of serial sharing is a greater cost. Therefore the overall set of applications should be considered.

SEC protocols maintain a lock called OK to Share in order to allow for wait sharing. For example, consider IPsec encapsulation using CBC mode. CBC requires every packet include an Initialization Vector (an IV). For IPsec, either the IV can be the final block of ciphertext from the previous packet (a Chained IV), or it can be a random value (a Random IV). The IPsec protocol state machine will block WAIT sharing of a shared descriptor until a Chained IV has been prepared and OK to Share is signalled. If instead a Random IV is used, OK to Share can be set as soon as the IPsec protocol state machine has updated Sequence Number in the PDB. It is probably not useful to use WAIT sharing with a Chained IV; two jobs from a single flow can only be present in multiple DECOs for a very limited period of time.

Never and Always Sharing should be used with extreme care. Selecting share type of Always will cause a shared descriptor to be shared between DECOs without consideration of state. In the IPsec encapsulation example, Always sharing can result in packets with duplicate sequence numbers. Duplicate sequence numbers can also result from using Never sharing, as a DECO will get a fresh copy of a Shared Descriptor from system memory, without any consideration for any pending writes to update the Shared Descriptor PDB from another DECO.

SEC includes built-in descriptor programming shortcuts for the following functions:

- IPsec ESP Encapsulation and Decapsulation
- SSL 3.0 Record Encapsulation and Decapsulation
- TLS 1.0, 1.1, and 1.2 Record Encapsulation and Decapsulation
- DTLS 1.0 and 1.2 Record Encapsulation and Decapsulation
- SRTP Packet Encapsulation and Decapsulation
- IEEE 802.1AEbw-2013 MACsec Encapsulation and Decapsulation
- IEEE 802.11-2012 WPA2 Encapsulation and Decapsulation for WiFi
- IEEE 802.16 WiMAX Encapsulation and Decapsulation
- 3G Double CRC
- 3G RLC Encryption and Decryption
- LTE PDCP Encapsulation and Decapsulation
- Cryptographic Blob Encapsulation and Decapsulation

Each detailed description of the function includes color-coded diagrams. Figure 9-1 shows the color coding key. Note that in the diagrams, processing order is reflected top-to-bottom, and PDU content is reflected left-to-right.

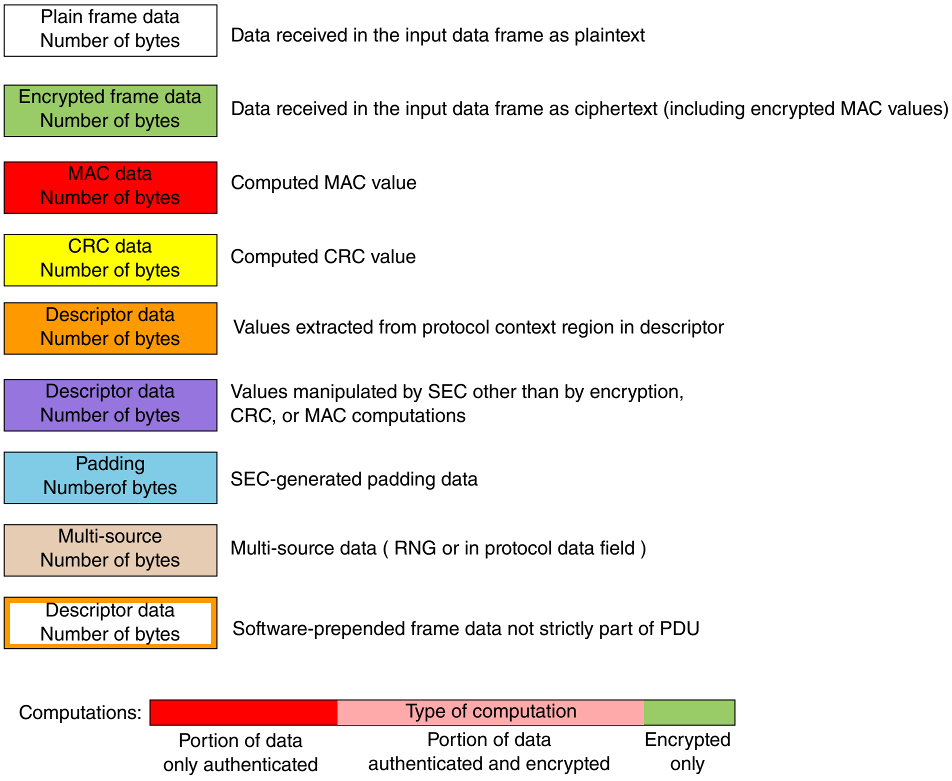

Plain frame data
Number of bytes
Data received in the input data frame as plaintext
Encrypted frame data
Number of bytes
Data received in the input data frame as ciphertext (including encrypted MAC values)
MAC data
Number of bytes
Computed MAC value
CRC data
Number of bytes
Computed CRC value
Descriptor data
Number of bytes
Values extracted from protocol context region in descriptor
Descriptor data
Number of bytes
Values manipulated by SEC other than by encryption, CRC, or MAC computations
Padding
Numberof bytes
SEC-generated padding data
Multi-source
Number of bytes
Multi-source data ( RNG or in protocol data field )
Descriptor data
Number of bytes
Software-prepended frame data not strictly part of PDU
Computations:
Type of computation
Portion of data only authenticated
Portion of data authenticated and encrypted
Encrypted only

**Figure 9-1. Protocol diagram color-coding key**

## 9.1 IPsec ESP encapsulation and decapsulation overview

SEC's built-in IPsec protocol supports data encapsulation, encryption, and data integrity checking for the following cipher suites:

- DES-CBC or 3DES-CBC with selected HMAC algorithms, AES-XCBC-MAC, or AES-CMAC
- AES-CBC with selected HMAC algorithms, AES-XCBC-MAC, or AES-CMAC
- AES-CTR (AES-Counter) with selected HMAC algorithms, AES-XCBC-MAC, or AES-CMAC
- AES-CCM
- AES-GCM

The PROTINFO field codes enumerated in Table 7-54 are used to define the specific encryption and data integrity algorithms to be used by the protocols.

SEC supports different IP versions as follows:

- For IPv4, SEC supports zero or one 32-bit option field.

- For IPv6, SEC supports up to 96 bytes of extension headers.
- For all modes, SEC treats the option field and extension headers as part of the IP header.

SEC supports IPsec with null encryption. For the most part IPsec with null encryption processes the packet in a very similar fashion to how IPsec processing occurs with encryption. Null encryption is not supported when AES-CCM is selected. When AES-GCM is selected, null encryption uses the AES-GMAC algorithm. In this case, the datagram is processed identically to packet processing for AES-GCM (including the use of an IV), except that the payload is not encrypted. For IPsec null encryption when using any HMAC algorithm, no IV is included in the encapsulated frame. Otherwise, the encapsulated packet contains all the other fields, including padding and pad length.

SEC also supports IPsec with null authentication (with selected cipher algorithms). For null authentication, SEC does not produce an ICV; an ICV is not written to the encapsulation output frame, and no ICV check is performed during decapsulation. Null Authentication may be combined with Null encryption, in which case SEC produces an output frame that consists of an IP header followed by an ESP header, payload, and then an ESP trailer with minimum padding.

The shared descriptor protocol data block (PDB) includes a field that indicates the byte length of the IP header with options/extensions.

The following table summarizes the IPsec protocol descriptors.

**Table 9-1. IPsec protocol descriptors**

| Encapsulation | | Decapsulation |
|---|---|---|
| Header | | Header |
| Protocol data block includes next header, SPI, sequence number, IV (if not from RNG) | | Protocol data block includes anti-replay information |
| Class 2 key data block | | Class 2 key data block |
| Class 1 key data block | | Class 1 key data block |
| Protocol = IPsec encrypt | | Protocol = IPsec decrypt |

## NOTE

Any bulk-data protocol using a cipher suite that includes any HMAC uses MDHA and for performance requires the use of a split key. Therefore for proper operation when using IPsec with HMAC, the KDEST field in the Class 2 KEY command must be set to MDHA Split Key. For first invocation, the Derived Key Protocol may be used to create both the split key form of the HMAC key as well as the actual key command loading the split key.

### 9.1.1 IPsec ESP encapsulation and decapsulation mode support

SEC supports two modes: transport and tunnel.

- During tunnel mode encapsulation, SEC treats the entire input frame as payload except for the optional Outer IP header. During tunnel mode decapsulation, the original IP header (with options and extensions) is uncovered.
- During tunnel mode encapsulation, SEC can prepend a new IP header (with options and extensions). For decapsulation SEC must be given a pointer to the start of the encapsulated data, skipping over the Outer IP header.
- For transport mode, the pre-encapsulation IP header and post-encapsulation IP header are virtually identical, except for the 2-byte length field and the header checksum.
- For both tunnel and transport modes, SEC recomputes the length field in the header, can optionally recompute the header checksum.

The Protocol Data Block, or PDB, is used to store relevant parameters within a descriptor. Parts of the PDBs are specific to particular cipher suites, but most is common to either encapsulation or decapsulation. The common PDB for encapsulation is described in PDB format for IPsec ESP Transport (and Legacy Tunnel) encapsulation, and for decapsulation in Common PDB format descriptions for IPsec ESP Transport (and Legacy Tunnel) decapsulation.

Prior to SEC Era 8, what is now called ESP Transport (and legacy tunnel) covered both tunnel and transport modes. More recent developments to ESP Tunnel support, such as the addition of direct support for NAT, have necessitated the addition of the new ESP Tunnel encapsulation and decapsulation threads. For the forseeable future, Tunnel mode IPsec encapsulation and decapsulation can be achieved by either the ESP Transport (and legacy tunnel) or the ESP Tunnel processing threads.

The PDBs for ESP Tunnel are similar to those for ESP Transport (and Legacy Tunnel) -- in particular that both share that there is a segment of the PDB specific to the chosen cipher suite. The common PDB for ESP Tunnel encapsulation is described in PDB format for IPsec ESP Tunnel encapsulation and for decapsulation in Common PDB format descriptions for IPsec ESP Tunnel decapsulation.

### 9.1.2 IPsec ESP error codes

This table lists the conditions under which IPsec encapsulation or decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine. Authentication failure in decapsulation can also produce an ICV check error.

### Table 9-2. IPsec ESP encapsulation and decapsulation error conditions for both Tunnel and Transport operation

| Condition | Error status | Applies to encapsulation/ decapsulation |
|---|---|---|
| Input frame too long<br>• For IPsec ESP Transport (and legacy tunnel), the input frame is longer than $2^{16}$ bytes<br>• For IPsec ESP Tunnel, the input frame is longer than $2^{20}$ bytes<br>Note this length restriction applies to the input frame received at the start of the protocol itself. Any bytes that are part of the input frame that are dispositioned by commands prior to either IPsec ESP protocol are not governed by the size restriction. | Protocol size error | Both |
| Reserved bit set to 1 in the PDB options byte | Protocol PDB error | Both |
| ARS in PDB set to reserved setting | Protocol PDB error | Decapsulation only |
| Tun/Trsp = 1 and NH_OFFSET != 0 | Protocol PDB error | Both |
| Inc IPHdr = 1 and PDB IP Hdr Length == 0 | Protocol PDB error | Both |
| Inc IPHdr = 1 and PDB IP Hdr Length is not a multiple of 4 | Protocol PDB error | Both |
| Tun/Trsp = 0, and NH_OFFSET points to a byte beyond PDB IP Hdr Length | Protocol PDB error | Both |
| OPERATION Command PROT ID selects IPsec Encap/Decap, and PROTINFO is not a valid protocol | Protocol Command Error | Both |
| ESN option = 0, and PDB SEQNUM is FFFFFFFF | Protocol Sequence Number Overflow | Encapsulation only |
| ESN option = 1, both ESN and SEQNUM are FFFFFFFF | Protocol Sequence Number Overflow | Encapsulation only |
| ESN option = 0, and SEQNUM overflows | Protocol Sequence Number Overflow | Decapsulation only |
| ESN option = 1, and [ESN and SEQNUM] overflows | Protocol Sequence Number Overflow | Decapsulation only |
| Anti-Replay detects a LATE packet | Protocol LATE error | Decapsulation only |
| Anti-Replay detects a REPLAY packet | Protocol REPLAY error | Decapsulation only |
| Output option 0 selected and AOFL option bit == 1 | Protocol PDB error | Decapsulation only |
| UDP-encapsulated-ESP CE Drop<br>See Manipulation of the Inner IP Header during ESP Tunnel decapsulation | CE DROP Error | Decapsulation only |

## 9.1.3 Programming for IPsec

IPsec in SEC is designed to provide stateful encapsulation and decapsulation of IPsec ESP packets. The SEC representation of the ESP security association is the PDB -- the Protocol Data Block. SPI and Sequence Number are maintained in the PDB, as are other parameters and selectible options. The format of the PDB varies depending upon cipher suite and options selected, and are shown in sections following.

In some circumstances, it may be desirable to override some contents of the PDB on a frame-by-frame basis. Some capability is provided for that by programming the DECO Protocol Override register. Use and capability of that functionality is also shown in sections following.

### 9.1.3.1 PDB format for IPsec ESP Transport (and Legacy Tunnel) encapsulation

IPsec ESP Transport (and Legacy Tunnel) encapsulation uses a mostly-common Protocol Data Block (PDB) format to maintain certain state and security association information. To complete encapsulation, SEC requires access to SPI, Sequence Number, Extended Sequence Number (if used), plus some cipher-suite-specific material, such as the IV used by AES-CBC and DES-CBC.

The PDB can, optionally, also provide for a common outer IP header to be written to the output frame prior to the tunnel-mode encapsulated header. The header material need not consist only of the outer IP header, but if extra material is included prior to the IP header (such as an ethernet header), then the several options that can manipulate the outer header (such as DSC and Cksm) will not work right.

This PDB diagram shows the common form, and the common definitions of the Options and HMO bits follow afterwards. Details for the cipher-suite-specific portion are found:

- For AES-CBC and DES-CBC specific IV format, refer to IPsec ESP encapsulation CBC-specific PDB segment format descriptions
- For AES-CTR specific Counter and IV format, refer to IPsec ESP encapsulation AES-CTR-specific PDB segment format descriptions
- For AES-CCM specific data format, refer to IPsec ESP encapsulation AES-CCM-specific PDB segment format descriptions
- For AES-GCM specific Salt and IV format, refer to IPsec ESP encapsulation AES-GCM-specific PDB segment format descriptions

All fields shown here should be programmed as appropriate per the negotiated tunnel parameters. NH Offset is used only for transport mode. For more detail on fields other than HMO and Options, see Process for IPsec ESP Transport (and Legacy Tunnel) encapsulation. HMO and Options are described below the PDB diagram.

## Table 9-3. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB

| | Descriptor Header (1 or 2 words) | | | | | |
|---|---|---|---|---|---|---|
| PDB Word 0 | HMO<br>(4 bits) | Reserved<br>(4 bits) | Next Header<br>(8 bits) | NH Offset<br>(8 bits) | Options<br>(8 bits) | |
| PDB Word 1 | Optional Extended Sequence Number (ESN) | | | | | DECO writes back to PDB as needed |
| PDB Word 2 | Sequence Number | | | | | |
| PDB Word 3 | Cipher-suite-specific portion of PDB | | | | | for CBC-mode see CBC IV Format<br>for CTR-mode see CTR IV Format<br>for CCM-mode see CCM IV Format<br>for GCM-mode see GCM IV Format |
| PDB Word 4 | | | | | | |
| PDB Word 5 | | | | | | |
| PDB Word 6 | | | | | | |
| PDB Word 7 | SPI | | | | | |
| PDB Word 8 | Reserved | | | Opt IP Header Length | | |
| PDB Word 9 | Optional IP Header (bytes 0-3) | | | | | |
| PDB Word 10 | Optional IP Header (bytes 4-7) | | | | | ID field is incremented and written back as needed |
| PDB Word 11 | Optional IP Header (bytes 8-11) | | | | | |
| PDB Word 12 | Optional IP Header (bytes 12-15) | | | | | |
| PDB Word 13 + | Optional IP Header (bytes 16+) | | | | | |

## Table 9-4. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, format of the options byte

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Cksm | DSC | IVsrc | ESN | IPHdr Src | Inc IPHdr | IPvsn | Tun/ Trsp |

## Table 9-5. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| 7<br>Cksm | Enable Checksum Update<br>0 - Do not perform any checksum computations<br>1 - For any changes performed to the IP header, perform computations to update the header checksum |
| 6<br>DSC | DiffServ Copy<br>0 - Do not copy<br>1 - Copy the IPv4 TOS or IPv6 Traffic Class byte from the inner IP header to the IP header copied from the PDB. |

*Table continues on the next page...*

### Table 9-5. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, description of the options byte (continued)

| Field | Description |
|---|---|
| 5<br>IVsrc | IV Source<br>0 - Chained IV Stored in PDB<br>1 - Random IV fetched from RNG prior to starting encapsulation |
| 4<br>ESN | Extended Sequence Number<br>0 - ESN not included in ICV computation. Optional ESN field of PDB is reserved in this case.<br>1 - ESN is copied from PDB and used for ICV computation but is not written to the output frame. ESN is incremented as if part of the sequence number and written back to the PDB as required. |
| 3<br>IPHdrSrc | IP Header source<br>0 - IP header in input frame<br>1 - IP header in PDB |
| 2<br>Inc IPHdr | Include Optional IP Header<br>0 - Do not prepend IP header<br>1 - Prepend IP Header to output frame |
| 1<br>IPvsn | This field indicates which version of IP is being used.<br>0 - IPv4<br>1 - IPv6 |
| 0<br>Tun/Trsp | Selects Tunnel or Transport Mode with respect to handling of the N (Next Header) byte<br>0 - Transport mode<br>1 - Tunnel mode |

This figure shows the format of the HMO field.

### Table 9-6. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, format of the HMO field

| 31 | 30 | 29 | 28 |
|---|---|---|---|
| Reserved | DFC | DTTL | SNR |

### Table 9-7. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, description of the HMO field

| Field | Description |
|---|---|
| 31 | Reserved |
| 30<br>DFC | Copy DF bit<br>0 - Do not copy DF bit<br>1 - If an IPv4 tunnel mode outer IP header is coming from the PDB, copy the DF bit from the inner IP header to the outer IP header. If not in tunnel mode or the outer IP header from the PDB is not included, setting DFC = 1 causes a Protocol PDB error. |
| 29 | Decrement TTL (Hop Limit) |

*Table continues on the next page...*

Table 9-7. IPsec ESP Transport (and Legacy Tunnel) encapsulation PDB, description of the HMO field (continued)

| Field | Description |
|---|---|
| DTTL | 0 - Do not decrement<br>1 - Based on the value of the Options Byte IPvsn field: if IPv4, decrement the inner IP header TTL field (byte 8) and update the IPv4 header checksum; if IPv6 decrement the inner IP header Hop Limit field (byte 7). If TTL is decremented below 0, an error is generated. |
| 28<br>SNR | Sequence Number Rollover enable<br>0 - Sequence Number (as extended by ESN) does not rollover; an error is generated if a rollover is attempted<br>1 - Sequence Number (as extended by ESN) permitted to roll over |

### 9.1.3.2 Common PDB format descriptions for IPsec ESP Transport (and Legacy Tunnel) decapsulation

IPsec ESP Transport (and Legacy Tunnel) decapsulation uses a mostly-common Protocol Data Block (PDB) format to maintain certain state and security association information. To complete decapsulation, SEC requires access to SPI, Sequence Number, Extended Sequence Number (if used). SPI and Sequence Number can be used as extracted from the input frame, but the Extended Sequence Number is not included in the encapsulated datagram. PDB words 1 and 2 are reserved for some cipher-suite-specific material, such as the Salt used by AES-GCM. These words are reserved for CBC-based cipher suites because no other information is required.

This PDB diagram shows the common form, and the common definitions of the Options and HMO bits follows afterwards. For the cipher-suite-specific portion, additional details can be found as follows:

- For AES-CBC and DES-CBC specific IV format, refer to IPsec ESP decapsulation CBC-specific PDB segment format descriptions
- For AES-CTR specific Counter and IV format, refer to IPsec ESP decapsulation AES-CTR-specific PDB segment format descriptions
- For AES-CCM specific data format, refer to IPsec ESP decapsulation AES-CCM-specific PDB segment format descriptions
- For AES-GCM specific Salt and IV format, refer to IPsec ESP decapsulation AES-GCM-specific PDB segment format descriptions

Fields shown here should be programmed as appropriate per the negotiated tunnel parameters. NH Offset is used only for transport mode. The Anti-replay scorecard should be initialized with zeros. For more detail on fields other than HMO and Options, see IPsec ESP Transport (and Legacy Tunnel) decapsulation procedure overview. HMO and Options are described below the PDB diagram.

### Table 9-8. IPsec ESP Transport (and Legacy Tunnel) decapsulation PDB

| | Descriptor Header (1 or 2 words) | | | | |
|---|---|---|---|---|---|
| PDB Word 0 | HMO<br>(4 bits) | IP Header Length<br>(12 bits) | NH Offset<br>(8 bits) | Options<br>(8 bits) | |
| PDB Word 1 | Cipher-suite-specific portion of PDB | | | | unused for CBC-mode<br>for CTR-mode see CTR IV Format<br>for CCM-mode see CCM IV Format<br>for GCM-mode see GCM IV Format |
| PDB Word 2 | | | | | |
| PDB Word 3 | Optional Extended Sequence Number (ESN) | | | | DECO writes back to PDB as needed |
| PDB Word 4 | Sequence Number | | | | |
| PDB word 5 | anti-replay scorecard 1<br>[present if ARS not 00b] | | | | |
| PDB word 6 | anti-replay scorecard 2<br>[present if ARS either 10b or 11b] | | | | |
| PDB word 7 | anti-replay scorecard 3<br>[present if ARS= 10b] | | | | |
| PDB word 8 | anti-replay scorecard 4<br>[present if ARS= 10b] | | | | |

### Table 9-9. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, format of the options byte

| 7-6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| ARS | CKSM | ESN | OUT_FMT | AOFL | IPVSN | TUN/TRSP |

### Table 9-10. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| 7-6<br>ARS | Anti-replay window size.<br>00 - No anti-replay window<br>01 - 32-entry anti-replay window<br>10 - 128-entry anti-replay window<br>11 - 64-entry anti-replay window |
| 5<br>CKSM | Checksum Enable<br>0 - Checksum calculation/verification disabled<br>1 - Checksum calculation/verification enabled |
| 4<br>ESN | Include Extended Sequence Number:<br>0 - ESN not used. Optional ESN field of PDB is reserved in this case. |

*Table continues on the next page...*

## Table 9-10. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, description of the options byte (continued)

| Field | Description |
|---|---|
| | 1 - ESN is used for ICV and anti-replay computations. The ESN is not found in the input frame |
| 3<br>OUT_FMT | Output Frame format:<br>0 - All Input Frame fields copied to Output Frame<br>1 - Output Frame is just the decapsulated PDU |
| 2<br>AOFL | Adjust Output Frame Length<br>0 - Don't adjust output frame length -- output frame length reflects output frame actually written to memory, including the padding, Pad Length, and Next Header fields.<br>1 - Adjust output frame length -- subtract the length of the padding, the Pad Length, and the Next Header byte from the output frame length reported to the frame consumer.<br>If outFMT==0, this bit is reserved and must be zero. |
| 1<br>IPVSN | This field indicates which version of IP is being used.<br>0 - IPv4<br>1 - IPv6 |
| 0<br>TUN/TRSP | Selects Tunnel or Transport Mode with respect to handling the N (Next Header) byte<br>0 - Transport mode<br>1 - Tunnel mode |

## Table 9-11. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, format of the HMO field

| 31 | 30 | 29 | 28 |
|---|---|---|---|
| ODF | DFV | DTTL | DSC |

## Table 9-12. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, description of the HMO field

| Field | Description |
|---|---|
| 31<br>ODF | ODF -- Override DF bit in IPv4 header of decapsulated output frame<br>0 - DF bit the IPv4 header in the output frame is identical to that in the input frame (encapsulated header in the input frame if in tunnel mode)<br>1 - DF bit in the IPv4 header in the output frame is replaced with the DFV value as shown below.<br>If IPv6 is selected, then this bit is reserved and must be zero. |
| 30<br>DFV | DFV -- DF bit Value<br>If ODF (see above) is set, this bit replaces whatever value would have otherwise been placed in the IPv4 Header DF bit field.<br>If ODF is not set, then this bit is reserved and must be zero. |
| 29<br>DTTL | Decrement TTL (Hop Limit)<br>0 - Do not decrement |

*Table continues on the next page...*

Table 9-12. IPsec ESP Transport (and Legacy Tunnel) Common decapsulation PDB, description of the HMO field (continued)

| Field | Description |
|---|---|
| | 1 - Based on the value of the Options Byte IPvsn field if IPv4, decrement the inner IP header TTL field (byte 8) and update the IPv4 header checksum; if IPv6 decrement the inner IP header Hop Limit field (byte 7). If TTL is decremented below 0, an error is generated. |
| 28<br>DSC | DiffServ Copy<br>0 - Do not copy<br>1 - Copy the IPv4 TOS or IPv6 Traffic Class byte from the outer IP header to the inner IP header. |

### 9.1.3.3 Overriding ESP Transport (and legacy Tunnel) PDB content with the DECO Protocol Override Register

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For IPsec ESP Transport (and Legacy Tunnel) encapsulation and decapsulation, several fields are maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired values in the job descriptor. For more information, see Job Ring interface. When using the Queue Manager Interface, QI builds the job descriptor with a LOAD IMMEDIATE to the DPOVRD register with the value of the STATUS/CMD field in the FD. For more information, see Queue Manager Interface (QI).

When DPOVRD is selected for use, SEC then uses the following values as specified in DPVORD instead of as specified in the PDB:

- IP Header Length
- NH OFFSET,
- Next Header (encapsulation only)
- ECN

Note that the values in DPOVRD in no way affect the values stored in the descriptor's PDB. DPOVRD works the same way regardless of the format of the IPsec encapsulation and decapsulation PDB, although the PDB's format varies depending on the chosen cipher suite. For details on how each of these fields are used, please refer to the appropriate section -- either PDB format for IPsec ESP Transport (and Legacy Tunnel) encapsulation or Common PDB format descriptions for IPsec ESP Transport (and Legacy Tunnel) decapsulation.

### Table 9-13. IPsec ESP Transport (and legacy Tunnel) format of the DPOVRD register for encapsulation and decapsulation

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OVRD | Reserved | | | ECN | | | | IP Header Length | | | | | | | |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| NH OFFSET | | | | | | | | Next Header [Encapsulation only]<br>Reserved for decapsulation | | | | | | | |

### Table 9-14. IPsec ESP Transport (and Legacy Tunnel) description of the DPOVRD register for encapsulation and decapsulation

| Field | Description |
|---|---|
| 31<br>OVRD | Selects whether DPOVRD overrides the values specified in the PDB.<br>0: Use the PDB as provided<br>1: Override values in PDB by using values in DPOVRD |
| 30-28 | Reserved |
| 27-24<br>ECN | If OVRD = 1 and the MSB of the ECN field (bit 27) = 1, the two LSBs of the ECN field (bits 25 and 24) replace the ECN bits in the IP header written to the output frame as part of encapsulation or decapsulation. |
| 23-16<br>IP Header Length | The length, in bytes, of the portion of the IP header that is not encrypted.<br>If IP Header Length = 0, the entire IP header is encrypted. In transport mode this indicates to SEC that this particular packet has an IP header that is not of typical length, so that SEC does not authenticate or encrypt any of the atypical IP headers found in the input frame. |
| 15-8<br>NH OFFSET | The location of the next header field within the IP header of the transport mode packet.<br>This location is indicated by the number of bytes from the beginning of the IP header.<br>For encapsulation, the value at this location is replaced with the contents of the Next Header byte (bits 7-0), and the value at the NH OFFSET location is used as the N byte in constructing the IPsec encapsulation trailer.<br>For decapsulation, the reverse occurs; the N byte value decrypted and extracted from the ESP trailer replaces the Next Header byte at this offset into the IP header. |
| 7-0<br>Encapsulation: Next Header<br>Decapsulation: Reserved | For encapsulation, this value is placed in the transport mode header at location NH OFFSET, replacing the value that was in the Next Hdr field of that particular header or extension header.<br>For decapsulation, this field is reserved. |

## 9.1.3.4 PDB format for IPsec ESP Tunnel encapsulation

The PDB for IPsec ESP Tunnel encapsulation descriptors is very similar to the PDB used by IPsec ESP Transport (and Legacy Tunnel) encapsulation. However, the use of the HMO and Options fields has changed significantly.

The Transport encapsulation PDB field NH Ofst is used only for transport-mode applications; the Tunnel encapsulation PDB replaces that field with AOIPHO - Actual Outer IP Header Offset. Tunnel encapsulation can permit the so-called Outer IP Header material to include extra material, such as an ethernet header, which will be ignored for the purpsoes of modifying the Outer IP header based on inner IP header values.

This PDB diagram shows the common form, and the common definitions of the Options and HMO bits follow afterwards. Details for the cipher-suite-specific portion are found:

- For AES-CBC and DES-CBC specific IV format, refer to IPsec ESP encapsulation CBC-specific PDB segment format descriptions
- For AES-CTR specific Counter and IV format, refer to IPsec ESP encapsulation AES-CTR-specific PDB segment format descriptions
- For AES-CCM specific data format, refer to IPsec ESP encapsulation AES-CCM-specific PDB segment format descriptions
- For AES-GCM specific Salt and IV format, refer to IPsec ESP encapsulation AES-GCM-specific PDB segment format descriptions

All fields shown here should be programmed as appropriate per the negotiated tunnel parameters. For more detail on fields other than HMO and Options, see IPsec ESP Tunnel encapsulation overview. HMO and Options are described below the PDB diagram.

**Table 9-15. IPsec ESP Tunnel encapsulation PDB**

| | Descriptor Header (1 or 2 words) | | | | | |
|---|---|---|---|---|---|---|
| PDB Word 0 | HMO<br>(4 bits) | Reserved<br>(4 bits) | Next Header<br>(8 bits) | AOIPHO<br>(8 bits) | Options<br>(8 bits) | |
| PDB Word 1 | Optional Extended Sequence Number (ESN) | | | | | DECO writes back to PDB as needed |
| PDB Word 2 | Sequence Number | | | | | |
| PDB Word 3 | Cipher-suite-specific portion of PDB | | | | | for CBC-mode see CBC IV Format<br>for CTR-mode see CTR IV Format<br>for CCM-mode see CCM IV Format<br>for GCM-mode see GCM IV Format |
| PDB Word 4 | | | | | | |
| PDB Word 5 | | | | | | |
| PDB Word 6 | | | | | | |
| PDB Word 7 | SPI | | | | | |
| PDB Word 8 | Reserved | | | Opt IP Header Length | | |
| PDB Word 9 | Optional IP Header (bytes 0-3) | | | | | |
| PDB Word 10 | Optional IP Header (bytes 4-7) | | | | | ID field is incremented and written back as needed |
| PDB Word 11 | Optional IP Header (bytes 8-11) | | | | | |

*Table continues on the next page...*

## Table 9-15. IPsec ESP Tunnel encapsulation PDB (continued)

| | | |
|---|---|---|
| PDB Word 12 | Optional IP Header (bytes 12-15) | |
| PDB Word 13 + | Optional IP Header (bytes 16+) | |

## Table 9-16. IPsec ESP Tunnel encapsulation PDB, format of the options byte

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | IVsrc | ESN | OIHI | | NAT | NUC |

## Table 9-17. IPsec ESP Tunnel encapsulation, description of the options byte

| Field | Description |
|---|---|
| 7-6 | Reserved |
| 5<br>IVsrc | IV Source<br>0 - Chained IV Stored in PDB<br>1 - Random IV fetched from RNG prior to starting encapsulation |
| 4<br>ESN | Extended Sequence Number<br>0 - ESN not included in ICV computation. Optional ESN field of PDB is reserved in this case.<br>1 - ESN is copied from PDB and used for ICV computation but is not written to the output frame. ESN is incremented as if part of the sequence number and written back to the PDB as required. |
| 3-2<br>OIHI | Outer IP Header Included<br>00 - No Outer IP Header Provided<br>01 - Outer IP Header from input frame -- The input frame provided will include *two* IP headers; first the outer IP header, then the IP header of the datagram being encapsulated. The output frame will include the Outer IP Header (modified as necessary), followed by the ESP Header, followed by the encrypted datagram<br>10 - Outer IP Header Referenced by PDB - The output frame will include an outer IP header fetched from the address in the appropriate PDB field. Note the extra memory reads associated with fetching the outer IP header from independent memory cannot be prefetched or otherwise optimized, so a significant performance penalty results from using this option.<br>11 - Outer IP Header from PDB - The output frame will include the Outer IP Header (modified as necessary) extracted from the appropriate fields of the PDB. |
| 1<br>NAT | Enable UDP-encapsulated-ESP as defined by RFC 3948 for traversing Network Address Translators (NATs).<br>0 - no UDP-encapsulated-ESP<br>1 - UDP-encapsulated-ESP enabled. |
| 0<br>NUC | enables NAT checksum<br>0 - If NAT is enabled, then no UDP checksum is computed, and the resulting UDP header uses a zero-value checksum.<br>1 - If NAT is enabled, then a UDP checksum is computed and included in the UDP header.<br>If NAT is not enabled, NUC must be zero, else an error will be generated. |

This figure shows the format of the HMO field.

Table 9-18. IPsec ESP Tunnel encapsulation PDB, format of the HMO field

| 31 | 30 | 29 | 28 |
|---|---|---|---|
| Reserved | DFC | DTTL | SNR |

Table 9-19. IPsec ESP Tunnel encapsulation PDB, description of the HMO field

| Field | Description |
|---|---|
| 31 | Reserved |
| 30<br>DFC | Copy DF bit<br>0 - Do not copy DF bit<br>1 - If an IPv4 tunnel mode outer IP header is coming from the PDB, copy the DF bit from the inner IP header to the outer IP header. If not in tunnel mode or the outer IP header from the PDB is not included, setting DFC = 1 causes a Protocol PDB error. |
| 29<br>DTTL | Decrement TTL (Hop Limit)<br>0 - Do not decrement<br>1 - Based on the value of the Options Byte IPvsn field: if IPv4, decrement the inner IP header TTL field (byte 8) and update the IPv4 header checksum; if IPv6 decrement the inner IP header Hop Limit field (byte 7). If TTL is decremented below 0, an error is generated. |
| 28<br>SNR | Sequence Number Rollover enable<br>0 - Sequence Number (as extended by ESN) does not roll over; an error is generated if a rollover is attempted<br>1 - Sequence Number (as extended by ESN) permitted to roll over |

#### 9.1.3.5 Common PDB format descriptions for IPsec ESP Tunnel decapsulation

IPsec ESP Tunnel decapsulation uses a mostly-common Protocol Data Block (PDB) format to maintain certain state and security association information. To complete decapsulation, SEC requires access to SPI, Sequence Number, Extended Sequence Number (if used). SPI and Sequence Number can be used as extracted from the input frame, but the Extended Sequence Number is not included in the encapsulated datagram. PDB words 1 and 2 are reserved for some cipher-suite-specific material, such as the Salt used by AES-GCM. These words are reserved for CBC-based cipher suites because no other information is required. All of this is common with the IPsec ESP Transport (and legacy tunnel) form of the decapsulation PDB. What is not common is the field AOIPHO (Actual Outer IP Header Offset), which provides an offset to the actual outer IP header in the input frame; allowing material preceeding the IP header (such as an ethernet header) to be passed from input frame to output frame. This field replaces the ESP Transport PDB field called NH Offset, which is used only for Transport mode processing.

This PDB diagram shows the common form, and the ESP Tunnel definitions of the Options and HMO bits follows afterwards. For the cipher-suite-specific portion, additional details can be found as follows:

- For AES-CBC and DES-CBC specific IV format, refer to IPsec ESP decapsulation CBC-specific PDB segment format descriptions
- For AES-CTR specific Counter and IV format, refer to IPsec ESP decapsulation AES-CTR-specific PDB segment format descriptions
- For AES-CCM specific data format, refer to IPsec ESP decapsulation AES-CCM-specific PDB segment format descriptions
- For AES-GCM specific Salt and IV format, refer to IPsec ESP decapsulation AES-GCM-specific PDB segment format descriptions

Fields shown here should be programmed as appropriate per the negotiated tunnel parameters. The Anti-replay scorecard should be initialized with zeros. For more detail on fields other than HMO and Options, see IPsec ESP tunnel decapsulation overview. HMO and Options are described below the PDB diagram.

**Table 9-20. IPsec ESP Tunnel decapsulation PDB**

| | Descriptor Header (1 or 2 words) | | | | |
|---|---|---|---|---|---|
| PDB Word 0 | HMO<br>(4 bits) | IP Header Length<br>(12 bits) | AOIPHO<br>(8 bits) | Options<br>(8 bits) | |
| PDB Word 1 | Cipher-suite-specific portion of PDB | | | | unused for CBC-mode<br>for CTR-mode see CTR IV Format<br>for CCM-mode see CCM IV Format<br>for GCM-mode see GCM IV Format |
| PDB Word 2 | | | | | |
| PDB Word 3 | Optional Extended Sequence Number (ESN) | | | | DECO writes back to PDB as needed |
| PDB Word 4 | Sequence Number | | | | |
| PDB word 5 | anti-replay scorecard 1 [present for any size of anti-replay window] | | | | |
| PDB word 6 | anti-replay scorecard 2 [present if anti-replay window size exceeds 32] | | | | |
| PDB word 7 | anti-replay scorecard 3 [present if anti-replay window size exceeds 64] | | | | |
| PDB word 8 | anti-replay scorecard 4 [present if anti-replay window size exceeds 96] | | | | |

**Table 9-21. IPsec ESP Tunnel Common decapsulation PDB, format of the options byte**

| 7-6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| ARS | TECN | ESN | rsv | rsv | rsv | ETU |

## Table 9-22. IPsec ESP Tunnel Common decapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| 7-6<br>ARS | Anti-replay window size.<br>00 - No anti-replay window<br>01 - 32-entry anti-replay window<br>10 - 128-entry anti-replay window<br>11 - 64-entry anti-replay window |
| 5<br>TECN | Tunnel ECN<br>0 - ECN Tunnelling disabled<br>1 - ECN Tunnelling enabled per RFC 6040. |
| 4<br>ESN | Include Extended Sequence Number:<br>0 - ESN not used. Optional ESN field of PDB is reserved in this case.<br>1 - ESN is used for ICV and anti-replay computations. The ESN is not found in the input frame |
| 3 | Reserved |
| 2-1 | Reserved |
| 0<br>ETU | EtherType update<br>0 - AOIPHO-defined material preceeding outer IP header copied as is to output frame<br>1 - AOIPHO-defined material preceeding outer IP header copied, except last two bytes are updated to proper EtherType value for the IP header following -- 0800h for IPv4, and 86ddh for IPv6 |

## Table 9-23. IPsec ESP Tunnel Common decapsulation PDB, format of the HMO field

| 31 | 30 | 29 | 28 |
|---|---|---|---|
| ODF | DFV | DTTL | DSC |

## Table 9-24. IPsec ESP Tunnel Common decapsulation PDB, description of the HMO field

| Field | Description |
|---|---|
| 31<br>ODF | ODF -- Override DF bit in IPv4 header of decapsulated output frame<br>0 - DF bit the IPv4 header in the output frame is identical to that in the input frame (encapsulated header in the input frame if in tunnel mode)<br>1 - DF bit in the IPv4 header in the output frame is replaced with the DFV value as shown below.<br>If IPv6 is selected, then this bit is reserved and must be zero. |
| 30<br>DFV | DFV -- DF bit Value<br>If ODF (see above) is set, this bit replaces whatever value would have otherwise been placed in the IPv4 Header DF bit field.<br>If ODF is not set, then this bit is reserved and must be zero. |
| 29<br>DTTL | Decrement TTL (Hop Limit)<br>0 - Do not decrement<br>1 - Based on the value of the Options Byte IPvsn field if IPv4, decrement the inner IP header TTL field (byte 8) and update the IPv4 header checksum; if IPv6 decrement the inner IP header Hop Limit field (byte 7). If TTL is decremented below 0, an error is generated. |

*Table continues on the next page...*

Table 9-24. IPsec ESP Tunnel Common decapsulation PDB, description of the HMO field (continued)

| Field | Description |
|---|---|
| 28<br>DSC | DiffServ Copy<br>0 - Do not copy<br>1 - Copy the IPv4 TOS or IPv6 Traffic Class byte from the outer IP header to the inner IP header. |

#### 9.1.3.6 Overriding ESP Tunnel PDB content with the DECO Protocol Override Register

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For IPsec ESP Tunnel encapsulation and decapsulation, several fields are maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired values in the job descriptor. For more information, see Job Ring interface. When using the Queue Manager Interface, QI builds the job descriptor with a LOAD IMMEDIATE to the DPOVRD register with the value of the STATUS/CMD field in the FD. For more information, see Queue Manager Interface (QI).

When DPOVRD is selected for use, SEC then uses the following values as specified in DPVORD instead of as specified in the PDB for both encapsulation and decapsulation:

- Outer IP Header Material Length
- AOIPHO
- OIMIF (encapsulation only)
- Next Header (encapsulation only)

Note that the values in DPOVRD in no way affect the values stored in the descriptor's PDB. DPOVRD works the same way regardless of the format of the IPsec encapsulation and decapsulation PDB, although the PDB's format varies depending on the chosen cipher suite. For details on how each of these fields are used, please refer to the appropriate section -- either PDB format for IPsec ESP Tunnel encapsulation or Common PDB format descriptions for IPsec ESP Tunnel decapsulation.

The following table shows the form of the DPOVRD register used for encapsulation. A separate format is used for decapsulation, and is shown below.

### Table 9-25. IPsec ESP Tunnel format of the DPOVRD register for encapsulation

<table>
<tr><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th></tr>
<tr><td>OVRD</td><td colspan="3">Reserved</td><td colspan="12">Outer IP Header Material Length</td></tr>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>OIMIF</td><td></td><td colspan="6">AOIPHO</td><td colspan="8">Next Header</td></tr>
</table>

### Table 9-26. IPsec ESP Tunnel description of the DPOVRD register for encapsulation

| Field | Description |
|---|---|
| 31<br>OVRD | Selects whether DPOVRD overrides the values specified in the PDB.<br>0: Use the PDB as provided<br>1: Override values in PDB by using values in DPOVRD |
| 30-28 | Reserved |
| 27-16<br>Outer IP Header Material Length | The length, in bytes, of the portion of the IP header that is not encrypted.<br>If IP Header Length = 0, the entire IP header is encrypted. In transport mode this indicates to SEC that this particular packet has an IP header that is not of typical length, so that SEC does not authenticate or encrypt any of the atypical IP headers found in the input frame. |
| 15<br>OIMIF | Outer IP Header Material in Input Frame<br>0: Use Outer IP Header Material as specified by PDB<br>1: For encapsulating this datagram, use Outer IP Header material found in input frame. Length specified by DPOVRD register value Outer IP Header Material Length; AOIPHO specifies offset into material for actual Outer IP Header. |
| 14 | Reserved |
| 13-8<br>AOIPHO | Actual Outer IP Header Offset<br>This allows the Outer IP Header Material to be preceeded by some additonal content, such as an Ethernet header. The value in this field indicats where the actual Outer IP Header starts in the material provided. |
| 7-0 | Next Header<br>Used in the Next Header field of the encapsulated payload. |

The following table shows the form of the DPOVRD register used for decapsulation. A separate format is used for encapsulation, and is shown above.

### Table 9-27. IPsec ESP Tunnel format of the DPOVRD register for decapsulation

<table>
<tr><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th></tr>
<tr><td>OVRD</td><td colspan="11">Reserved</td><td colspan="4">AOIPHO</td></tr>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="4">AOIPHO (continued)</td><td colspan="12">Outer IP Header Material Length</td></tr>
</table>

### Table 9-28. IPsec ESP Tunnel description of the DPOVRD register for decapsulation

| Field | Description |
|---|---|
| 31 | Selects whether DPOVRD overrides the values specified in the PDB. |

*Table continues on the next page...*

**Table 9-28. IPsec ESP Tunnel description of the DPOVRD register for decapsulation (continued)**

| Field | Description |
|---|---|
| OVRD | 0: Use the PDB as provided<br>1: Override values in PDB by using values in DPOVRD |
| 30-20 | Reserved |
| 19-12<br>AOIPHO | Actual Outer IP Header Offset<br>This allows the Outer IP Header Material to be preceeded by some additonal content, such as an Ethernet header. The value in this field indicats where the actual Outer IP Header starts in the material provided. |
| 15<br>OIMIF | Outer IP Header Material in Input Frame<br>0: Use Outer IP Header Material as specified by PDB<br>1: For encapsulating this datagram, use Outer IP Header material found in input frame. Length specified by DPOVRD register value Outer IP Header Material Length; AOIPHO specifies offset into material for actual Outer IP Header. |
| 11-0<br>Outer IP Header Material Length | The length, in bytes, of the portion of the IP header that is not encrypted.<br>If IP Header Length = 0, the entire IP header is encrypted. In transport mode this indicates to SEC that this particular packet has an IP header that is not of typical length, so that SEC does not authenticate or encrypt any of the atypical IP headers found in the input frame. |

### 9.1.3.7 IPsec ESP encapsulation CBC-specific PDB segment format descriptions

These figures show the format of the segment of the IPsec ESP encapsulation PDB used with CBC-based cipher suites.

This segment is common to both ESP Tunnel and ESP Transport PDB forms.

**Table 9-29. CBC-specific segment of IPsec ESP encapsulation PDB**

| | | |
|---|---|---|
| | common PDB above<br>(common encapsulation PDB shown for ESP Transport in Table 9-3) and for ESP Tunnel in Table 9-15 | |
| PDB Word 3 | IV bytes 0-3 | DECO writes back to PDB as needed |
| PDB Word 4 | IV bytes 4-7 | |
| PDB Word 5 | IV bytes 8-11<br>(AES only - reserved for DES) | |
| PDB Word 6 | IV bytes 12-15<br>(AES only - reserved for DES) | |
| | more common PDB below | |

### 9.1.3.8 IPsec ESP encapsulation AES-CTR-specific PDB segment format descriptions

These figures show the format of the portion of the IPsec ESP encapsulation PDB used with AES-CTR-based cipher suites.

This segment is common to both ESP Tunnel and ESP Transport PDB forms.

**Table 9-30. AES-CTR specific segment of IPsec ESP encapsulation PDB**

| | | |
|---|---|---|
| | common PDB above<br>(common encapsulation PDB shown for ESP Transport in Table 9-3) and for ESP Tunnel in Table 9-15 | |
| PDB Word 3 | Counter Nonce | |
| PDB Word 4 | Counter Initial Count | |
| PDB Word 5 | IV bytes 0-3 | DECO writes back to PDB as needed |
| PDB Word 6 | IV bytes 4-7 | |
| | more common PDB below | |

### 9.1.3.9 IPsec ESP encapsulation AES-CCM-specific PDB segment format descriptions

These figures show the format of the portion of the IPsec ESP encapsulation PDB used with AES-CCM cipher suites.

This segment is common to both ESP Tunnel and ESP Transport PDB forms.

**Table 9-31. AES-CCM specific segment of IPsec ESP encapsulation PDB**

| | | | | |
|---|---|---|---|---|
| | common PDB above<br>(common encapsulation PDB shown for ESP Transport in Table 9-3) and for ESP Tunnel in Table 9-15 | | | |
| PDB Word 3 | Reserved<br>(8 bits) | Salt<br>(24 bits) | | |
| PDB Word 4 | $B_0$ Flags<br>(8 bits) | $CTR_0$ Flags<br>(8 bits) | Counter Initial Count<br>(16 bits) | |
| PDB Word 5 | IV bytes 0-3 | | | |
| PDB Word 6 | IV bytes 4-7 | | | |
| | more common PDB below | | | |

### 9.1.3.10 IPsec ESP encapsulation AES-GCM-specific PDB segment format descriptions

These figures show the format of the portion of the IPsec ESP encapsulation PDB used with AES-GCM-based cipher suites.

This segment is common to both ESP Tunnel and ESP Transport PDB forms.

**Table 9-32. AES-GCM specific segment of IPsec ESP encapsulation PDB**

| | | |
|---|---|---|
| | common PDB above<br>(common encapsulation PDB shown for ESP Transport in Table 9-3) and for ESP Tunnel in Table 9-15 | |
| PDB Word 3 | Salt | |
| PDB Word 4 | Reserved | |
| PDB Word 5 | IV bytes 0-3 | |
| PDB Word 6 | IV bytes 4-7 | |
| | more common PDB below | |

### 9.1.3.11 IPsec ESP decapsulation CBC-specific PDB segment format descriptions

For both IPsec ESP decapsulation PDBs -- ESP Tunnel and ESP Transport (and legacy tunnel) -- the two words reserved for cipher-specific additions are unused and therefore reserved.

**Table 9-33. CBC-specific segment of IPsec ESP decapsulation PDB**

| | | |
|---|---|---|
| | descriptor header above | |
| PDB Word 1 | Reserved | |
| PDB Word 2 | Reserved | |
| | more common PDB below<br>(common decapsulation PDB shown for ESP Transport in Table 9-8) and for ESP Tunnel in Table 9-20 | |

### 9.1.3.12 IPsec ESP decapsulation AES-CTR-specific PDB segment format descriptions

The initial counter value required for decapsulation using counter mode is built using the Nonce value assigned to the security association and the IPsec IV extracted from the ESP header. The Nonce value is provided in the AES-CTR-specific portion of the decapsulation PDB in word 1. The lower 4 bytes of the initial counter value is specified using PDB word 2. RFC 3686 calls this segment the Block Counter, and specifies an initial value of 00000001h.

This PDB segment is common to both ESP Tunnel and ESP Transport PDBs.

**Table 9-34. AES-CTR specific segment of IPsec ESP decapsulation PDB**

| | descriptor header above | |
|---|---|---|
| PDB Word 1 | Counter Nonce | |
| PDB Word 2 | Counter Initial Count | |
| | more common PDB below<br>(common decapsulation PDB shown for ESP Transport in Table 9-8) and for ESP Tunnel in Table 9-20 | |

### 9.1.3.13 IPsec ESP decapsulation AES-CCM-specific PDB segment format descriptions

These figures show the format of the portion of the IPsec ESP decapsulation PDB used with AES-CCM cipher suites. This form is common to both the IPsec ESP Tunnel and ESP Transport PDBs.

**Table 9-35. AES-CCM specific segment of IPsec ESP decapsulation PDB**

| | descriptor header above | | | |
|---|---|---|---|---|
| PDB Word 1 | Reserved<br>(8 bits) | Salt<br>(24 bits) | | |
| PDB Word 2 | $B_0$ Flags<br>(8 bits) | $CTR_0$ Flags<br>(8 bits) | Counter Initial Count<br>(16 bits) | |
| | more common PDB below<br>(common decapsulation PDB shown for ESP Transport in Table 9-8) and for ESP Tunnel in Table 9-20 | | | |

### 9.1.3.14 IPsec ESP decapsulation AES-GCM-specific PDB segment format descriptions

The GCM IV required for decapsulation is built using the Salt value assigned to the security association and the IPsec IV extracted from the ESP header. The salt value is provided in the GCM-specific portion of the decapsulation PDB in word 1 for both ESP Tunnel and ESP Transport PDB forms.

Table 9-36. AES-GCM specific segment of IPsec ESP decapsulation PDB

| | descriptor header above | |
|---|---|---|
| PDB Word 1 | Salt | |
| PDB Word 2 | Reserved | |
| | more common PDB below<br>(common decapsulation PDB shown for ESP Transport in Table 9-8) and for ESP Tunnel in Table 9-20 | |

## 9.1.4 IPsec ESP Transport (and Legacy Tunnel) encapsulation overview

SEC supports tunnel and transport encapsulation of either complete IP packets or of IP payloads, per IPsec ESP requirements. The IPsec ESP Transport hardware (as specified in the PROTOCOL OPERATION COMMAND with PROTID=01) is designed to support Transport mode ESP datagram encapsulation. Limited Tunnel mode support is also provided, but may be deprecated in future SEC hardware revisions; additional ESP Tunnel mode support is (and will continue to be) supported by the IPsec ESP Tunnel threads.

Once the payload is identified, cryptographic encapsulation proceeds identically for Transport mode datagrams, for Tunnel mode datagrams when using the ESP Transport (and legacy tunnel) encapsulation thread, and for Tunnel mode datagrams when using the ESP Tunnel encapsulation thread. This procedure is described in IPsec ESP Cryptographic Encapsulation.

As part of encapsulation, SEC updates the length field of the IP header. This field has different characteristics depending on whether your device uses IPv4 or IPv6. The following table summarizes these differences:

Table 9-37. Differences in the length field by IPrev

| Characteristic | IPv4 | IPv6 |
|---|---|---|
| Name | IP Total Length | Payload Length |
| Byte range | 2:3 | 4:5 |
| What includes | Length of IP header itself | Length of all extension headers; does not include the length of the IP header |

**NOTE**

To handle the differences between IP revs in terms of what the length field includes, the ESP Tunnel protocol assumes that the length field provided with the IP header is appropriate for IPv4

> or IPv6. In other words, it assumes that the IP header Total Length field (IPv4) includes the length of the IP header and options whereas the IP header Payload Length field (IPv6) does not include the length of the IP header and extensions. (ESP Transport continues to use the PDB option bit that indicates IPversion and ignores the IPversion field within the IP header altogether.)

Note that SEC can optionally update the header checksum based on other changes made to the header. This update is valid only if the checksum was correct for the header prior to any other changes made.

### 9.1.4.1 Encapsulating the IP header in tunnel mode

For tunnel mode, the IP header is encapsulated as part of the payload. SEC can prepend a new IP header from the PDB or the input frame after it computes the proper length value. It computes this value as follows:

SEC-added ESP Header (SPI, Seq Num, and IV) length + the payload + ESP Trailer (padding, Pad Len, N, and ICV) length = proper length value

Note that for the final length field to be correct, the correct length value must have been programmed into the PDB field titled "Opt IP Hdr Length".

For IPv4, prior to RFC 6864, the IP Header identification field was required to be unique for each packet transmitted. Therefore, if SEC is prepending an IP Header from the PDB, it increments by 1 the bytes corresponding to the IP Header Identification field in the PDB, per RFC 791.

### 9.1.4.2 Encapsulating the IP header in transport mode

No new IP header is prepended in transport mode. Instead, SEC inserts all these added fields after the received IP header and updates the length to account for all the bytes SEC adds: those in the ESP Header and the ESP Trailer. Note that SEC does not add bytes for the payload length because it should be included in the original IP header length field.

### 9.1.4.3 Process for IPsec ESP Transport (and Legacy Tunnel) encapsulation

This figure shows an example of the IPsec ESP Transport (and Legacy Tunnel) encapsulation processing-sequence.

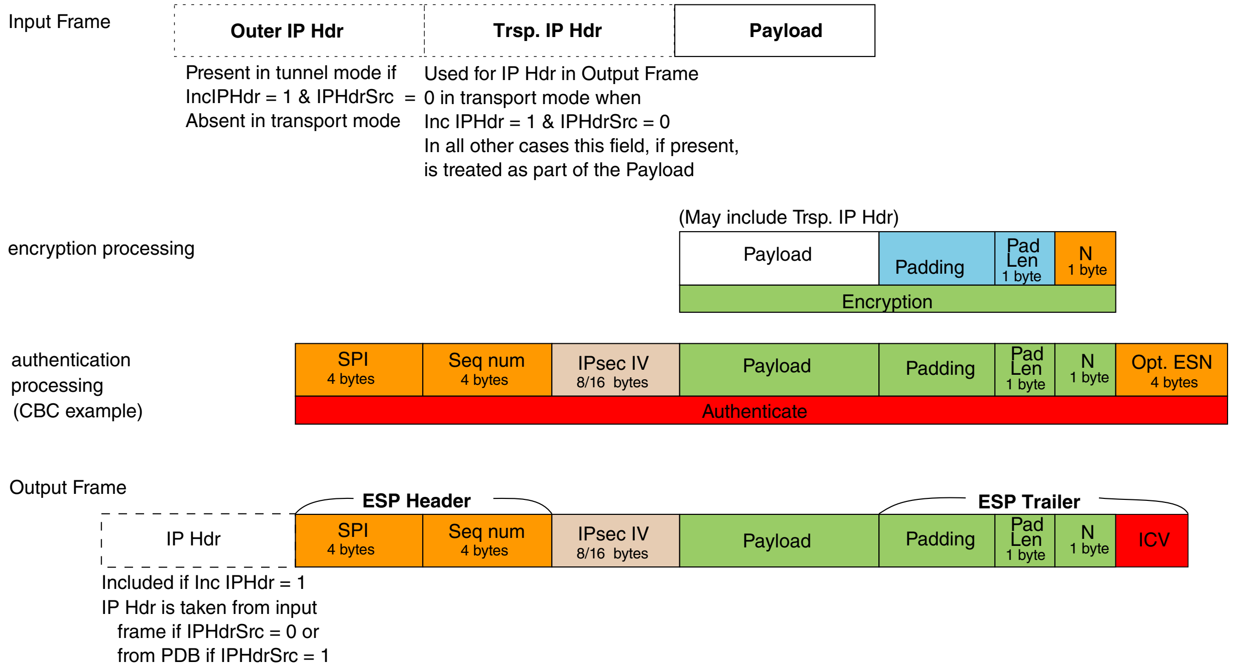


**Figure 9-2. Example IPsec ESP Transport (and Legacy Tunnel) encapsulation processing sequence (DES-CBC, AES-CBC, or AES-CTR)**

As shown above, processing begins when SEC receives an input frame. Note that the Transport IP Header is optional for transport mode because SEC does not authenticate or encrypt the IP header. The transport IP header may be omitted in transport mode by clearing IncIPHdr, setting IPHdrSrc, or clearing IP Header Length. For tunnel mode, the input frame must include an IP header because SEC must authenticate and encrypt this header as part of the payload.

When a IP header is included in transport mode, the N (Next Header) byte receives special treatment if the NH_OFFSET byte of the PDB is set to a non-zero value, as follows:

- If transport mode ESP with IPv4 for any non-zero value of NH_OFFSET (typically set to 01h), the N byte used in the ESP trailer comes from byte 9 of the IP header, and byte 9 of the IP header is replaced with the Next Header byte from the PDB.
- If transport mode ESP with IPv6 and NH_OFFSET = 01h, the N byte used in the ESP trailer comes from byte 6 of the IP header, and byte 6 of the IP header is replaced with the Next Header byte from the PDB.
- If transport mode ESP with IPv6 and NH_OFFSET > 01h, the N byte used in the ESP trailer comes from byte (NH_OFFSET x 8) of the IP header, and byte

(NH_OFFSET x 8) of the IP header is replaced by the Next Header byte from the PDB.

- In all cases, if NH_OFFSET contains zero, no swap occurs and the ESP trailer N byte is copied directly from the PDB Next Header byte, as in tunnel mode.

As shown in Process for IPsec ESP Transport (and Legacy Tunnel) encapsulation, the output frame results from SEC IPsec encapsulation regardless of the cipher suite used.

## 9.1.5 IPsec ESP Cryptographic Encapsulation

Construction of the ESP Header, Payload and ESP trailer is common for both ESP Tunnel and Transport modes, as are the cryptographic processes involved in encapsulating. However, the procedure and set up of cryptographic context differs, depending upon the cipher suite chosen.

### 9.1.5.1 Process for IPsec encapsulation when using AES-CBC or DES-CBC

This figure shows stages of IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation.

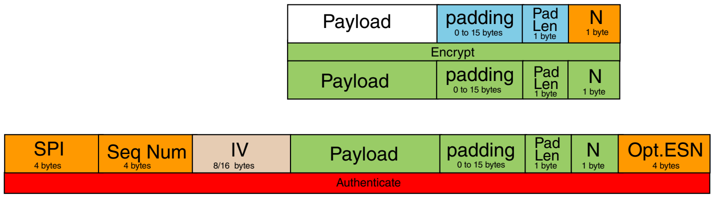


**Figure 9-3. Stages of IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-CBC or DES-CBC**

When using the AES-CBC or DES-CBC cipher suite, SEC performs IPsec encapsulation by doing the following:

1. Begins by prepending authenticate-only data (the SPI and Seq Num found in the PDB) to the payload. The authenticate-only data is passed to the output frame for transmission and in parallel is passed to the authentication CHA.
2. Appends 8- or 16-byte IV (IV size is the cipher block size) after the sequence number and sends it to the authentication CHA (typically a Class 2 CHA); in parallel, it sends it to the output frame.

**NOTE**

> The IV can come from two possible sources, depending on the setting of the IVsrc field in the PDB Options Byte:
>
> - The RNG can generate it randomly for a packet (IVsrc = 1).
> - The IV can be a chained IV, meaning that the final block of ciphertext from the previous packet is used as the IV for the next packet (IVsrc = 0).

3. Writes the IV also into the first locations of the Context Register for the Class 1 CHA
4. After encryption is completed, writes the last block of ciphertext back to the PDB and Seq Num is incremented and updated in the PDB in memory

After the IV is in place, the Payload itself is fetched from the Input Frame and encrypted. The result of the encryption process is pushed onto the output frame, and is also pushed into the authentication CHA (normally Class 2). SEC generates the appropriate padding of monotonically increasing bytes, the first byte having the value 01h. SEC computes the Pad Length (padding such that Next Header is the last byte in a cryptographic block), and it appends Next Header (N) as found in the PDB. The padding, Pad Length, and Next Header are encrypted and authenticated immediately following the payload.

Optionally, IPsec can use an extended sequence number (ESN) that is authenticated but not transmitted. If an ESN is found in the PDB, it is the last thing given to the authentication CHA. The ESN is incremented whenever the Seq Num rolls over.

For AES-CBC and DES-CBC, the update to the header length field reflects the addition of the following:

- 4-byte SPI
- 4-byte Seq Num
- 8 or 16-byte IV
- Padding (0-7 bytes for DES, 0-15 bytes for AES)
- 1-byte pad length
- 1-byte Next (N) field
- the ICV (the length of which is dependent on the cipher suite chosen)

Null Authentication may be chosen along with AES-CBC or DES-CBC confidentiality. If null authentication is chosen, then encryption proceeds as described before, steps that cause data to be authenticated are skipped. As no ICV is genererated, writing of ICV to the output frame is also skipped.

### 9.1.5.2 Process for IPsec encapsulation when using AES-CTR

This figure shows IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-CTR.

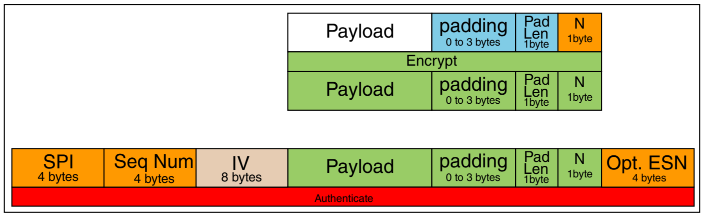


**Figure 9-4. IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-CTR**

When using an AES-CTR-based cipher suite, SEC performs IPsec encapsulation by doing the following:

1. Begins by prepending authenticate-only data to the payload
2. Passes the SPI and sequence number found in the PDB to the output frame for transmission and to the authentication CHA
3. Appends the IV after the sequence number for authentication (typically a Class 2 operation) and the Output Frame.

**NOTE**

The IV can come from two possible sources, depending on the setting of the IVsrc field in the PDB Options byte

- The RNG can generate it for a packet (IVsrc = 1)
- The IV value within the PDB can be treated as a pseudo-sequence number (IVsrc = 0).

If the second option is chosen, the actual sequence number and the pseudo-sequence number are incremented and updated after use in the PDB in memory.

4. Does not write the 8-byte IV to the Class 1 Context Register directly, but combines it with a 4-byte nonce value and a 4-byte initial count constant (0000 0001h per RFC 3686). Both constants are static in the PDB.

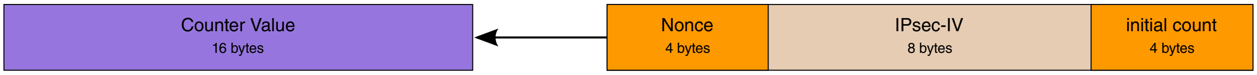


**Figure 9-5. Initial counter construction**

5. Writes the 16-byte counter value into the counter value segment of the Class 1 Context Register, at offset 16.

6. Following generation of the authentication-only data prepended to the payload, fetches the payload itself from the input frame and encrypts it. The result of the encryption process is pushed onto the output frame, and is also pushed into the authentication CHA (normally Class 2).
7. Applies monotonically increasing padding bytes, followed by a Pad Len byte, such that the end of the frame after ICV ends on a 4-byte boundary. SEC next appends Next Header (N) as found in the PDB. The Next Header is encrypted and authenticated immediately following the payload.

Optionally, IPsec can use an extended sequence number (ESN) that is authenticated but not transmitted. If an ESN is found in the PDB, it is the last thing given to the authentication CHA. The ESN is incremented whenever the Seq Num rolls over.

The update to the IP Header length field reflects the addition of the following:

- 4-byte SPI
- 4-byte Seq Num
- 8-byte IPsec IV
- 0-3 bytes of padding
- 1-byte pad length
- Next byte
- The ICV (the length of which is dependent on the cipher suite chosen).

Null Authentication may be chosen along with AES-CTR confidentiality. If null authentication is chosen, then encryption proceeds as described before, steps that cause data to be authenticated are skipped. As no ICV is genererated, writing of ICV to the output frame is also skipped.

#### 9.1.5.3 Process for IPsec encapsulation when using AES-CCM

When using the AES-CCM cipher suite, SEC performs IPsec encapsulation by doing the following:

1. Constructs the CCM $B_0$ and Initial Counter ($CTR_0$) values from Flag bytes, the Salt, and the IPsec IV as follows:
   - A nonce is constructed by appending the 8-byte IPsec IV to a 3-byte salt value. The salt value is a static value stored in the PDB. The IPsec IV is either a pseudo-sequence number in the PDB that is incremented after use or a random number.

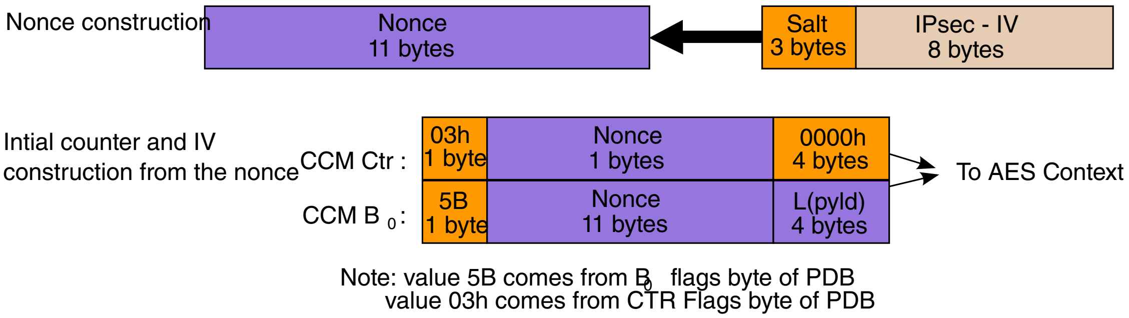


**Figure 9-6. IPsec AES-CCM context construction**

- To construct the Initial Counter value, SEC prepends the nonce with the counter field flags, which is a static byte from the PDB (with a value of 03h), and appends another 4 bytes that remain static in the PDB.
- To construct the CCM $B_0$, SEC uses the $B_0$ flags byte of the PDB, whose value must be selected by the user according to the size of ICV transmitted.
  - For an 8-byte ICV, select a value of 5Bh.
  - For a 12-byte ICV, select a value of 6Bh.
  - For a 16-byte ICV, select a value of 7Bh.
- Note that the cipher suite value in the Protocol Command selects the ICV size.

2. Writes the CCM $B_0$ and Initial Counter values to the Class 1 Context Register (16 bytes starting with offset zero receive the CCM $B_0$; the CCM Ctr immediately follows).
3. After programming the Class 1 Context Registers, prepends authenticate-only data (AAD) to the payload (see the following figure).

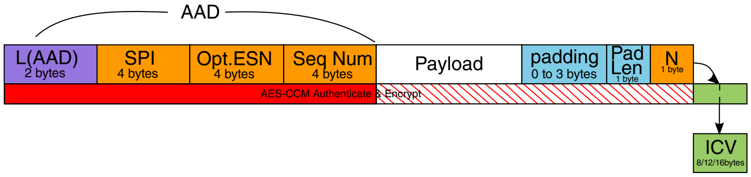


**Figure 9-7. IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-CCM**

a. The AES-CCM mode encapsulation data starts with the formatted AAD, which consists of a 2-byte field indicating the length of AAD field, and the ESP Header, which consists of:
   - SPI and Seq Num from the PDB (which are also passed to the output frame)
   - ESN from the PDB and if enabled (ESN is not passed to the output frame).

b. The formatted AAD includes zero-padding to the nearest block (16-byte boundary) and is passed into the input-data FIFO, but not into the output frame.

c. The 8-byte IPsec IV is appended after the sequence number to the output frame, but is not written to the input-data FIFO.

4. Following generation of the formatted AAD, fetches the payload from the input frame and encrypts it.
5. The result of the encryption process is pushed onto the output frame; applies monotonically increasing padding bytes, followed by a Pad Len byte, such that the end of the frame after ICV ends on a 4-byte boundary.
6. Appends Next Header (N) as found in the PDB; N is encrypted and authenticated immediately following the payload.

Optionally, IPsec can use an extended sequence number (ESN) that is authenticated but not transmitted. For combination algorithms (that is, unlike the AES-CBC), the ESN is appended to the AAD after the SPI. ESN, which is found in the PDB, is incremented whenever the sequence number rolls over.

The update to the IP Header length field reflects the addition of the following:

- 4-byte SPI
- 4-byte Seq Num
- 8-byte IPsec IV
- 0-3 bytes of padding
- 1-byte pad length
- The Next byte
- The ICV (the length of which is dependent on the cipher suite chosen. The ICV may be 8, 12, or 16 bytes, depending upon the PROTINFO code provided.)

### 9.1.5.4 Process for IPsec encapsulation when using AES-GCM

When using the AES-GCM cipher suite, SEC performs IPsec encapsulation by doing the following:

1. Constructs the nonce, which is a 12-byte GCM IV.
   - The GCM IV is created by appending the IPsec IV value to the 4-byte Salt value (see Figure 9-8).
   - The Salt is a static value found in the PDB.
   - The IPsec IV value is either a pseudo-sequence number in the PDB that is incremented after use or it is a random number.
   - Per RFC 4106, Salt concatenated with IV creates a value called the Nonce. This same value is called an IV (a GCM-IV) by specifications defining AES-GCM, such as NIST document SP800-38D.

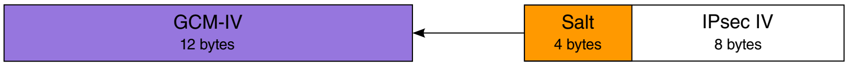


**Figure 9-8. IPsec GCM-IV building**

2. DECO writes the GCM-IV to the input-data FIFO after padding it with zeros to the nearest block (16-byte) boundary.
3. The 8- or 12-byte authentication-only data (AAD) is zero padded by the DECO to the nearest block (16-byte) boundary and then written to the input-data FIFO
   - The AAD consists of the 4-byte SPI, the 4-byte ESN (if present), and the 4 byte Seq Num, concatenated together (see Figure 9-9).
   - The SPI and Seq Num are written to the output frame (in addition to the input-data FIFO) and followed by the IPsec IV.
   - ESN is written to the input-data FIFO only and not to the output frame.

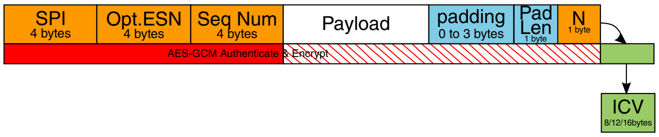


**Figure 9-9. IPsec ESP Transport (and Legacy Tunnel) encryption and authentication for encapsulation with AES-GCM**

4. Inputs the payload, monotonically increasing padding bytes, a Pad Len byte and the next header (N) to the input-data FIFO, and encrypts these.
5. Pops the output-data FIFO off the encrypted result and writes it to the output frame.
6. Writes AES-GCM ICV to the output frame.

The update to the IP Header length field reflects the addition of the following:

- 4-byte SPI
- 4-byte Seq Num
- 8 byte IPsec IV
- 0-3 bytes of padding
- 1-byte pad length
- the Next byte
- ICV (the length of which is dependent on the cipher suite chosen by the PROTINFO field -- see Table 7-53)

### 9.1.6 IPsec ESP Transport (and Legacy Tunnel) decapsulation procedure overview

SEC is designed to decapsulate received IPsec ESP Transport (and limited Tunnel mode) packets. The following figure provides an illustration of the procedure for both modes. For mode-specific details, see IPsec ESP Transport Mode outer IP header decapsulation procedure or IPsec ESP Transport (and Legacy Tunnel) outer IP header decapsulation procedure (tunnel mode). For details on cryptographic processing for both Tunnel and Transport modes, see IPsec ESP Cryptographic Decapsulation.

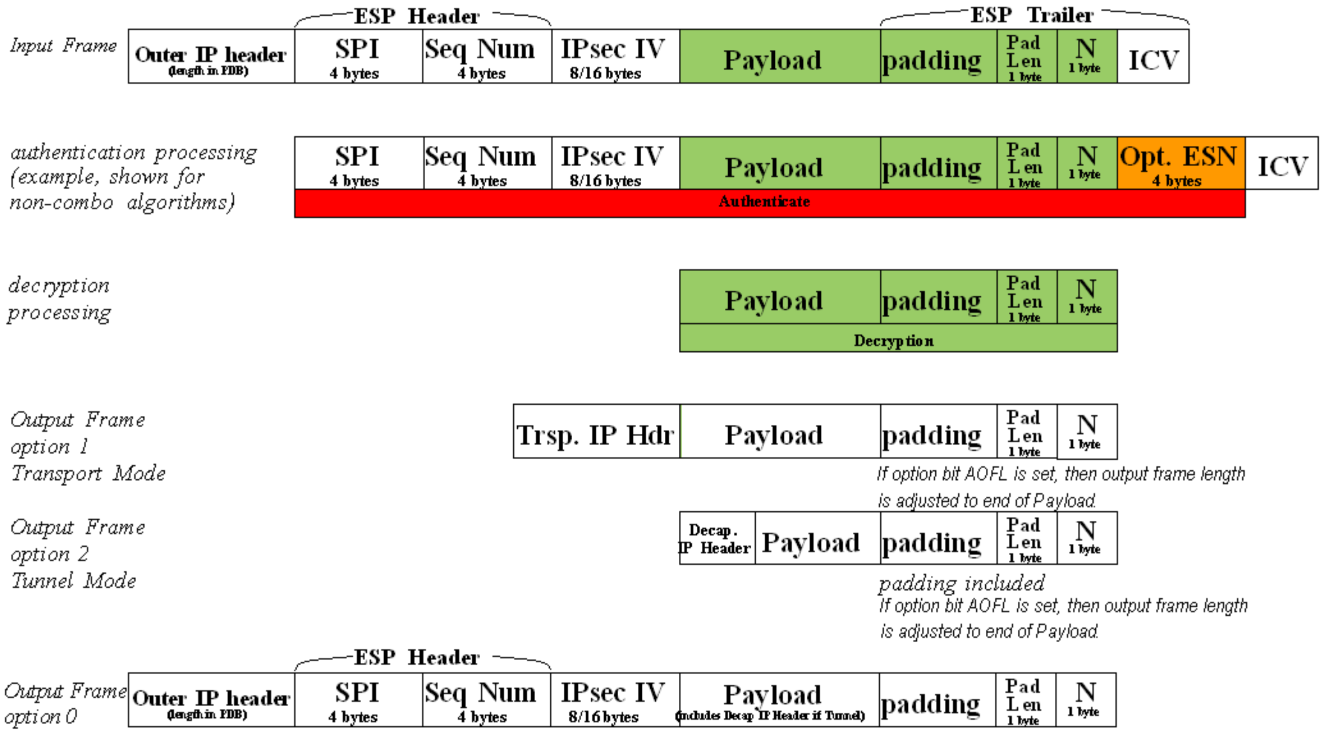

Input Frame
ESP Header
ESP Trailer
Outer IP header
(length in PDB)
SPI
4 bytes
Seq Num
4 bytes
IPsec IV
8/16 bytes
Payload
padding
Pad Len
1 byte
N
1 byte
ICV
authentication processing (example, shown for non-combo algorithms)
Opt. ESN
4 bytes
Authenticate
decryption processing
Decryption
Output Frame option 1 Transport Mode
Trsp. IP Hdr
If option bit AOFL is set, then output frame length is adjusted to end of Payload
Output Frame option 2 Tunnel Mode
Decap. IP Header
padding included
If option bit AOFL is set, then output frame length is adjusted to end of Payload
Output Frame option 0
(includes Decap IP Header if Tunnel)

**Figure 9-10. IPsec ESP Transport (and Legacy Tunnel) decapsulation procedure for both transport and tunnel modes**

SEC does the following to decapsulate an IPsec ESP datagram:

1. Receives an input frame as illustrated in Figure 9-10.
2. Processes the input frame (minus the outer IP header) for authentication. Each mode handles the outer IP header differently; see their individual sections for details.
3. Optionally updates the checksum of the valid IP header being written to the output frame, if other changes are being made to the IP header. Note the validity of the updated checksum depends upon the original checksum being valid to the IP header before any changes were made.
4. Decrypts the payload, padding, pad length field (Pad Len), and next header (N) field. Note that these fields are marked with green shading in Figure 9-10. For details on cryptographic processing for both Tunnel and Transport modes, see IPsec ESP Cryptographic Decapsulation.
5. Outputs the resulting output frame based on the selected mode and options.

Note that SEC does not have enough available buffering to decide how much padding to drop for all possible values of the Pad Len byte at the end of the frame. As a result, the output frame in options 1 and 2 includes the padding, the pad length byte, and the N byte.

If the AOFL Options bit is set in the PDB for output options 1 and 2, SEC uses the post-decryption Pad Length byte to compute the effective frame length after the fact. It adjusts the output frame length to tell consumers of the output frame to ignore padding, pad length, and N. These fields are written to the output frame, so the provided output frame must be long enough to receive these fields.

Selecting option 0 for the output frame format (see Figure 9-10) causes SEC to decrypt the payload, padding, pad length and next header fields, but leave the Outer IP header, ESP header and IPSEC IV as is.

### 9.1.6.1 IPsec ESP Transport Mode outer IP header decapsulation procedure

In transport mode, SEC processes the outer IP header by doing the following:

1. Finds the PDB field "IP Header Length"; this header length, in bytes, includes the length of all option fields and extension headers.
2. Recomputes the length field by subtracting the length of the ESP Header, the IPsec IV, and the complete ESP trailer, which consists of the ICV, the padding, the 1-byte pad length (Pad Len) field, and the 1-byte next header (N) field.
3. Replaces the byte at the location in the outer IP header indicated by the PDB field "NH Offset" with the decrypted Next Header value from the ESP Trailer (shown as N in Figure 9-10).
4. Passes the modified outer IP header from the input frame to the output frame, where it becomes the first field.

Note that SEC only recomputes the checksum field in the transport mode IP header if the Cksm bit in the PDB Options Byte is set to 1 (see Table 9-4).

Figure 9-10 shows the output frame resulting from decapsulation in transport mode as option 1.

### 9.1.6.2 IPsec ESP Transport (and Legacy Tunnel) outer IP header decapsulation procedure (tunnel mode)

In tunnel mode, SEC processing does not modify the outer unencapsulated IP header. The decapsulated IP header is included in the output frame, and normally the outer IP header is discarded. The format of the output frame is controlled by PDB Options bit outFMT.

- If outFMT=0, Output Frame option 0 as shown in Figure 9-10 is selected. Unencrypted fields in the input frame are copied as-is to the output frame, and after decryption, all other fields are copied to the output frame, with the exception of the

ICV. It is then the responsibility of another processing element to discard the outer IP header, the ESP header, and the ESP trailer.

- If outFMT=1, Output Frame option 1 as shown in Figure 9-10 is selected. The outer IP header and ESP header are used as required for decapsulation processing but are not written to the output frame. When outFMT=1, PDB Options bit AOFL (Adjust Output Frame Length) comes into play in determining the contents of the output frame:
    - If AOFL= 0, then the entire decrypted ESP trailer is written to the output frame.
    - If AOFL= 1, then SEC adjusts the output frame length after decryption has completed. SEC cannot know where payload ends until the N byte is decrypted, and goes to no special effort to decrypt it early. As a result, padding may be written to the output frame. However, once the end of payload is known, SEC adjusts the output frame length to reflect the end of proper payload, and rewinds the output frame pointer to that point.

Because the original (inner) IP header was encrypted and authenticated during the encapsulation procedure, it is uncovered during decapsulation.

Other PDB Options bits control how the inner IP header is presented:

- Setting DSC will cause SEC to copy the IPv4 TOS byte or the IPv6 Traffic Class byte from the outer IP header to the inner IP header before writing the inner header to the output frame.
- Setting DTTL will cause SEC to decrement the IPv4 TTL or IPv6 hop limit byte before writing the inner IP header to the output frame.
- Setting ODF will cause SEC to overwrite the IPv4 DF bit in the inner IP header with the value in DFV.
- Setting Cksm causes the IPv4 header checksum to be updated as a result of any changes made to the inner IPv4 header.

### 9.1.7 IPsec ESP Cryptographic Decapsulation

The IPsec ESP cryptographic processes associated with decapsulation are common for both ESP Tunnel and Transport modes, but vary depending upon cipher suite. In all cases, the ICV is computed to cover both the ESP Header and the encrypted payload and ESP Trailer. Payload and ESP Trailer are decrypted as well.

### 9.1.7.1 IPsec decapsulation procedure when using AES-CBC or DES-CBC

This figure shows IPsec ESP Transport (and Legacy Tunnel) authentification for decapsulation when the cipher is AES-CBC or DES-CBC.

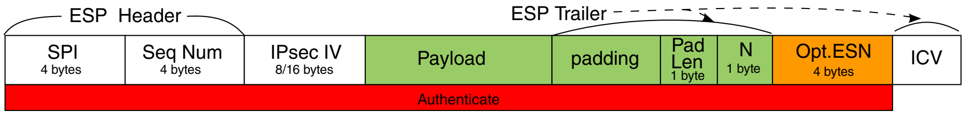


**Figure 9-11. IPsec ESP Transport (and Legacy Tunnel) authentication for decapsulation when cipher is AES-CBC or DES-CBC**

When using the AES-CBC or DES-CBC cipher suite, SEC performs IPsec decapsulation by doing the following:

1. Receives the packet from the Input Frame serially, first receiving the outer IP header, then the ESP header, which consists of the SPI and the sequence number, and finally the IPsec IV.
2. Pushes the SPI and sequence number into a CHA, typically a Class 2 CHA, for authentication.
3. Pushes the IPsec IV into a CHA for authentication and then copies it to the Class 1 CHA Context Register (register offset 0).
4. Also gives the ICV to the authentication CHA, which compares the received ICV to the value computed by authenticating all data as described above in red. If the comparison fails, then an ICV CHECK FAIL is signalled in the Job Completion Status Word.
5. Receives payload data (see Figure 9-12) from the input frame and pushes it into the input-data FIFO.

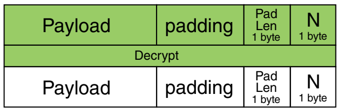


**Figure 9-12. IPsec ESP Transport (and Legacy Tunnel) AES-CBC/3DES-CBC decryption for decapsulation**

6. Tags the payload data for authentication and decryption.
7. If the decapsulated output frame option 0 is selected, pushes the results of payload decryption to the output frame along with the SPI, sequence number, and IPsec IV. If the decapsulated output frame option 1 is selected, only the decrypted payload is pushed to the output frame.

### 9.1.7.2 Process for IPsec decapsulation when using AES-CTR

This figure shows the IPsec ESP Transport (and Legacy Tunnel) authentication for decapsulation when the cipher is AES-CTR.

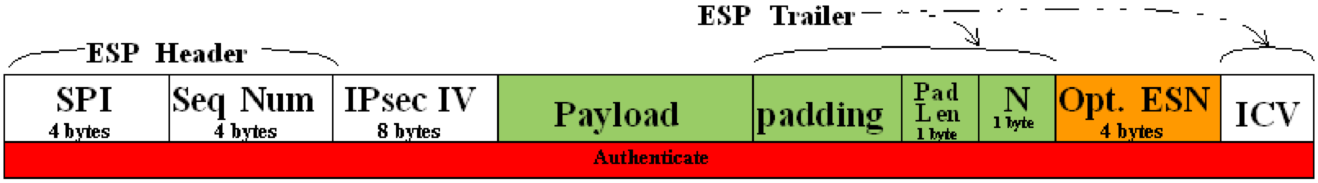


**Figure 9-13. IPsec ESP Transport (and Legacy Tunnel) authentication for decapsulation when cipher is AES-CTR**

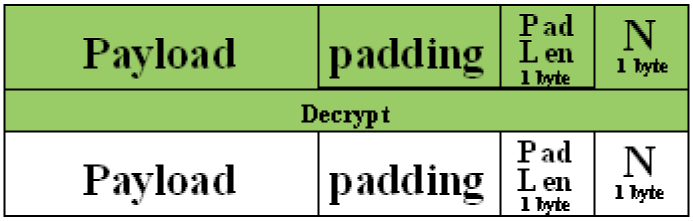


**Figure 9-14. IPsec ESP Transport (and Legacy Tunnel) AES-CTR decryption for decapsulation**

When using the AES-CTR cipher suite, SEC performs IPsec decapsulation occurs by doing the following:

1. Receives the packet from the input frame serially, first the outer IP header, then the ESP header, which consists of the SPI and the sequence number, and finally the IPsec IV.
2. Pushes the SPI and sequence number into a CHA, typically a Class 2 CHA, for authentication.
3. Constructs the counter value by prepending the 8-byte IPsec IV with a 4-byte Nonce and appending IPsec IV with a 4-byte initial count value (typically 0000 0001h); both the Nonce and initial count values are extracted as is from the PDB.
4. Writes the constructed counter value to the Class 1 Context Register, offset 16.

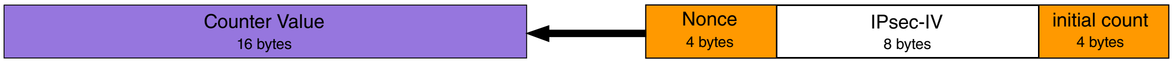


**Figure 9-15. Initial counter construction**

5. Also gives the ICV to the authentication CHA, which compares the received ICV to the value computed by authenticating all data as described above. If the comparison fails, then an ICV CHECK FAIL is signalled in the Job Completion Status Word.

6. Receives payload data (see Figure 9-12) from the input frame and pushes it into the input-data FIFO
7. Tags the payload data for authentication and decryption
8. If the decapsulated output frame option 0 is selected, pushes the payload decryption to the output frame along with the SPI, sequence number, and IPsec IV. If the decapsulated output frame option 1 is selected, only the decrypted payload is pushed to the output frame.

### 9.1.7.3 Process for IPsec decapsulation when using AES-CCM

This figure shows the IPsec ESP Transport (and Legacy Tunnel) decryption and authentication for decapsulation when the cipher is AES-CCM.

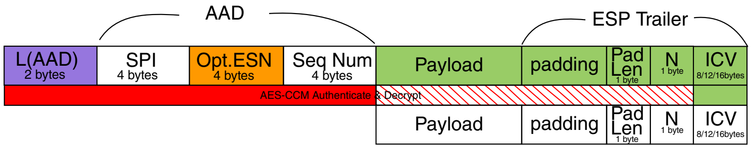


**Figure 9-16. IPsec ESP Transport (and Legacy Tunnel) decryption and authentication for decapsulation with AES-CCM**

When using the AES-CCM cipher suite, SEC performs IPsec decapsulation by doing the following. Note that this cipher suite can receive packets out of order:

1. Computes the length of AAD, and creates the Formatted AAD from a 2-byte representation of the length concatenated to the AAD, which consists of SPI, and optional ESN, and the sequence number
2. Writes this portion of the input frame to the input-data FIFO
3. Optional: If, the ESN is included, determines the correct ESN value and inserts it between SPI and Seq Num as the input frame is inserted to the input-data FIFO.

**NOTE**

Because packets may be received out of order and because the ESN increments upon Seq Num rollover, SEC may have to adjust the ESN. SEC does not increment the ESN value until the entire anti-replay window reflects post-rollover Seq Nums.

4. Contructs an 11-byte nonce value by first extracting the 8-byte IPsec IV from the input frame and then prepending a 3-byte Salt value obtained from the PDB (see the following figure).

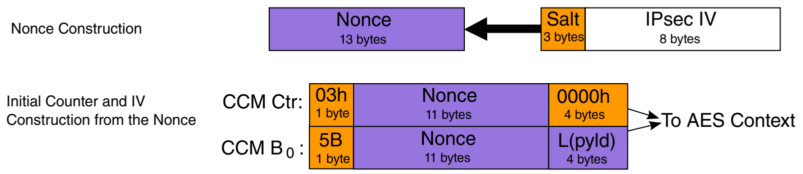


**Figure 9-17. IPsec AES-CCM context construction**

5. Constructs the CCM $B_0$ value by prepending the $B_0$ flags, which are a single-byte constant found in the PDB, and postpending the 4-byte SEC-computed payload length to create a 16-byte IV (see Figure 9-17).
6. Writes the 16-byte IV to the Class 1 Context Register (offset 0).

**NOTE**

Software must properly provision the PDB with the $B_0$ flags, per RFC 3610. Typical values matching the Nonce configuration (as specified in RFC 4309) are as follows:

- For an 8-byte ICV, use 5b.
- For a 12-byte ICV, use 6b.
- For a 16-byte ICV, use 7b.

7. Constructs the initial count value by prepending the counter field flags (another single-byte constant extracted from the PDB, shown as $CTR_0$) to the nonce and appending a four-byte value, also extracted from the PDB (typically 4 bytes of zeros) (see Figure 9-17).
8. Writes the resulting 16-byte initial count to the Class 1 Context Register immediately following the IV (that is, at offset 16).

**NOTE**

Software must properly provision the PDB with the counter field flags, per RFC 3610. The expected value matching the nonce configuration, as specified in RFC 4309, is 03h.

9. Also gives the ICV to the authentication CHA, which compares the received ICV to the value computed by authenticating all data as described above. If the comparison fails, then an ICV CHECK FAIL is signaled in the Job Completion Status Word.
10. Receives payload data from the input frame and pushes it into the input-data FIFO
11. Tags the payload data for authentication and decryption
12. If the decapsulated output frame option 0 is selected, pushes the results of payload decryption to the output frame along with the SPI, sequence number, and IPsec IV. If the decapsulated output frame option 1 is selected, only the decrypted payload is pushed to the output frame.

#### 9.1.7.4 Process for IPsec decapsulation when using AES-GCM

These figures show IPsec decapsulation when using AES-GCM.

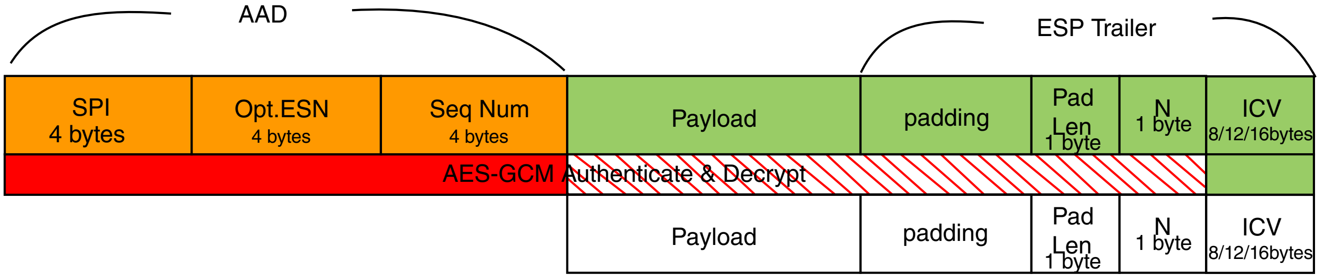


**Figure 9-18. IPsec ESP Transport (and Legacy Tunnel) decryption and authentication for decapsulation with AES-GCM**

When using the AES-GCM cipher suite, SEC performs IPsec decapsulation by doing the following:

1. Computes the GCM-IV by extracting the 4-byte Salt value from the PDB and appending to that the 8-byte IPsec IV received in the input frame (see the following figure).

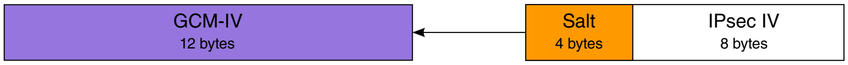


**Figure 9-19. IPsec GCM IV building**

2. Pads this GCM-IV with 4-bytes of zeros and writes the resulting 16-byte value to the input-data FIFO.
3. Pushes the AAD (the 4-byte SPI, an optional 4-byte ESN, and the 4-byte Seq Num) onto the input-data FIFO, zero padded to a block (16-byte) boundary if necessary (see Figure 9-18).

**NOTE**

Because packets may be received out-of-order and because the ESN increments upon Seq Num rollover, SEC may have to adjust the ESN. SEC does not increment the ESN value until the entire anti-replay window reflects post-rollover Seq Nums.

4. Pushes the encrypted payload and the ESP Trailer (padding, Pad Len, N and ICV) onto the input-data FIFO for decryption and authentication.

### 9.1.7.5 Use of SPI and the sequence number in decapsulation

When decapsulating, SEC ignores the SPI except when performing authentication.

SEC uses the sequence number for anti-replay checking (see Anti-replay checking in IPsec ESP decapsulation). Using fields in the PDB, SEC maintains a replay window of up to 128 packets.

- Any packet received and observed to be a duplicate of a previously received packet (within the window) has REPLAY indicated in the Job Completion Status Word.
- Any packet received and observed to fall prior to a value permitted by the replay window is tagged as LATE in the Job Completion Status Word.

### 9.1.7.6 Optional use of ESN in ESP decapsulation

After the encrypted data, the packet may infer an extended sequence number (ESN). If so configured:

1. The ESN is copied from the PDB, with a possible modification (see below).
2. The ESN is pushed into the authentication CHA, which is typically the Class 2 CHA.
3. Upon rollover of the sequence number, the packet encapsulator increments the ESN.

SEC needs to detect which sequence numbers correspond to the pre-incremented ESN and which correspond to the post-increment ESN. SEC adjusts the ESN as necessary when authenticating the packet. Once the entire anti-replay window reflects the rollover of the sequence number, SEC increments the ESN values stored in the PDB.

### 9.1.7.7 Anti-replay checking in IPsec ESP decapsulation

The IPsec decapsulation protocol uses SEC also performs anti-replay checking by doing the following:

1. Confirms the received ICV against the computed ICV
2. Stores a Seq Num value in the PDB; this value indicates the newest packet still within the window
3. Stores a bit array that supports an anti-replay window of up to 128 packets; this array indicates which packets have been received, with the least significant bit representing the packet with the aforementioned Seq Num, and which packets have not been received.

The ARS bits in the PDB Options field controls the size of the anti-replay window, as follows:

- If 01b is programmed into ARS, then a 32-bit window is selected and SEC only uses word 6 of the PDB for implementing the anti-replay scorecard.
- When ARS contains 11b, then an anti-replay window of 64 is selected, and the LS bit of word 7 represents the PDU immediately older than that represented by the MS bit of word 6.
- When an anti-replay window of 128 is selected by programming ARS to 10b, then the LS bit of word 8 represents the PDU immediately older than that represented by the MS bit of word 7 and the LS bit of word 9 represents the PDU immediately older than the MS bit of word 8.
- For any size anti-replay window, the LS bit of word 6 of the entire scorecard is used to indicate the status of the PDU represented by the values of ESN and Seq Num stored in the PDB, and each bit to the left represents a packet earlier in time.

In this version of SEC, the IPsec protocol actually uses the Anti-replay built-in protocol described in Anti-Replay built-in checking.

#### 9.1.7.7.1 When anti-replay checking is enabled

If anti-replay checking is enabled, the anti-replay scorecard (ARS) is updated with each PDU that has passed its ICV check.

If the current PDU represents a more recent packet than any previously received, then the ESN/Seq Num fields in the PDB are updated to match the received PDU, and the anti-replay scorecard is shifted left so that the LS bit of word 6 represents the current packet. For 64-bit windows, any bits shifted left out of PDB word 6 are shifted into the right end of PDB word 7. For 128-bit windows, words 8 and 9 are handled in a similar fashion to word 7.

If the current PDU is older than at least one PDU received before (as represented by the input state of the PDB ESN/Seq Num fields), then the scorecard is not shifted, but the bit representing the received packet is set to 1.

In either case, if that bit was already set to 1, then the received packet is considered a REPLAY packet, and the return status for the frame indicates REPLAY. If the bit representing the PDU has fallen off the end of the anti-replay scorecard (that is, if the difference between the received packet's Seq Num is greater than the anti-replay size), then the received packet is considered LATE, and the return status for the frame indicates LATE.

#### 9.1.7.7.2 When anti-replay checking is disabled

If anti-replay checking is disabled, the PDB is always updated to match the ESN or Seq Num values for the last PDU received.

#### 9.1.7.8 ICV checking during IPsec ESP decapsulation

The last segment (normally 12 bytes) of the input frame is the Integrity Check Value, or ICV. Once computed, the CHA responsible for computing the ICV on the received decapsulated packet will compare the computed ICV to the received ICV. If PDB Options indicates presence of ESN (Extended Sequence Number), the ESN value is taken out of the PDB and is used as part of ICV computation. For cryptographic modes CBC and CTR, the ICV is computed as if the ESN were received between the ESP Trailer's Next Header byte and the ICV. For cryptographic modes CCM and GCM, the ICV is computed as if the ESN were received between the SPI and the Sequence Number. If the computed ICV and the Received ICV do not match, then an ICV ERROR is signalled, and processing halts.

**NOTE**

> The ICV is checked before sequencing and padding is checked. That is, an ICV ERROR takes precedence and will mask LATE, REPLAY, and BAD ESP PADDING errors. Only after the ICV compares favorably is the sequencing and the padding checked.

### 9.1.8 IPsec ESP Tunnel encapsulation overview

The IPsec ESP Tunnel Protocol-thread, when selected for encapsulation, encapsulates the contents of the Input Frame, using the encryption and authentication functions selected in the Protocol Operation command PROTINFO field. The Input Frame must not be so long such that when an Outer IP Header is added, a Jumbo Datagram is constructed. In other words, the maximum length after encapsulation must not exceed 65535 bytes (including an outer IPv4 header but not including a 40 byte outer IPv6 header).

Note that the IPsec ESP Transport (and legacy tunnel) Protocol thread can also be used for IPsec tunnel-mode encapsulation. Some of the details differ; in particular the IPsec ESP Transport (and legacy tunnel) Protocol thread cannot be used in conjunction with UDP-encapsulated-ESP, and the use of the Outer IP Header is more greatly constrained. Both threads perform cryptographic encapsulation identically; for further details, see IPsec ESP Cryptographic Encapsulation.

The details of how the IPsec ESP Tunnel encapsulation Protocol thread handles the outer IP header are described below.

#### 9.1.8.1 Handling the Outer IP Header during ESP Tunnel encapsulation

During ESP Tunnel mode encapsulation, an outer IP header is applied to the new datagram, and the IP header of the datagram being encapsulated is encrypted. SEC provides several options for providing this outer IP header, as controlled by the PDB Options field OIHI and DPOVRD bit OIMIF:

- OIHI = 11b - Outer IP Header Material is copied from the PDB.
- OIHI = 10b - Outer IP Header Material is copied from memory -- from the location addressed by the PDB
- OIHI = 01b - Outer IP Header Material is copied from input frame.
- OIHI = 00b - No Outer IP Header is applied
- OIMIF = 1b overrides the OIHI-specified source for the Outer IP Header Material, instead using the input frame. Whereas the scope of OIHI is for every frame in the flow, the scope of OIMIF is to the particular frame to which OIMIF applies.

The description of ESP Tunnel mode encapsulation refers to "Outer IP Header Material" becuase specific provision is made for the material to contain more than just the Outer IP Header. PDB / DPOVRD field AOIPHO specifies a number of additional bytes that preceed the actual Outer IP Header. This additional material could consist of additional outer headers, such as an Ethernet header. The additional material is copied to the output frame as is. The Outer IP Header, on the other hand, is subject to additional processing.

The PDB is designed such that if OIHI selects Outer IP header material from PDB, that the Outer IP header Material starts offset by some multiple of 8 bytes within the descriptor. If using AOIPHO, then both the Outer IP header Material and the actual Outer IP header must start offset on an 8-byte multiple within the descriptor. In such a circumstance, AOIPHO reflects the number of bytes to be used that is part of the overall material but is not part of the actual Outer IP Header. For example, if AOIPHO is used to provide for a 14B Ethernet header header (two mac addresses and ethertype) in addition to a standard 20B IPv4, then AOIPHO is programmed to 14, and Outer IP Header Material length is programmed to 34. However, in constructing the PDB, there must be two extra bytes of padding between the Ethernet header and the actual Outer IP header that SEC will skip. The Outer IP Header Material will then occupy a total of 36 bytes in the PDB, including the 2 bytes of padding.

**NOTE**

> The 8-byte alignment rule only applies when OIHI selects Outer IP header material from the PDB. If OIHI selects outer IP header material from external memory or from the input frame, then no padding material is required to align the actual Outer IP header.

### 9.1.8.2 Outer IP Header handling with UDP-encapsulated-ESP

The ESP Tunnel encapsulation thread is capable of handling RFC 3948 UDP-encapsulated-ESP. If PDB Options bit NAT is set, then the ESP Tunnel encapsulation procedure includes special handling as follows:

- The last 4 bytes of the Outer IP Header Material are treated as the Source and Destination Port fields of the UDP header
- SEC inserts two bytes after the Destination Port field for the UDP Packet Length. The length written in this field to the output frame includes the 8-byte UDP header, the ESP header, the encrypted datagram, the ESP trailer, and the ICV.
- SEC inserts two bytes after the UDP Packet Length field for the UDP Packet Checksum.
  - If PDB Options bit NUC = 0, then no proper UDP Packet Checksum is computed, and the field is left as two bytes of zeros.
  - If PDB Options bit NUC = 1, then a proper UDP Packet Checksum is computed across all the bytes written to the output frame as accounted for by the UDP Packet Length field, *plus* an IP Pseudo-Header covering the appropriate fields of the Outer IP Header. Note that these fields differ, depending upon whether the Outer IP Header is IPv4 or IPv6.

**NOTE**

DECO includes hardware for computing a 16-bit one's complement checksum. Normally, the use of the hardware is controlled by SEQ FIFO STORE commands; that is SEQ FIFO STORE Source Field values can be used to determine which bytes written to the output frame are included in a checksum computation. However, for IPsec ESP Tunnel encapsulation, if NAT and NUC are selected in the PDB Options byte, then those controls are overridden, and the checksum hardware is used for computing the UDP checksum.

### 9.1.8.3 ESP Tunnel Outer IP Header manipulation

The ESP Transport (and legacy tunnel) PDB Options byte contains several bits that control how that thread might handle the Outer IP Header. These bits include:

- Cksm: if enabled, and if IPv4 is selected, update the outer header checksum per any manipulations made to the outer IP header
- DSC: if enabled, copy the IPv4 TOS or IPv6 Traffic Class byte from the inner IP header to the outer IP header
- IPvsn, selecting the version of IP header handled

For the ESP Tunnel thread, these are not options. The version of IP header is known based on the Version field of the inner and outer IP headers, and these need not be the same. As long as an outer IP header is provided, SEC*will* copy the TOS or Traffic Class byte from the inner header to the outer header, and if the outer header is IPv4, SEC*will* update the checksum resulting from this and any other changes to the header.

In addition to manipulations described above, SEC will decrement the outer IP header Time-to-Live field (for IPv4) and the Hop Limit field (for IPv6). SEC will also compute the appropriate length field value for the Outer IP Header.

The PDB HMO field for both the ESP Transport (and legacy tunnel) and ESP Tunnel encapsulation threads is identical. Copying the DF bit from inner IPv4 header to outer IPv4 header is optional, as is decrementing the Time-to-Live (IPv4) / Hop Limit (IPv6) field.

### 9.1.8.4 ESP Tunnel handling of Next Header

The last byte encrypted during encapsulation is the Next Header byte. The unencrypted value is designed to indicate the type of payload that has been encapsulated. The value SEC uses during encapsulation comes either from the Next Header field of the PDB, or from Next Header field of the DPOVRD register, if the most significant bit of DPOVRD is set.

## 9.1.9 IPsec ESP tunnel decapsulation overview

The IPsec ESP Tunnel Protocol-thread, when selected for decapsulation, decapsulates the contents of the Input Frame, using the decryption and authentication functions selected in the Protocol Operation command PROTINFO field. The input frame must contain an ESP header, an encrypted payload with appropriate padding, and an ICV, and in total must not exceed 65535 bytes in length (including an outer IPv4 header but not including a 40 byte outer IPv6 header). The Input Frame may also contain Outer IP Header Material at the front of the Input Frame. The primary purpose for inclusion of an Outer IP Header as an input to the decapsulation process is to allow the Protocol-thread to copy selected fields from the Outer IP Header to the decapsulated Inner IP Header. The Descriptor Protocol Data Block (PDB) contains fields and control bits to specify the precise processing performed. This is described in sections below.

Note that the IPsec ESP Transport (and legacy tunnel) Protocol thread can also be used for IPsec tunnel-mode decapsulation. Some details differ; in particular the IPsec ESP Transport (and legacy tunnel) Protocol thread cannot be used in conjunction with UDP-encapsulated-ESP, and the use of the Outer IP Header is more greatly constrained. Both threads perform cryptographic decapsulation identically; for further details, see IPsec ESP Cryptographic Decapsulation.

After decryption of the ESP trailer has completed, SEC performs a check of the cryptographic padding field to ensure it conforms to requirements and was not corrupted during transmission or decapsulation. The detection of corrupted Cryptographic padding will result in the signalling of a BAD ESP PADDING error, and will halt processing.

### 9.1.9.1 Input material preceding the outer IP header

Unlike the IPsec ESP Transport (and legacy tunnel) thread, the IPsec ESP Tunnel thread is designed to accept lower level headers or other material that may precede the actual Outer IP Header. PDB / DPOVRD field AOIPHO defines the number of bytes that precede the actual Outer IP Header. The PDB value is used normally; if DPOVRD is used to override the PDB, the the AOIPHO field in DPOVRD is used instead.

Normally, any additional material preceding the actual Outer IP Header is not included in the output frame, like the Outer IP Header. PDB Options bit ETU enables copying of the preceding material, under the assumption that it represents an Ethernet header.

Because the last two bytes of an Ethernet header are the EtherType field, SEC does not copy the last two bytes of the preceding material (as defined by AOIPHO), but instead replaces them with a proper EtherType value, depending on the version of the decapsulated Inner IP Header: if IPv4, then 0x0800 is put into the Output Frame immediately prior to the Inner IP Header. If IPv6, then 0x86DD is put into the Output Frame.

### 9.1.9.2 Handling the Outer IP Header during ESP Tunnel decapsulation

That Outer IP Header Material may consist of some segment of data prior to the actual Outer IP Header, the Outer IP Header, and a UDP header suitable for NAT. The resulting Output Frame will consist of the decapsulated payload, with extra material (including cryptographic padding) removed. Note that the Decapsulation thread is not designed to detect and remove TFC padding. In some circumstances, Input Frame material outside the decapsulated payload that was prior to the actual Outer IP Header in the Input Frame may be copied with adjustment to the Output Frame.

The primary purpose for inclusion of an Outer IP Header as an input to the decapsulation process is to allow the Protocol-thread to copy selected fields from the Outer IP Header to the decapsulated Inner IP Header. The Descriptor Protocol Data Block (PDB) contains fields and control bits to specify the precise processing performed. This is described below.

### 9.1.9.3 Manipulation of the Inner IP Header during ESP Tunnel decapsulation

The PDB Options field for ESP Tunnel decapsulation contains several bits to control how the Inner IP Header is manipulated after decryption.

In an IPv6 Header, DS and ECN are carried in 8 bits labelled Traffic Class, and straddle the lower half of the first byte, and the upper half of the second byte. An IPv4 header uses DS and ECN in the Type of Service (TOS) field, found in the second byte of an IPv4 Header. (The upper 6 bits is DS, and the lower 2 bits is ECN.) If the PDB options select DSC but not TECN, then the entire TOS / TC byte is copied from the Outer IP Header to the decapsulated Inner IP Header.

The ESP Tunnel decapsulation thread is designed to handle ECN Tunnelling upon IPsec decapsulation as defined in RFC 6040. If Options bit TECN is set, then SEC updates the inner IP header ECN bits as based on their current value, plus the value of the ECN bits in the outer IP header. If TECN is selected without DSC, the Protocol thread will update the ECN portion of the TOS / TC field in the Inner IP Header but not the DS field. Details of ECN tunnelling are shown in the table below. Note that for the case of CE DROP error assertion, the output frame is generated correctly, but for this version of SEC, LATE and REPLAY checking is skipped.

Table 9-38. RFC 6040 ECN tunneling

| receive inner ECN | received outer ECN | update inner ECN with | additional action |
|---|---|---|---|
| 00 | 00 / 01 / 10 | 00 | |
| 00 | 11 | 11 | Return complete frame with new CE DROP error asserted |
| 01 | 00 / 01 | 01 | |
| 01 | 10 | 10 | |
| 01 | 11 | 11 | |
| 10 | 00 / 01 / 10 | 10 | |
| 10 | 11 | 11 | |
| 11 | all | 11 | |

If the Inner IP Header is IPv4, then the PDB ODF Option applies as follows: the DF bit in the Inner IP Header is replaced with the value of the DFV PDB Options bit before the Inner IP header is written to the Output Frame.

Upon decapsulation, the IPv4 inner Header TTL field is decremented (by one) before the decapsulated Inner Header is written to the Output Frame. If an IPv6 Header instead, then the decrement by one is applied to the Hop Limit field.

All changes made to the decapsulated Inner IPv4 Header result in a update of the Header Checksum. Note that if the Header Checksum was incorrect to begin with, the resulting Checksum will also be incorrect.

#### 9.1.9.4 Decapsulation Output Frame Length

The length of the Output Frame depends upon the version of the Inner IP Header as follows:

- For IPv4, the Inner IP Header contains a Total Length field, and that field is used to specify the length of the Output Frame. As a result, only the intended encapsulated payload is written to the Output Frame; the remainder can be reliably prevented from being written to memory.
- For IPv6, the Inner IP Header contains a Payload Length that indicates the length of the encapsulated payload (including the Extension Headers), but not the length of the encapsulated base IPv6 header (which is always 40 bytes in length).

## 9.2 SSL/TLS/DTLS record encapsulation and decapsulation overview

SEC supports the following versions of the TLS family of security protocols:

- SSL 3.0
- TLS 1.0
- TLS 1.1
- TLS 1.2
- DTLS 1.0 (a variant of TLS 1.1)
- DTLS 1.2 (a variant of TLS 1.2)

The variants of the protocol are similar, but have the following key differences:

- Handling of IVs for block ciphers has evolved over the versions
- The list of supported cipher suites has evolved over the variants.

- DTLS 1.0 is a variant of TLS 1.1 with an explicit sequence number.
- TLS 1.2 and DTLS 1.2 replace the MD5/SHA-1-based pseudo-random function (PRF) with a PRF computed using only SHA-256 or SHA-384

The PROTINFO field codes enumerated in Table 7-55 define the cipher suites used by the protocol, and SEC's built-in protocol processing sequences handles the remaining details. Detailed processing descriptions must be described differently for different versions. PDB and PDB Override programming, decapsulation output frame options, and finding the last byte of the encrypted payload during decapsulation processing are all common to the different protocol versions, and are described first.

**Table 9-39. SSL/TLS/DTLS protocol descriptors**

| Encapsulation | | Decapsulation |
|---|---|---|
| Header | | Header |
| Protocol data block includes next header, SPI, sequence number, IV (if not from RNG) | | Protocol data block includes sequence number, anti-replay information (DTLS only) |
| Class 2 key data block | | Class 2 key data block |
| Class 1 key data block | | Class 1 key data block |
| Protocol = <protocol> encrypt | | Protocol = <protocol> decrypt |

**NOTE**

Any bulk-data protocol using a cipher suite that includes any HMAC uses MDHA and for performance requires the use of a split key. Therefore for proper operation when using IPsec with HMAC, the KDEST field in the Class 2 KEY command must be set to MDHA Split Key. For first invocation, the Derived Key Protocol may be used to create both the split key form of the HMAC key as well as the actual key command loading the split key.

**NOTE**

Sharing MD5 SMAC and HMAC keys is restricted. SEC will prevent mis-sharing of MD5 Keys if shared descriptor SHARE is set to NEVER, WAIT, or SERIAL. ALWAYS should not be used. For more information on sharing, please refer to Table 7-1.

## 9.2.1 Programming and processing details common to all versions of SSL, TLS, and DTLS

Certain details of processing, and how to program the shared descriptor, are common to all supported versions of SSL, TLS, and DTLS:

- Protocol Data Block Programming formats vary by cipher suite, but are common across protocol versions, including encapsulation and decapsulation.
- Using the Datapath Override (DPOVRD) register to provide a non-default Type field for encapsulation
- Finding the last byte of payload and determining the pre-encapsulation record header
- Decapsulation Output Frame Formats

### 9.2.1.1 PDB use and format for SSL, TLS, and DTLS encapsulation and decapsulation

Unlike other protocols' PDBs, the SSL/TLS/DTLS PDB varies in content and field size based on PROTINFO and Options settings. In particular, for CBC-mode cipher suites, the IV field is only 2 words if the PROTINFO field of the Operation Command selects DES or 3DES. Also, the ICV Len field is present only if necessary per the TrICV bit in the PDB's Options field. The format of the PDB is, as much as possible, kept common between all different versions. One notable exception is the Sequence Number: Rather than using a 64-bit sequence number, DTLS uses a 16-bit epoch and a 48-bit sequence number.

SSL and all versions of TLS use identical PDBs for both encapsulation and decapsulation. DTLS PDBs for decapsulation are almost the same as all the others, with the addition of the Anti Replay Scorecard prior to the ICV Length word. The Options byte is somewhat different for decapsulation, with the addition of a field to control the size of the Anti Replay window

#### 9.2.1.1.1 PDB for SSL, TLS, and DTLS when a Block Cipher is used

Block ciphers in SSL family require an initialization vector -- an IV. The IV randomizes the payload prior to encryption. For SSL and TLS version 1.0, the IV is the final cipher block of the previous record. TLS 1.2 uses a random IV. TLS 1.1 allows the final cipher block of the previous record to be masked with a random or fixed mask. The IV field is designed to store IV state, as required, between a previous record and a next record. The Anti-replay Scorecard fields are used only for DTLS Decap, and only as many words as required to implement the window size chosen by the Options byte ARS field.

**Table 9-40. Block cipher shared descriptor PDB for SSL, TLS, and DTLS encapsulation and decapsulation**

| | | | | |
|---|---|---|---|---|
| | Descriptor Header (1 or 2 words) | | | |
| PDB Word 0 | Type<br>(8 bits) | Version<br>(16 bits) | Options<br>(8 bits) | |
| PDB Word 1<br>*for SSL and TLS* | Seq Num 1 | | | DECO writes back to PDB as needed |
| PDB Word 1<br>*for DTLS* | Epoch | Seq Num 1 | | |
| PDB Word 2<br>*all protocols* | Seq Num 2 | | | |
| PDB Word 3<br>*either-CBC and I/E=1* | IV word 1 | | | |
| PDB Word 4<br>*either-CBC and I/E=1* | IV word 2 | | | |
| PDB Word 5<br>*AES-CBC and I/E=1* | IV word 3 | | | |
| PDB Word 6<br>*AES-CBC and I/E=1* | IV word 4 | | | |
| PDB Word 7 or 5 or 3 | Anti-Replay Scorecard word 1 | | | DECO writes back to PDB as needed<br>***for DTLS Decap only***<br>(First PDB word identifying number used when AES-CBC and I/E=1)<br>(Second PDB word identifying number used when DES-CBC and I/E=1)<br>(Third PDB word identifying number used when I/E=0) |
| PDB Word 8 or 6 or 4 | Anti-Replay Scorecard word 2 | | | |
| PDB Word 9 or 7 or 5 | Anti-Replay Scorecard word 3<br>(when anti-replay window is 128 ) | | | |
| PDB Word 10 or 8 or 6 | Anti-Replay Scorecard word 4<br>(when anti-replay window is 128 ) | | | |
| Last PDB Word | ICV Len<br>(8 bits) | Reserved<br>(24 bits) | | |

#### 9.2.1.1.2 PDB for SSL, TLS, and DTLS when AES-Counter mode is used

The TLS implementation of AES-Counter is based on a draft RFC that was permitted to expire. The PDB requires a 48-bit WRITE_IV for constructing the initial counter value, which per the draft is a product of the 48 "rightmost" bits of either CLIENT_WRITE_IV or SERVER_WRITE_IV -- whichever was generated by this side of the negotiation. The draft RFC for TLS with AES-counter specifies initial lower 16 bits are to be programmed as zeros, but a different can be used by programming the PDB field "Constant 0000h" otherwise. The Anti-replay Scorecard fields are used only for DTLS Decap, and only as many words as required to implement the window size chosen by the Options byte ARS field.

**Table 9-41. AES-Counter cipher shared descriptor PDB for SSL, TLS, and DTLS encapsulation and decapsulation**

| | Descriptor Header (1 or 2 words) | | | |
|---|---|---|---|---|
| PDB Word 0 | Type<br>(8 bits) | Version<br>(16 bits) | Options<br>(8 bits) | |
| PDB Word 1<br>*for SSL and TLS* | Seq Num 1 | | | DECO writes back to PDB as needed |
| PDB Word 1<br>*for DTLS* | Epoch | Seq Num 1 | | |
| PDB Word 2<br>*all protocols* | Seq Num 2 | | | |
| PDB Word 3 | WRITE_IV<br>(Upper 32 bits) | | | |
| PDB Word 4 | Write IV<br>(lower 16 bits) | Constant 0000h | | |
| PDB Word 5 | Anti-Replay Scorecard word 1 | | | DECO writes back to PDB as needed<br>*for DTLS Decap only* |
| PDB Word 6 | Anti-Replay Scorecard word 2 | | | |
| PDB Word 7 | Anti-Replay Scorecard word 3 (when anti-replay window is 128 ) | | | |
| PDB Word 8 | Anti-Replay Scorecard word 4 (when anti-replay window is 128 ) | | | |
| Last PDB Word | ICV Len<br>(8 bits) | Reserved<br>(24 bits) | | *for DTLS Decap only* |

### 9.2.1.1.3 PDB for TLS and DTLS when AES-GCM is used

AES GCM state required in the PDB that required for all cipher suites, plus a 4 byte Salt value that is essentially extra key material. Salt, and the 8-byte sequence number, are concatenated to form a 12-byte GCM IV. The Anti-replay Scorecard fields are used only for DTLS Decap, and only as many words as required to implement the window size chosen by the Options byte ARS field.

**Table 9-42. AES-GCM AEAD shared descriptor PDB for TLS 1.2 and DTLS 1.2 encapsulation and decapsulation**

| | Descriptor Header (1 or 2 words) | |
|---|---|---|

*Table continues on the next page...*

**Table 9-42. AES-GCM AEAD shared descriptor PDB for TLS 1.2 and DTLS 1.2 encapsulation and decapsulation (continued)**

| | | | | |
|---|---|---|---|---|
| PDB Word 0 | Type<br>(8 bits) | Version<br>(16 bits) | Options<br>(8 bits) | |
| PDB Word 1<br>*for SSL and TLS* | Seq Num 1 | | | DECO writes back to PDB as needed |
| PDB Word 1<br>*for DTLS* | Epoch | Seq Num 1 | | |
| PDB Word 2 | Seq Num 2 | | | |
| PDB Word 3 | Salt | | | |
| PDB Word 4 | Anti-Replay Scorecard word 1 | | | DECO writes back to PDB as needed<br>***for DTLS Decap only*** |
| PDB Word 5 | Anti-Replay Scorecard word 2 | | | |
| PDB Word 6 | Anti-Replay Scorecard word 3<br>(when anti-replay window is 128 ) | | | |
| PDB Word 7 | Anti-Replay Scorecard word 4<br>(when anti-replay window is 128 ) | | | |
| Last PDB Word | ICV Len<br>(8 bits) | Reserved<br>(24 bits) | | *for DTLS Decap only* |

#### 9.2.1.1.4 PDB for TLS and DTLS when AES-CCM is used

AES-CCM uses the most complex PDB of all. Besides Type, Version, and Sequence Number, Several constants get pulled out of the PDB in order to create $B_0$ and $CTR_0$ that must be written into Class 1 context for AESA to perform CCM mode encapsulation properly. The Anti-replay Scorecard fields are used only for DTLS Decap, and only as many words as required to implement the window size chosen by the Options byte ARS field.

**Table 9-43. AES-CCM AEAD shared descriptor PDB for TLS 1.2 and DTLS 1.2 encapsulation and decapsulation**

| | | | | |
|---|---|---|---|---|
| | Descriptor Header (1 or 2 words) | | | |
| PDB Word 0 | Type<br>(8 bits) | Version<br>(16 bits) | Options<br>(8 bits) | |
| PDB Word 1<br>*for SSL and TLS* | Seq Num 1 | | | DECO writes back to PDB as needed |
| PDB Word 1<br>*for DTLS* | Epoch | Seq Num 1 | | |
| PDB Word 2<br>*all protocols* | Seq Num 2 | | | |
| PDB Word 3 | WRITE_IV32 | | | |

*Table continues on the next page...*

Table 9-43. AES-CCM AEAD shared descriptor PDB for TLS 1.2 and DTLS 1.2 encapsulation and decapsulation (continued)

| | | | | |
|---|---|---|---|---|
| | *(write IV generated by this endpoint)* | | | |
| PDB Word 4 | $B_0$ Flags | $CTR_0$ Flags | Reserved | |
| PDB Word 5 | Reserved | $CTR_0$ lower 3 bytes 000000h | | |
| PDB Word 6 | Anti-Replay Scorecard word 1 | | | DECO writes back to PDB as needed<br>***for DTLS Decap only*** |
| PDB Word 7 | Anti-Replay Scorecard word 2 | | | |
| PDB Word 8 | Anti-Replay Scorecard word 3<br>(when anti-replay window is 128 ) | | | |
| PDB Word 9 | Anti-Replay Scorecard word 4<br>(when anti-replay window is 128 ) | | | |
| Last PDB Word | ICV Len<br>(8 bits) | Reserved<br>(24 bits) | | *for DTLS Decap only* |

### 9.2.1.1.5 Programming the Options byte with the PDB for SSL, TLS and DTLS

The encapsulation options byte contains three control bits, described below. Note that W/B and I/E are used only for CBC-based cipher suites, and are NOT for use with SSL or TLS 1.0.

Table 9-44. SSL, TLS, DTLS encapsulation PDB, format of the options byte

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | TrICV | Reserved | | w/b | e/i |

Table 9-45. SSL, TLS, and DTLS encapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| 7-5 | Reserved |
| 4<br>TrICV | Truncate ICV<br>0 Normal ICV as defined per cipher suite.<br>1 ICV length is determined by ICV Len field in PDB. |
| 3-2 | Reserved |
| 1<br>w/b | IV writeback<br>0 IV field in PDB held constant.<br>1 IV field in PDB written back with last block of ciphertext.<br>*NOTE: Block Cipher ONLY. For stream or AEAD ciphers, this bit is reserved and must be 0.*<br>*NOTE: SSL and TLS 1.0, this bit is reserved and must be 0.* |
| 0<br>e/i | e/i: Explicit/Implicit random IV.<br>0 Implicit Random IV field transmitted as part of encrypted payload. |

### Table 9-45. SSL, TLS, and DTLS encapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| | 1 Explicit Random IV field transmitted as plaintext IV.<br>*NOTE: Block Cipher ONLY. For stream or AEAD ciphers, this bit is reserved and must be 0.*<br>*NOTE: SSL and TLS 1.0, this bit is reserved and must be 0.* |

The decapsulation options byte for SSL and TLS contains five control bits, described below. DTLS decapulation options add two ARS control bits. Note that W/B and I/E are used only for CBC-based cipher suites, and are NOT for use with SSL or TLS 1.0.

### Table 9-46. SSL, TLS, DTLS decapsulation PDB, format of the options byte

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| DTLS: ARS<br>SSL, TLS: Reserved | | rsv | TrICV | outFMT | | w/b | e/i |

### Table 9-47. SSL, TLS, and DTLS decapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| 7-6<br>ARS | Anti-replay window size<br>00 - No anti-replay window<br>01 - 32 entry anti-replay window<br>10 - 128 entry anti-replay window<br>11 - 64 entry anti-replay window<br>*Note ARS is used only with DTLS, not with TLS or SSL* |
| 5 | Reserved |
| 4<br>TrICV | Truncate ICV<br>0 Normal ICV as defined per cipher suite.<br>1 ICV length is determined by ICV Len field in PDB. |
| 3-2<br>outFMT | Decapsulation Output Frame format<br>00 - Option 1: Output is payload only<br>01 - Option 3: Output frame consists of Header and Payload<br>10 - Option 2: Output frame consists of entire input frame, decrypted<br>11 - Reserved -- results in PDB error |
| 1<br>w/b | IV writeback<br>0 IV field in PDB held constant.<br>1 IV field in PDB written back with last block of ciphertext.<br>*NOTE: Block Cipher ONLY. For stream or AEAD ciphers, this bit is reserved and must be 0.*<br>*NOTE: SSL and TLS 1.0, this bit is reserved and must be 0.* |
| 0<br>e/i | e/i: Explicit/Implicit random IV.<br>0 Implicit Random IV field transmitted as part of encrypted payload. |

Table 9-47. SSL, TLS, and DTLS decapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| | 1 Explicit Random IV field transmitted as plaintext IV.<br>*NOTE: Block Cipher ONLY. For stream or AEAD ciphers, this bit is reserved and must be 0.*<br>*NOTE: SSL and TLS 1.0, this bit is reserved and must be 0.* |

### 9.2.1.2 Overriding the PDB for SSL, TLS, and DTLS Encapsulation

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For SSL, TLS, and DTLS, the header TYPE field is maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired TYPE value in the job descriptor. For more information, see Job Ring interface. When using the Queue Manager Interface, QI builds the job descriptor with a LOAD IMMEDIATE to the DPOVRD register with the value of the STATUS/CMD field in the FD. For more information, see Queue Manager Interface (QI).

Table 9-48. SSL/TLS/DTLS encapsulation-DECO Protocol Override Register format

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | OVRD | Reserved | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | Reserved | | | | | | | | Type | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 9-49. SSL/TLS/DTLS encapsulation-DECO Protocol Override Register description

| Field | Description |
|---|---|
| 31<br>OVRD | Indicates whether to use the contents of DPOVRD to override values specified in the PDB<br>0 Use the PDB as provided.<br>1 Override values in PDB by using values in DPOVRD. |
| 30-8 | Reserved |
| 7-0<br>Type | This value is used for constructing the SSL or TLS packet header (instead of the Type field in the PDB) if OVRD = 1. |

### 9.2.1.3 Computing the pre-encrypted record length during decapsulation

TLS was developed such that the record is encapsulated using the pre-encryption length in the header, but the length field of the encapsulated record includes additions such as the ICV, any nonce or IVs, and padding. Because SEC performs decryption and authentication processing simultaneously, it must pre-compute the pre-encryption length to use for authentication processing. This is particularly challenging for block ciphers, which during encapsulation adds some number of bytes of padding, that number being unknown to decryption processing until the last byte of ciphertext has been decrypted.

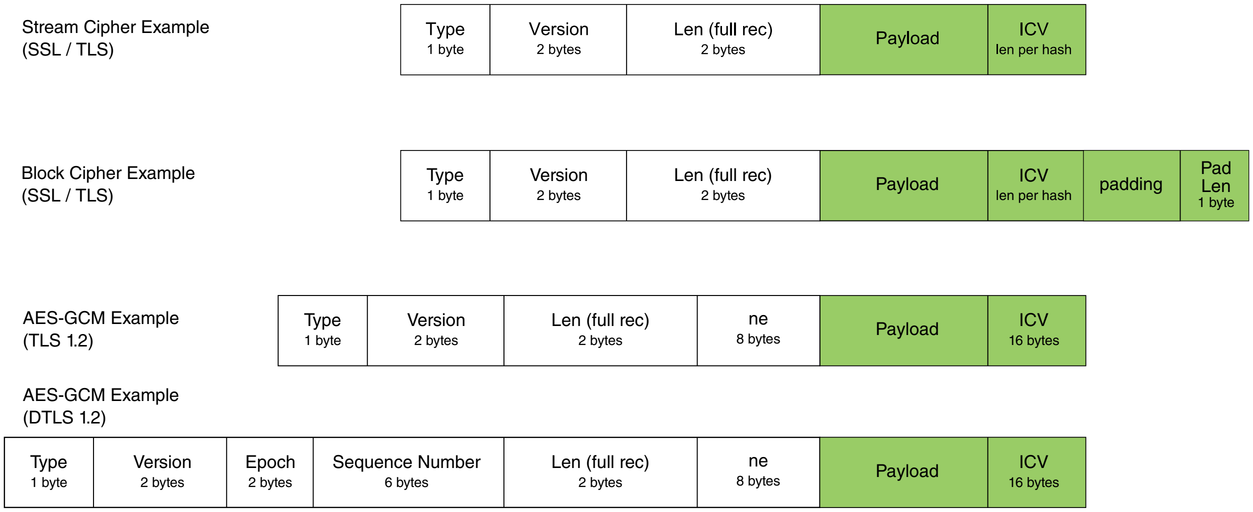


**Figure 9-20. Some examples of encapsulated records**

For stream ciphers, the required computation is relatively simple: subtract the length of the ICV. The ICV length in the case of all supported stream cipher suites turns out to be the size of the underlying hash unless TLS extension Truncated_HMAC has been negotiated, and PDB option TrICV has been set. In that case, the ICV length subtracted is the value found in the PDB ICV Len field.

Some AEAD ciphers are like stream ciphers, in that the ICV Length must be subtracted from the record length prior to beginning decapsulation. For DTLS, a second eight bytes needs to be subtracted from the record length to account for Epoch (two bytes) and Sequence Number (six bytes).

AES-GCM is an AEAD cipher that requires a slightly different encapsulated record -- it is like stream ciphers, except for the addition of ne (nonce-explicit). For this form, the length of the unprotected record is found by subtracting both the ICV length (16 bytes) and the ne length (8 bytes)

DTLS includes an explicit 8-byte Epoch / Sequence Number pair, in both the encapsulated and unencapsulated forms of the record, regardless of the cipher suite selected. These eight bytes are always included in the record length.

During encapsulation using block ciphers (CBC mode), some number of bytes of padding, plus one byte of pad length, are added to the encrypted payload to ensure the encrypted payload is an integral multiple of the underlying cipher block size. The total subtracted from the encapsulated record length is the ICV length (described above for stream ciphers), any IVs added during encapsulation, the length of padding, and 1 for the pad length byte.

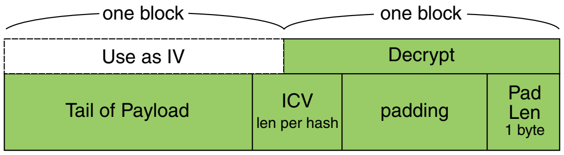


**Figure 9-21. Example of last two blocks of ciphertext**

To compute the length of padding, prior to starting decapsulation, SEC jumps to the end of the input frame, grabs the two last block of ciphertext, and performs a quick decryption of the last block, using the penultimate block as an IV. The last byte of the decrypted block *is* the padlength.

## 9.2.1.4 SSL, TLS, DTLS Decapsulation Output frame options

Programming the outFMT field of the PDB Options byte offers three options for providing the decapsulated record:

- Option 1 is the bare record encapsulated record, with everything but the payload removed.
    - For jobs submitted through QI, the payload length can be determined by the output buffer length returned as part of the frame description
    - For jobs submitted through a Job Ring, the actual record length will be returned only if INCL_SEQ_OUT is set in the JRCFGR_JRx_MS register associated with the particular Job Ring.
- Option 2 returns the entire decrypted record, with the modification that the length field reflects the length of the plaintext-payload. This option allows software

inspection of padding (which is recommended by standards but is not implemented by SEC).

- Option 3 returns the record header (length-adjusted) and payload.

Some example diagrams follow for reference. First is an example showing option 1, where the returned output frame consists of just the record payload.

**Figure 9-22. Decapsulation output frame Option 1**

Next, some examples of option 2. For option 2, all fields are found in the input frame are returned; albeit with the record length field modified to reflect the decapsulated record length, and the record payload decrypted. The simplest form is used by stream ciphers and most AEAD ciphers. AES-GCM is the exception for AEAD ciphers, including the nonce_explicit field. The block cipher forms include the padding and the pad-length byte, with either the random IV or the masked IV included for TLS 1.1 and TLS 1.2. Finally, for any permitted cipher, DTLS varies from the TLS 1.2 form by inclusion of the Epoch and Sequence Number.

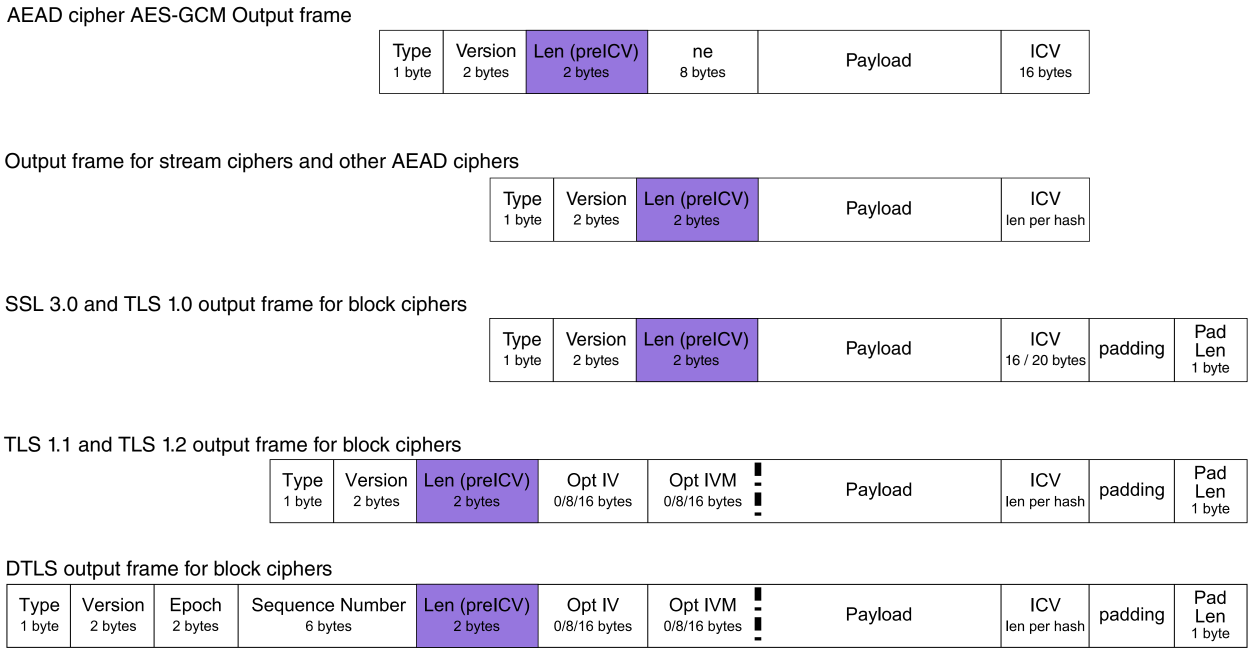


**Figure 9-23. Examples of decapsulation output frames for output format Option 2**

For Option 3, all versions of SSL, TLS, and DTLS return the same unprotected TLS record, regardless of cipher suite. This consists of Type, Version, Record Length, and Payload. For DTLS, Epoch and Sequence Number are also included. DTLS is virtually the same, including Epoch and Sequence Number.

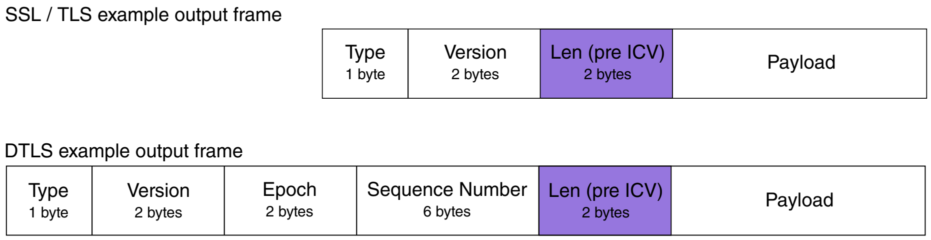


**Figure 9-24. Decapsulation example output frames for Option 3**

#### 9.2.1.5 SSL / TLS / DTLS error codes

This table lists the conditions under which SSL, TLS, or DTLS encapsulation or decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine. Authentication failure in decapsulation can also produce an ICV check error.

**Table 9-50. SSL, TLS, and DTLS encapsulation and decapsulation error conditions**

| Condition | Error status | Applies to: |
|---|---|---|
| PDB Options Field outFMT programmed to 11b | Invalid Setting in PDB | Decapsulation |
| Bad Protocol Operation command (often caused by protocol version and cipher suite mismatch) | undefined protocol command | both encapsulation and decapsulation |
| Required keys not present when protocol starts execution | Key not written before start of protocol | both encapsulation and decapsulation |
| Sequence Number rolls over back to zero | Sequence Number Overflow | both encapsulation and decapsulation, TLS only |
| Received a sequence number far enough below the latest sequence number it "fell off" the window | Anti-replay LATE error | DTLS decapsulation, if anti-replay is turned on |
| Received a recent, repeated sequence number | Anti-replay REPLAY error | DTLS decapsulation, if anti-replay is turned on |

### 9.2.2 Process for SSL 3.0 and TLS 1.0 record encapsulation

SEC performs SSL 3.0/TLS 1.0 encapsulation by doing the following:

1. Receives an input frame containing the payload.

2. Examines the contents of the DECO Protocol Override Register (DPOVRD)
    - If the MS bit (OVRD) is set, the job descriptor has selected a record-type override
    - If the MS bit is cleared, no record-type override is selected
3. Optional: If a record-type override has been selected, extracts the sequence number and the version field from the PDB and uses the least significant byte of DPOVRD for the record type
4. Optional: If no record-type override has been selected, extracts the sequence number, the record type and version fields from the PDB
5. Concatenates the sequence number, the record type (from whichever source is selected) and the version together, and pushes them into the Class 2 CHA for authentication (Note that version is excluded for SSL)
6. Pushes the record type and version fields onto the output frame
7. Increments the sequence number prior to writing it back to memory
8. Extracts and pushes the payload length, which is part of the frame description, as a 2-byte field into the Class 2 CHA for authentication
9. Adds the ICV length to the payload length, along with the length of any padding (including Pad Len) SEC adds; note that padding and Pad Len are added only for block ciphers (see Processing SSL 3.0 and TLS 1.0 record encapsulation with block ciphers)
10. Pushes the encrypted-payload length onto the output frame

Note that while the pre-ICV length is authenticated, the full record length (the length after ICV and padding is appended) is what is transmitted.

SEC supports two output frame formats: one for block ciphers and one for stream ciphers. See Processing SSL 3.0 and TLS 1.0 record encapsulation with block ciphers for more details.

### 9.2.2.1 Differences between SSL 3.0 and TLS 1.0 (record encapsulation)

For record encapsulation, the only differences between SSL 3.0 and TLS 1.0 are:

- For SSL:
    - A custom SSL-MAC is used for message authentication
    - Version field in header is 0300h
    - Version field is not part of authentication computation
    - AES is not part of any valid cipher suite
- For TLS 1.0:
    - A HMAC is used for message authentication
    - Version field in header is 0301h

- Version field is included in authentication computation
- Cipher suites including AES are supported

### 9.2.2.2 Processing SSL 3.0 and TLS 1.0 record encapsulation with block ciphers

This figure shows the process of SSL 3.0/TLS 1.0 when a block cipher (such as AES-CBC) is used.

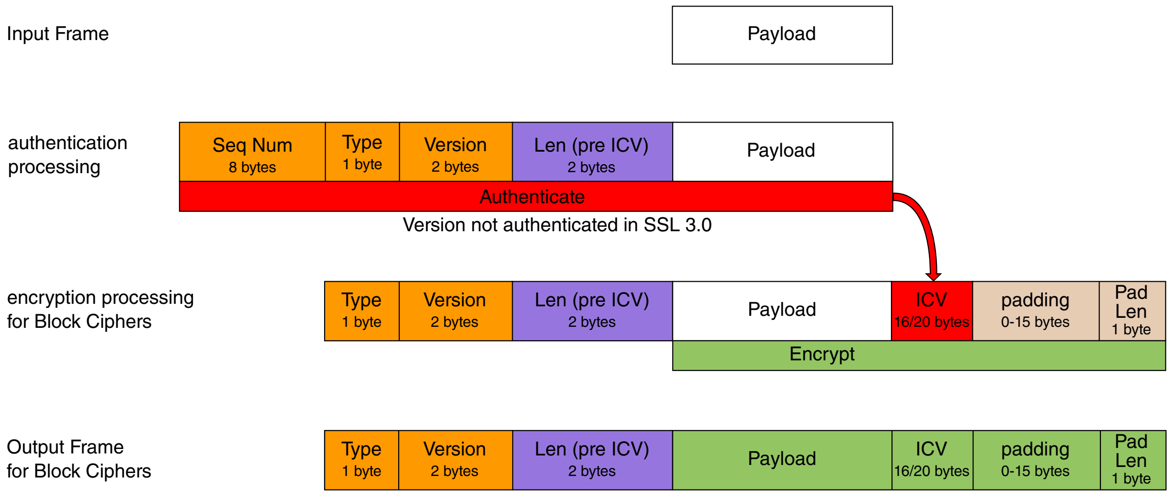


**Figure 9-25. SSL 3.0/TLS 1.0 block cipher encapsulation**

1. If the selected cipher suite includes a block cipher, the IV is extracted from the PDB and written to the CCB Class 1 Context Register (offset 0) as appropriate for the chosen block cipher. For AES, this IV is 16 bytes; for DES, it is 8 bytes.
2. The IV in the PDB is overwritten with the first block in the Class 1 Context Register (containing the final block of ciphertext) after encryption has completed.
3. As payload is extracted from the input frame, it is pushed onto the input-data FIFO and tagged for both encryption and authentication.
4. The last byte of the payload is the last byte authenticated. As a result, the ICV computed is also encrypted.
5. Following the ICV, SEC adds the minimal padding and a pad length byte such that the pad length byte is the last byte in a cipher block. Per the standard for TLS, the value of every byte of padding is the same as the value of the pad length byte.

**Example: Using an AES-128-CBC-SHA cipher suite with a plaintext-payload length of 32 bytes**

When using an AES-128-CBC-SHA cipher suite with a plaintext-payload length of 32 bytes, the ICV is 20 bytes.

Payload (32) + ICV (20) = 52 bytes.

The next multiple of the AES-CBC block size of 16 is 64.

Therefore, SEC appends 12 bytes, each with value 0Bh, immediately after the ICV.

6. This SEC-generated padding gets pushed into the Class 1 CHA for encryption, as shown in the following figure.
7. The resulting encrypted payload (which includes the encrypted ICV and the encrypted padding and pad length) is pushed onto the output frame, as shown in the following figure.

## 9.2.3 Process for SSL 3.0 and TLS 1.0 record decapsulation

SEC performs SSL 3.0/TLS 1.0 decapsulation by doing the following:

1. Receives an input frame consisting of the record header and the protected record.
2. Computes the length of the unprotected record, as described in Computing the pre-encrypted record length during decapsulation.
3. Decrypts the record payload, MAC (ICV), and any padding.
4. Extracts the implicit sequence number from the PDB, and updates the incremented number back to the PDB.
5. Computes a MAC on the implicit sequence number, the record header (using the unprotected record length), and the decrypted record payload. When SSL 3.0 is selected, the version field of the record header is excluded from the MAC computation.
6. Compares the computed MAC against the decrypted ICV. If the two do not match, SEC returns an ICV Check Fail value in the Job Completion Status Word, and updated PDB is not written back to memory.
7. Returns an output frame per the selected output format, as described in SSL, TLS, DTLS Decapsulation Output frame options.

**CAUTION**

For SSL and TLS, SEC does not perform any form of replay checking. Records are required to arrive in order. The use of an implicit sequence number guarantees that any records that arrive out of order result in an ICV failure.

### 9.2.3.1 SSL 3.0 and TLS 1.0 Record Decapsulation for block ciphers

This figure shows SSL 3.0/TLS 1.0 decapsulation when a block cipher is used.

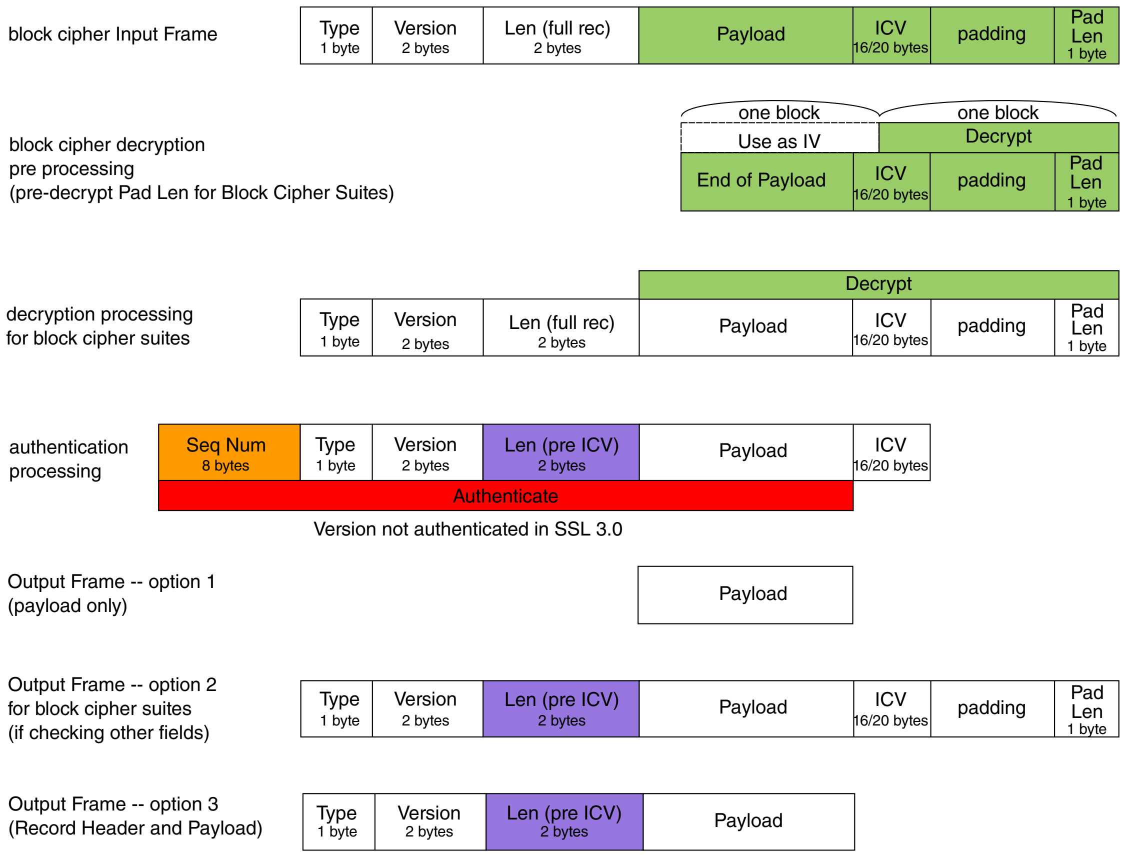


**Figure 9-26. SSL 3.0 / TLS 1.0 Block Cipher Decapsulation**

Block cipher decapsulation follows the process set forth in Process for SSL 3.0 and TLS 1.0 record decapsulation, but with a few extra steps added. In particular:

1. SEC supports block cipher decapsulation only with CBC mode. CBC mode requires an IV. Prior to beginning of decryption, the IV is extracted from the PDB and written to the Class 1 Context Register.
2. After decryption is complete, the final block of ciphertext is written back into the PDB to be used as the IV for the next record in sequence.

### 9.2.3.2 Differences between SSL 3.0 and TLS 1.0 (record decapsulation)

For record decapsulation, the only differences between SSL 3.0 and TLS 1.0 are the message authentication code and the version code applied.

- For SSL, a custom SSL-MAC is used for message authentication, and the version code is set to 0300h.
- For TLS, standard HMAC is used for message authentication, and the version code is set to 0301h.
- For SSL, the Version field is not included in the SSL-MAC Computation. It is included in the TLS HMAC computation.

## 9.2.4 Process for TLS 1.1 and TLS 1.2 record encapsulation

In general, SEC performs TLS 1.1/1.2 encapsulation by doing the following:

1. Begins encapsulation when it receives an input frame containing the payload
2. Extracts the sequence number (which is incremented and written back to memory) and version fields from the PDB
3. Extracts the record type from the PDB unless the MS bit of DPOVRD = 1, indicating that the record type field comes from the least significant byte of DPOVRD
4. Use the selected keyed MAC function to authenticate the sequence number and the record header (Type, Version, Record Length). Note that the Record Length value that is part of the authentication function does not include any additions to the record due to payload protection, such as the ICV.
    - For an HMAC, MDHA is used to perform the authentication computations, so the sequence number and record header are passed to the input Data FIFO tagged as Class 2 message data.
    - For AEAD using AES (CCM or GCM), the sequence number and record header are passed to the input Data FIFO tagged as AAD.
5. The record type and version fields are also pushed onto the output frame
6. The length of the protected record is computed and is pushed onto the output frame.

For more specifics, please refer to the appropriate section as follows:

- For encapsulation using either the AES-CBC or DES-CBC confidentiality algorithms, see Processing TLS 1.1 and TLS 1.2 record encapsulation with block ciphers (AES or DES).
- For encapsulation using the AES-Counter confidentiality algorithms, see Processing TLS 1.1 and TLS 1.2 record encapsulation with stream ciphers.
- For encapsulation using either AEAD algorithms AES-CCM or AES-GCM, see Processing TLS 1.1 and TLS 1.2 record encapsulation with AEAD ciphers.

#### 9.2.4.1 Differences between TLS 1.0, TLS 1.1, and TLS 1.2 Record Encapsulation

The main difference between TLS 1.0 and TLS 1.1 is in how block ciphers handle IVs. Because of security concerns, TLS 1.1 adopted the use of a random mask instead of a purely implicit IV, chained from the previous record. If a TLS 1.0 style implicit IV is used, the IV, which is the final block of ciphertext from the previously encapsulated record, is XORed with a random mask that is prepended to payload and treated as payload through the encryption process. The diagrams in this section represent this as IVM for IV Mask.

For cipher suites supported by TLS version 1.1, record encapsulation is identical between TLS 1.1 and TLS 1.2. For block ciphers, options bits IE and WB must be set to select a pure random IV.

TLS 1.2 specifies how to use AEAD (authenticated encryption with additional authenticated data) algorithms with TLS. Therefore SEC now supports AES-GCM and AES-CCM with TLS 1.2.

#### 9.2.4.2 Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation

An Initialization Vector (IV) is used to provide per-packet randomization. For CBC-Mode (Ciphers AES or DES), This value is used to randomize the input in a reproducible but unpredictable manner. This randomization prevents attacks based upon knowing the structure of the plaintext. The TLS standard has evolved to provide more cryptographically secure IVs. As such, SEC supports IV generation in three ways: Explicit IV, Implicit IV with Mask, or TLS 1.0 compatibility IV.

- If an Explicit IV is chosen, RNG generates a random IV (shown as Opt IV in Figure 9-28), which is written to the Class 1 Context Register and also to the output frame.
- If an Implicit IV with Mask is chosen, the IV is extracted from the PDB and written to the Class 1 Context Register. RNG also generates a random IV Mask (shown as Opt IVM in Figure 9-28), which is encrypted but not authenticated.
- For TLS 1.0 Compatibility Mode, IV is extracted from the PDB, and is written to the Class 1 Context Register. The final block of the encrypted record is saved back to memory for use as the next IV. This truly chained IV is supplemented with a RNG-generated IV Mask, which is prepended to the payload and encrypted, but not authenticated.

Other cipher suites supported by SEC that use an IV are AEAD algorithms, and AES-Counter based cipher suites. In all these ciphers, the IV does not randomize the data, but instead randomizes the key-stream generation process.

- For AES-CCM and AES-GCM, RNG generates a random nonce_explicit (shown in Figure 9-30 and in Figure 9-27 as ne), which is combined with SALT from the PDB to form the nonce. SALT is a form of Write_IV, generated by the PRF as part of key generation.
- For AES-GCM the 12-byte nonce is passed into the input Data FIFO tagged as IV.
- For AES-CCM, the nonce is formed from the appropriate 4 byte Write IV (server or client), concatenated with the 8 byte Sequence number. The nonce is in turn used to create $CTR_0$ and $B_0$.

AES-CCM mode has a much more complex use model for the IV than other modes. As said above, the IV is combined with WRITE_IV32 that has been programmed into the PDB to form a nonce identical to that generated for AES-GCM. This nonce is used differently for AES-CCM, and the SEC TLS state machine contains special instructions to explicitly construct $B_0$ and $CTR_0$, and to write them to the Class 1 Context Register.

$B_0$ is constructed:

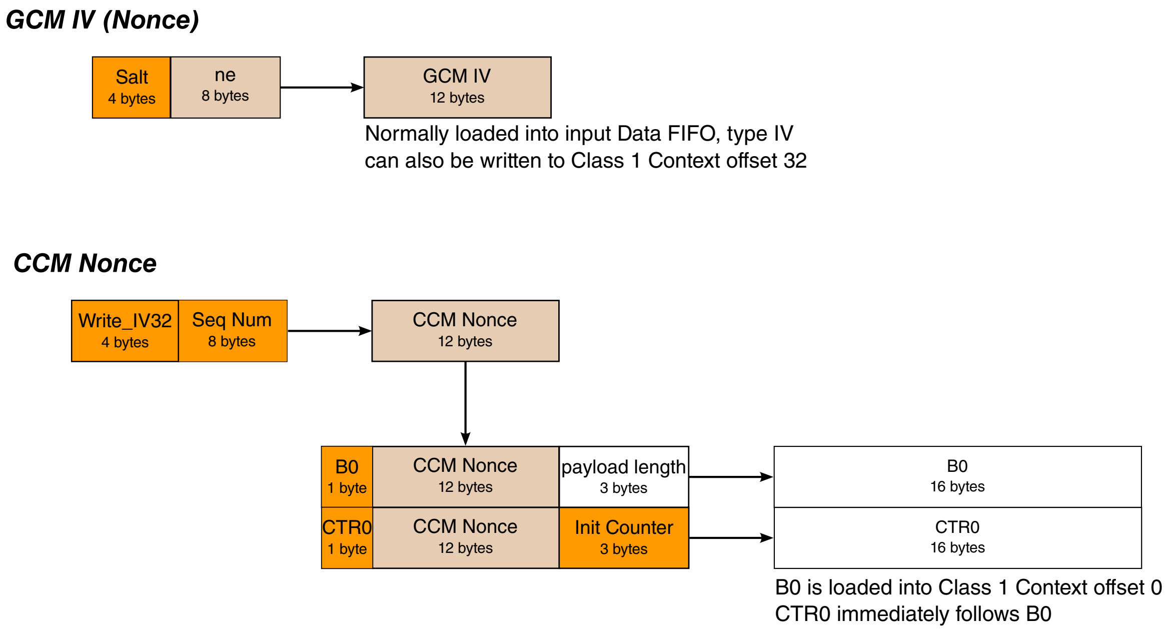
- The first byte is taken from the PDB $B_0$ Flags field
- The next 12 bytes consist of the Nonce
- The final 3 bytes are constructed by the state machine and reflect the length of the payload being encapsulated

$CTR_0$ is constructed:

- The first byte is taken from the PDB $CTR_0$ Flags field
- The next 12 bytes consist of the Nonce
- The final 3 bytes are taken from the $CTR_0$ Constant field in the PDB.

**NOTE**

Proper CCM-mode encapsulation relies upon proper programming of $B_0$ Flags, $CTR_0$ Flags, and $CTR_0$ Constant fields into the Shared Descriptor PDB. Per RFCs 3610 and 6655, the $CTR_0$ Constant should be programmed with zeros, $CTR_0$ Flags should be programed with 0x02, and $B_0$ Flags should be programmed to 0x7A for ciphersuites requiring a 16-byte authentication tag, and to 0x5A for cipher suites requiring an 8-byte authentication tag.



Normally loaded into input Data FIFO, type IV can also be written to Class 1 Context offset 32

B0 is loaded into Class 1 Context offset 0
CTR0 immediately follows B0

**Figure 9-27. TLS 1.1 /1.2 Nonce Generation and use for AEAD ciphers**

## 9.2.4.3 Processing TLS 1.1 and TLS 1.2 record encapsulation with block ciphers (AES or DES)

This figure shows TLS 1.1/1.2 authentication for encapsulation using block ciphers (AES or DES).

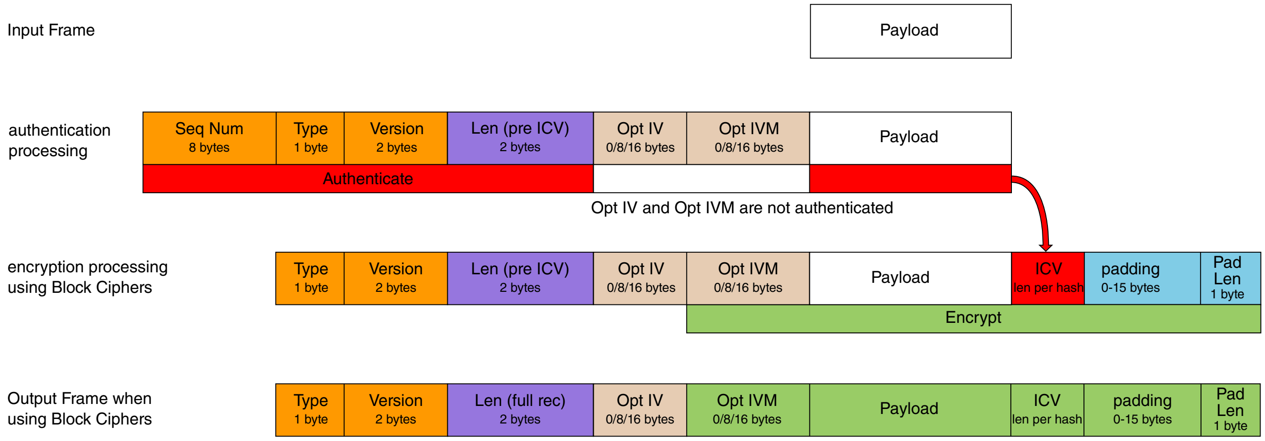


**Figure 9-28. TLS 1.1/1.2 authentication for encapsulation using block ciphers (AES or DES)**

For block cipher processing, the length of either the IV or the IV Mask is also added to the record header length field. Although the transmitted length field is of the record (except the TLS record header), the length field used during authentication is the length of the payload itself.

Payload is processed as follows:

1. As payload is extracted from the Input Frame, it is pushed onto the input-data FIFO, tagged for both encryption and authentication.
2. The last byte of the payload is the last byte authenticated; the last word of data is so tagged in the iNformation FIFO.
3. The ICV computed as a result of authentication is pushed back into the Class 1 CHA, and is also encrypted.
4. Following the ICV, SEC adds the minimal padding and a pad length byte such that the pad length byte is the last byte in a cipher block. The value of every byte of padding is the same as the value of the pad length byte.

**Example: An AES-128-CBC-SHA ciphersuite with a plaintext-payload length of 32 bytes**

Using an AES-128-CBC-SHA ciphersuite with a plaintext-payload length of 32 bytes, the ICV is 20 bytes.

Payload (32) + ICV (20) = 52 bytes.

The next multiple of the AES-CBC block size of 16 is 64.

Therefore, SEC appends 12 bytes, each with value 0Bh, immediately after the ICV.

5. This SEC-generated padding is pushed into the Class 1 CHA for encryption.
6. The resulting encrypted payload, which includes the encrypted ICV and the encrypted padding and pad length, is pushed onto the output frame.

### 9.2.4.4 Processing TLS 1.1 and TLS 1.2 record encapsulation with stream ciphers

IVs are not transmitted for stream ciphers.

Payload is processed as follows:

1. As payload is extracted from the Input Frame, it is pushed onto the input-data FIFO, tagged for both encryption and authentication.

2. The last byte of the payload is the last byte authenticated; the last word of data is so tagged in the iNformation FIFO.
3. The ICV computed as a result of authentication is pushed back into the Class 1 CHA, and is also encrypted.
4. The resulting encrypted payload (which includes the encrypted ICV is pushed onto the output frame.

Note that for stream ciphers, padding is not required, so the padding and Pad Len fields are skipped altogether.

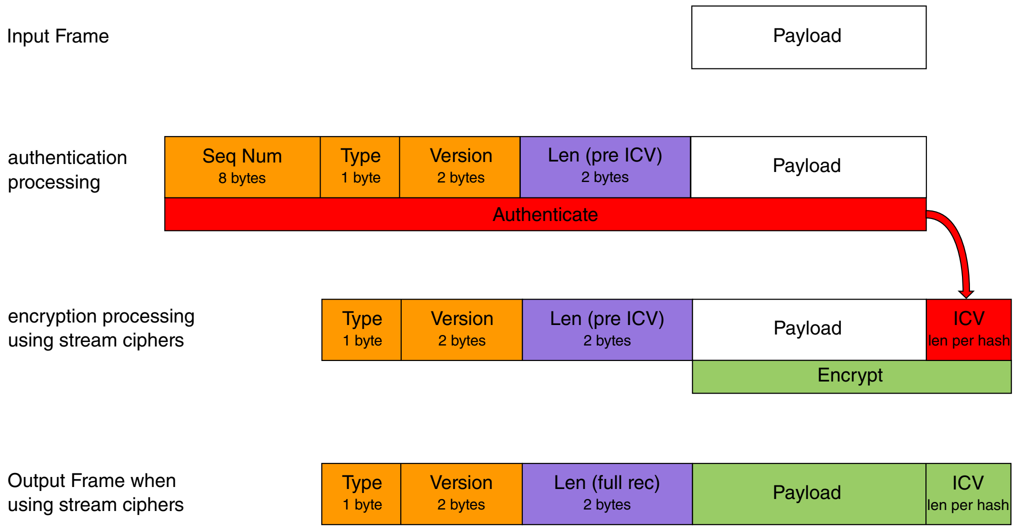


Figure 9-29. TLS 1.1 /1.2 encapsulation for stream ciphers

### 9.2.4.5 Processing TLS 1.1 and TLS 1.2 record encapsulation with AEAD ciphers

AEAD stands for Authenticated Encryption with Additional Data. Introduced to TLS with version 1.2, it provides a new structure: an independently vetted algorithm that combines encryption and authentication in one.

For both AES-CCM and AES-GCM, payload is processed as follows:

1. As payload is extracted from the input frame, it is pushed into the input-data FIFO and tagged for both encryption and authentication.
2. The last byte of the payload is the last byte authenticated; the last word of data is so tagged in the iNformation FIFO.
3. The ICV computed as a result of authentication is also encrypted.

4. The resulting encrypted payload, which includes the ICV, is pushed onto the output frame, as shown in the following figure.

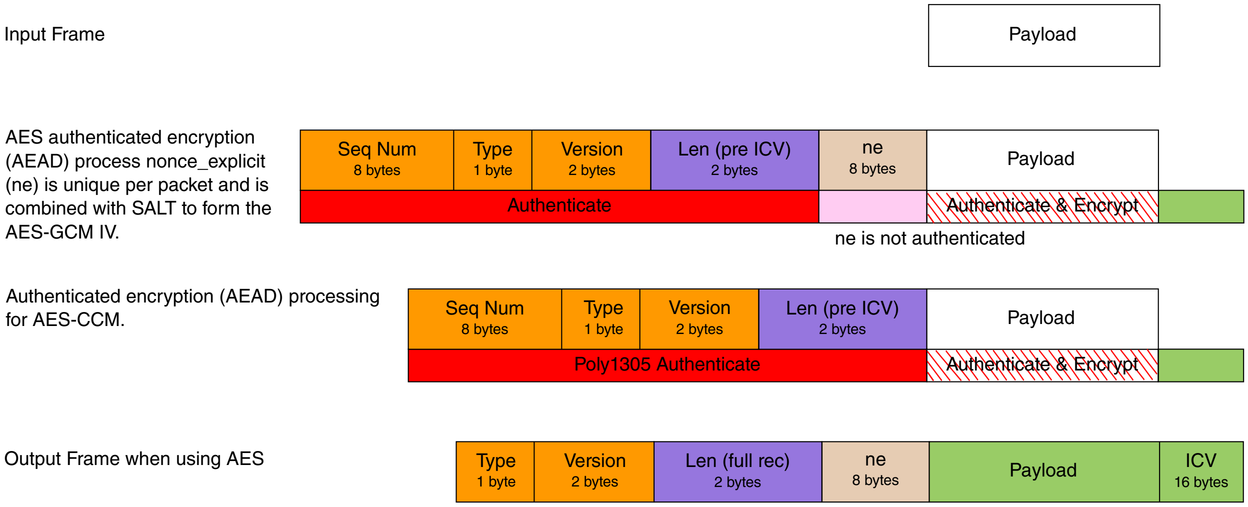


**Figure 9-30. TLS 1.2 encapsulation for AEAD ciphers**

For AES-GCM ciphers, TLS 1.2 defines a special value: the nonce. The default nonce is 12 bytes and is comprised of two values from two sources: the salt and the nonce_explicit (ne). Nonce generation is discussed in Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation.

- Salt is generated as key material and remains constant throughout the lifetime of the keys.
- nonce_explicit is generated randomly by RNG for each frame.

In the AES-GCM algorithm, the 12-byte nonce is used as the GCM-IV, which sets the first counter value used for encryption.

The TLS 1.2 protocol thread also supports encapsulation using AEAD algorithm AES-CCM. AES-CCM uses a 12-byte nonce generated from the 4 byte Write IV and the 8-byte Sequence Number. However the usage is different, as is the method for programming AESA with the nonce. Section Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation provides more details.

## 9.2.5 Process for TLS 1.1 and TLS 1.2 record decapsulation

For TLS record decapsulation, SEC must authenticate:

- All the plaintext fields

- Implicit sequence number
- Decrypted payload

The details depend upon the type of cipher suite used. For cipher suites based on block ciphers (using either AES-CBC or DES-CBC), refer to Decapsulation of TLS 1.1 and TLS 1.2 records when a block cipher is used. For cipher suites based on stream ciphers, refer to Decapsulation of TLS 1.1 and TLS 1.2 records when a stream cipher is used. For cipher suites based on AEADs (Authenticated Encryption with Additional Data), refer to Decapsulation of TLS 1.2 records when an AEAD is used.

In general, processing consists of two different computations: decryption, and integrity checking. During encapsulation, the record and the original header was integrity protected; the record length reflecting the header and payload, but not any other additions resulting from encapsulation. As a result, for decapsulation, the record header has to be modified to reflect the original length.

For block and stream cipher encapsulation, the integrity computation was performed on plaintext. So for decapsulation, the payload must be decrypted before being integrity checked. AEAD ciphers define an algorithm (or pair of algorithms) that perform both encryption and intregity computations. TLS performs the algorithm as specified by the algorithm definition.

In all cases, the sequence number is extracted from the PDB and is included in the integrity computation, prior to the record header.

### 9.2.5.1 Decapsulation of TLS 1.1 and TLS 1.2 records when a stream cipher is used

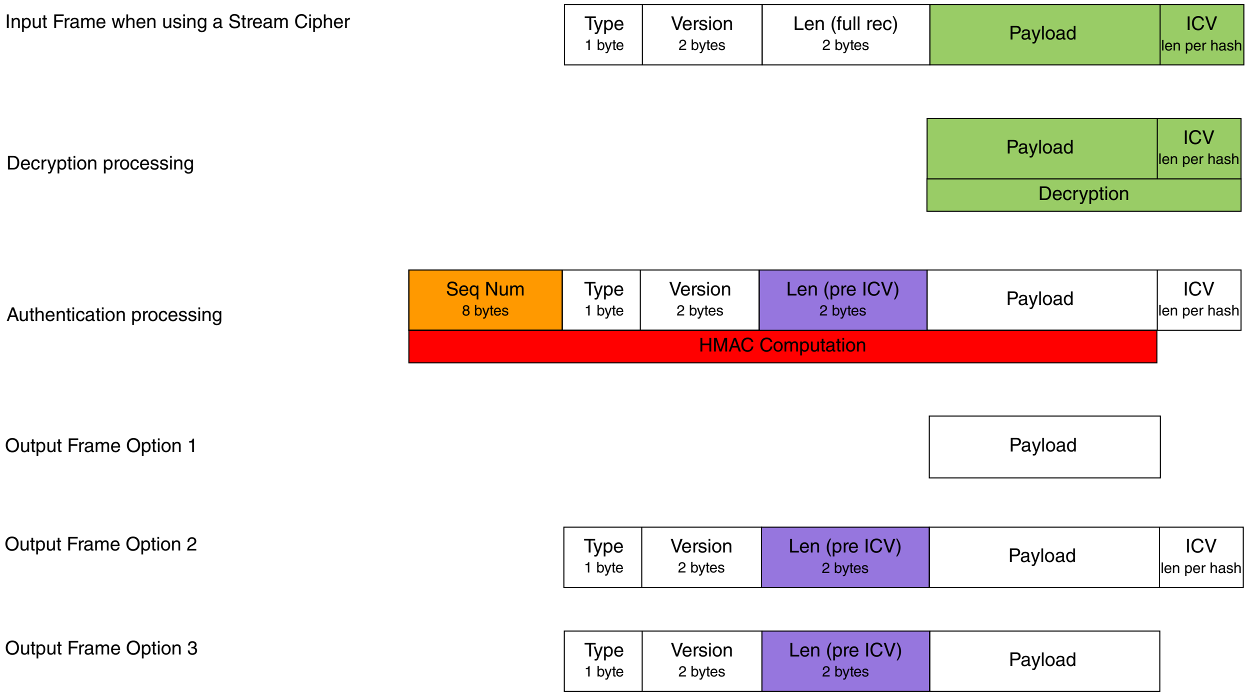


**Figure 9-31. TLS 1.1 / 1.2 Decapsulation when using a Stream Cipher**

A stream cipher is the simplest TLS construction for SEC to decapsulate. The record header gets pre-processed, so that the length of the ICV can be removed from the record length. The modified header is then passed into MDHA to be the first segment of input frame to be integrity checked.

Subsequent bytes are all encrypted during encapsulation, thereby requiring decapsulation. All encrypted bytes up to but not including the encrypted ICV are tagged "class 1 outsnoop to class 2", thereby automatically shunting the decrypted bytes into MDHA for integrity checking.

Decrypted payload is written out to the memory location specified. If the PDB Options outFMT field selects, the record header preceeds the payload, and if the outFMT selects, the decrypted ICV will be written out as well. See Table 9-47 for details on programming outFMT.

The ICV, after decryption, is put into MDHA as an ICV. MDHA, after completing the integrity computation, compares the computed ICV against the decrypted ICV, and signals an ICV error if the match fails.

### 9.2.5.2 Decapsulation of TLS 1.1 and TLS 1.2 records when a block cipher is used

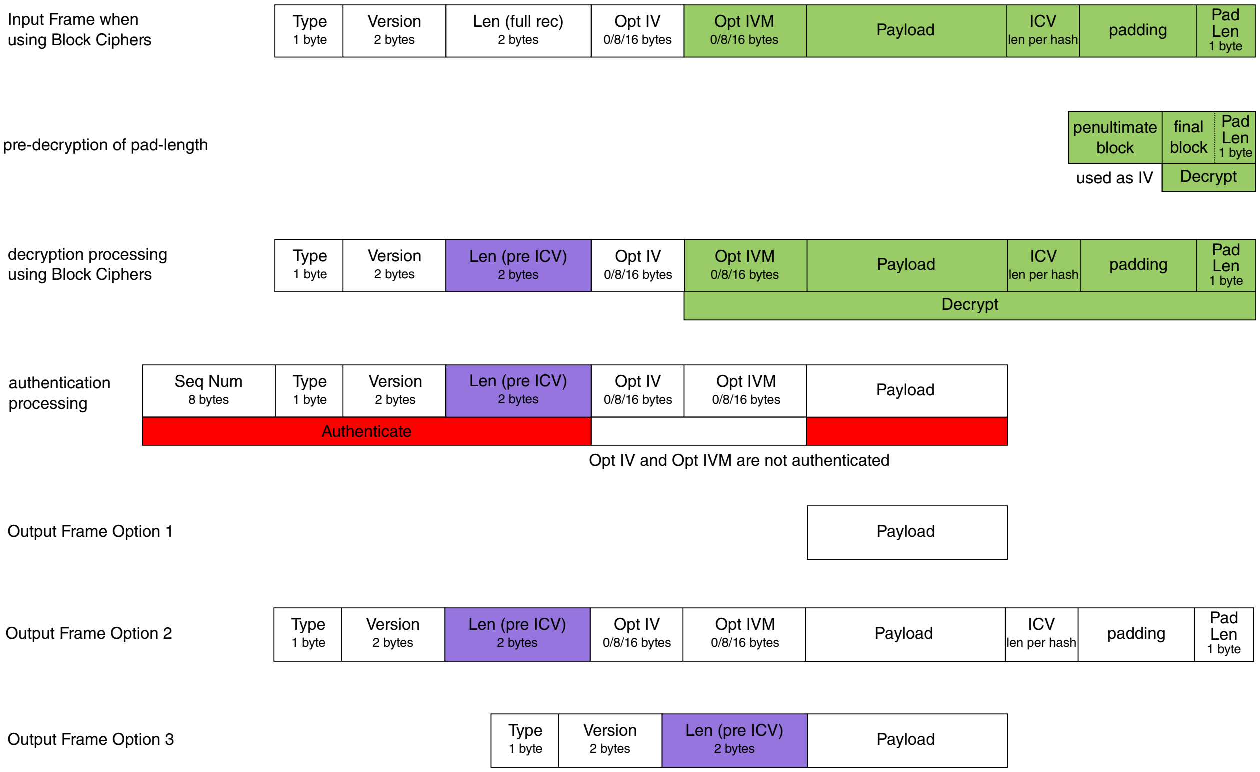


**Figure 9-32. TLS 1.1 / 1.2 Decapsulation when using a Block Cipher**

A block cipher (AES-CBC, DES-CBC) requires that encryption be performed on multiples of the block size. TLS specifies padding when a block cipher is used, and a pad-length byte is the last byte of the encrypted record. The integrity computation includes the header, which includes the *original* record length. That is, the length of all fields added as part of the encapsulation process need to be subtracted before authentication can begin. In order to compute the HMAC in parallel with decryption, the final byte of the record -- the pad-length byte, must be decrypted *first.* As a result, SEC will start by loading the final two blocks of the record (16 bytes for DES, 32 bytes for AES), and use the first half as IV for decrypting the final block. Once the pad-length byte is decrypted, this material is thrown away, the pre-encapsulation record length is computed, and used for the integrity check process.

From this point, block cipher decapsulation proceeds much like stream cipher decapsulation -- the modified header is passed into MDHA to be the first segment of the input frame to be integrity checked.

Block Ciphers use an IV to randomize the input, to thwart cryptanalysis. TLS 1.0 uses a chained IV -- literally the final block of ciphertext in record i becomes the IV for record i +1. TLS 1.2 specifies use of a random IV; setting PDB options WB=0 and IE=1 is used to specify a pure random IV that is explicitly part of the encapsulated record. For the IV, TLS 1.1 is a transitional standard that allows for an implicit IV or an explicit IV, and allows a random mask to be used. IE and WB can be programmed as required for TLS 1.1. The IV, whether implicit or explicit, is not included in the integrity computation. More info on programming WB and IE can be found at Table 9-47.

Subsequent bytes are all encrypted during encapsulation, thereby requiring decapsulation. All encrypted bytes up to but not including the encrypted ICV are tagged "class 1 outsnoop to class 2", thereby automatically shunting the decrypted bytes into MDHA for integrity checking.

Decrypted payload is written out to the memory location specified. If the PDB Options outFMT field selects, the record header preceeds the payload, and if the outFMT selects, the decrypted ICV and padding will be written out as well. SEC does not check the contents of the padding bytes after decryption, so outFMT should be programmed to 01b to select output type 2 if software checking of the padding bytes is required. Note that the ICV computation does cover the padding bytes. See Table 9-47 for details on programming outFMT.

The ICV, after decryption, is put into MDHA as an ICV. MDHA, after completing the integrity computation, compares the computed ICV against the decrypted ICV, and signals an ICV error if the match fails.

#### 9.2.5.3 Decapsulation of TLS 1.2 records when an AEAD is used

TLS 1.2 adds the capability to use AEAD ciphers for record protection.

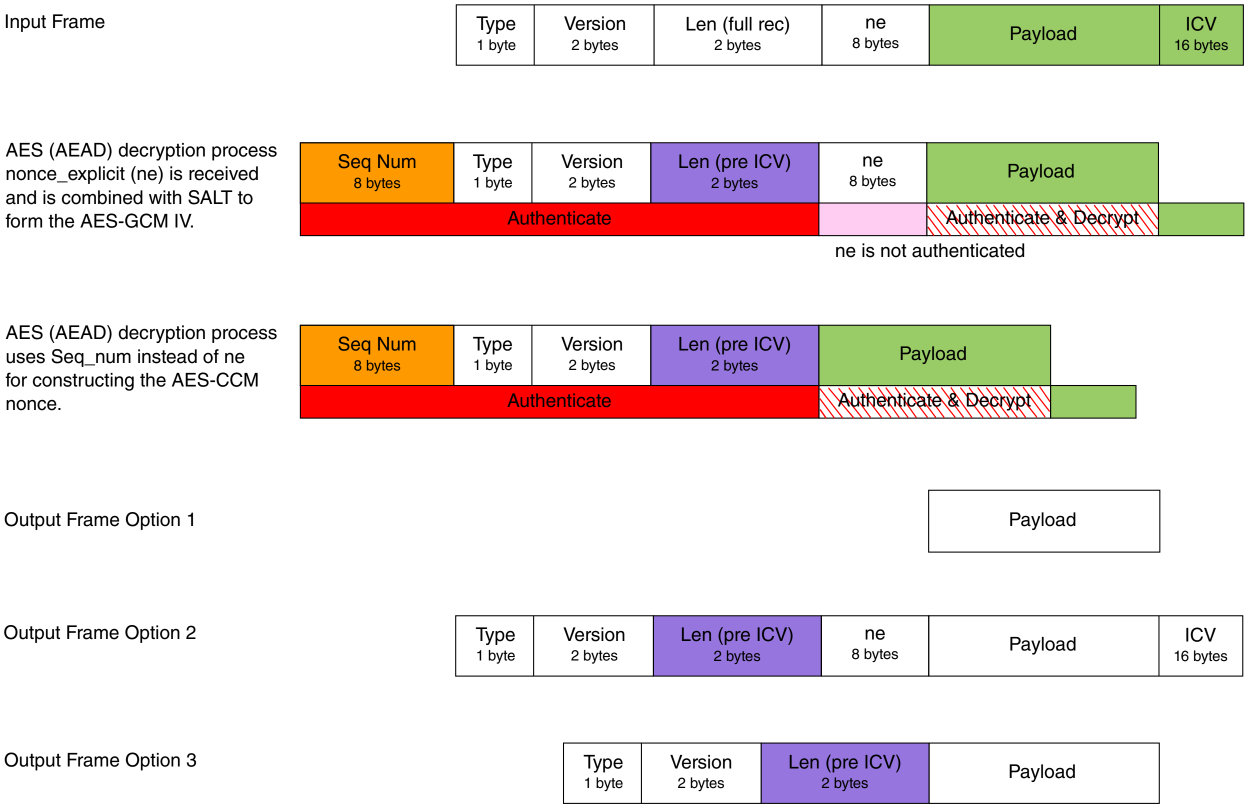


**Figure 9-33. TLS 1.1 / 1.2 Decapsulation when using an AEAD Cipher**

AEAD ciphers often include use of a nonce or an IV. Please refer to Figure 9-27 for more information about constructing the nonce or IV for encapsulation. Note that in the case of decapsulation, the received nonce_explicit is used instead of generated.

AES-GCM and AES-CCM processes the record very similarly to a stream cipher, that nonce_explicit (ne in figures) is not included in either decryption nor the integrity check. The nonce_explicit is used as a parameter to randomize the key-stream generated by AES-Counter mode. (AES-Counter is the confidentiality portion of AES-GCM and AES-CCM.

AES-CCM, like AES-GCM, is a AEAD cipher using AES-Counter for confidentiality. However SEC implements AES-CCM a little differently, in that it creates a 12-byte NONCE from the 4-byte Write IV and the 8-byte Seqence Number. The 12-byte nonce is then used as shown in Figure 9-27 to construct the AES-CCM IV ($B_0$) and Initial Counter ($CTR_0$).

### 9.2.6 Process for DTLS record encapsulation

This version of SEC supports DTLS versions 1.0 and 1.2, and supports several cipher suites, including:

- Triple-DES-CBC with HMAC-SHA-1
- AES-128-CBC with HMAC-SHA-1
- AES-256-CBC with HMAC-SHA-256
- AES-128-CTR with HMAC-SHA-1
- AES-256-GCM
- AES-128-CCM-8

Note this list is not complete. For a complete list, please see Table 7-55.

#### 9.2.6.1 Differences between DTLS and TLS

DTLS 1.0 is a variant of TLS 1.1, and DTLS 1.2 is a variant of TLS 1.2. The DTLS procedure for record encapsulation is different from TLS in that:

- DTLS requires the insertion of an explicit sequence number.
- DTLS authenticates the header fields in a different order than transmitted. The field order for authentication matches how TLS performs authentication.

The explicit Sequence Number is necessary to support the DTLS-specific requirement of support for out-of-order reception of records. Because TLS uses implicit sequence numbers, it cannot support out-of-order reception.

### 9.2.6.2 Process of DTLS Record Encapsulation when using a Block Cipher

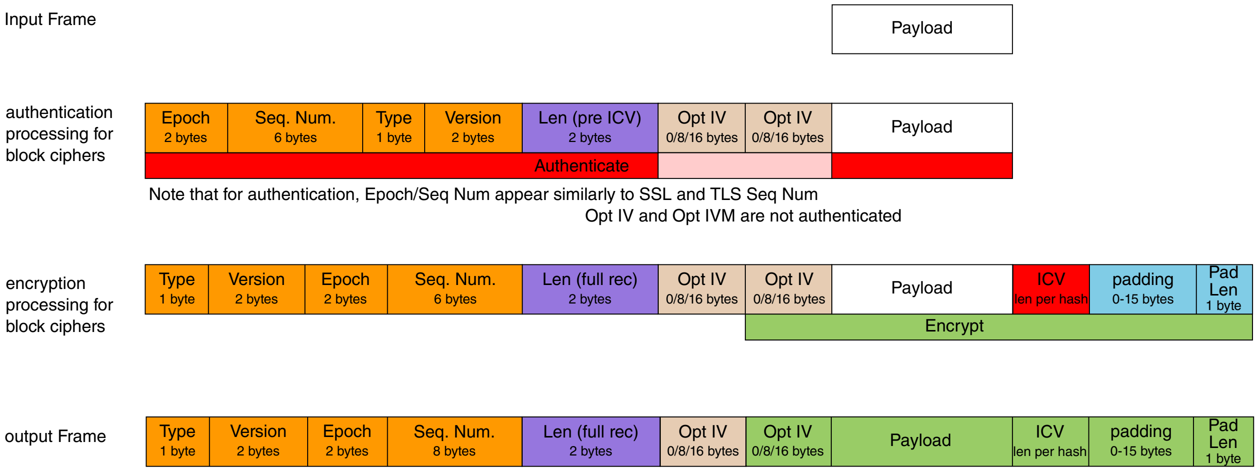


**Figure 9-34. DTLS Record Encapsulation when using a Block Cipher**

The DTLS block cipher encapsulation procedure is as follows:

1. SEC begins encapsulation when it receives an input frame containing the payload requiring encapsulation.
2. SEC extracts the version, epoch, and sequence number fields from the PDB.
3. SEC checks the most significant bit of the Datapath Override register:
   - If DPOVRD[MS] is cleared, SEC extracts the record type from the PDB
   - If DPOVRD[MS] is set, SEC instead takes the record type field comes from the least significant byte of the DECO Protocol Override Register.
4. SEC pushes the concatenated epoch, sequence number, record type, and version into the Class 2 CHA for authentication and then onto the output frame.

**NOTE**

The order of these fields for authentication is different than for transmission.

5. The payload length, which is part of the frame description, is extracted and pushed as a two byte field into the Class 2 CHA for authentication, after adding to it to reflect the length of the record header and the IV. In the diagram, this is called Len (pre ICV)
6. Another record length reflecting the added ICV, padding, and the pad length byte is computed. This version of length is pushed into the output frame, and is transmitted in the clear. Pad Length is computed to be the minimum required for the chosen cipher suite, such that the total length is the smallest multiple of the block size of the

chosen cipher required to be able to encrypt Payload, ICV, padding, and the pad length byte.

7. The sequence number is then incremented and updated to the PDB in memory.
8. PDB Options bits IE and WB are examined to determine how to construct the IV.
   - If w/b is set, then the last block of ciphertext from the previous block was saved in the PDB IV field. This is extracted and XORed with a random number to form the IV
   - If e/i is set, then the the IV is explicitly included prior to the payload. Otherwise, the IV is encrypted as if part of the payload.

**NOTE**

The proper setting for DTLS 1.2 is w/b=0 and e/i=1.

9. Payload, ICV, padding, and the pad length byte are encrypted use the selected CHA, using CBC mode; the encryption product is pushed into the output frame.

### 9.2.6.3 Process of DTLS Record Encapsulation when using a Stream Cipher

SEC performs DTLS AES-Counter mode encapsulation very similarly to TLS 1.1 record encapsulation as described in Processing TLS 1.1 and TLS 1.2 record encapsulation with stream ciphers. Like DTLS Block cipher encapsulation, the major difference between TLS AES-Counter and DTLS AES-Counter processing is authentication placement of the explicit sequence number.

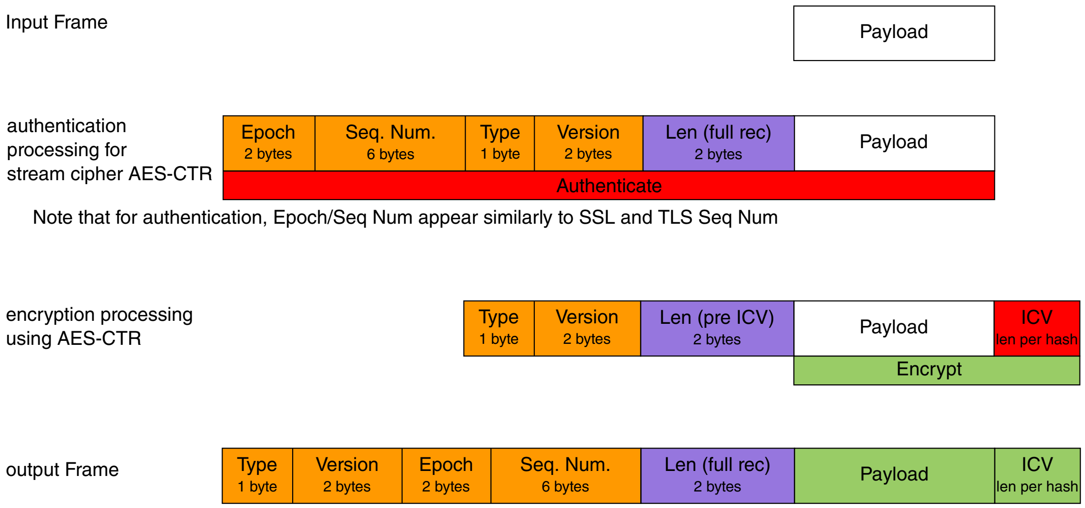


**Figure 9-35. DTLS Record Encapsulation when using a Stream Cipher**

The DTLS stream-cipher encapsulation procedure is as follows:

1. SEC begins encapsulation when it receives an input frame containing the payload requiring encapsulation.
2. SEC extracts the version, epoch, and sequence number fields from the PDB.
3. SEC pushes the concatenated epoch, sequence number, record type, and version into the Class 2 CHA for authentication and then onto the output frame.

**NOTE**

The order of these fields for authentication is different than for transmission.

4. The sequence number and Write_IV are extracted from the PDB and written into Class 1 Context to become the packets' initial counter value for AES-Counter Encryption.
5. The sequence number is then incremented and updated to the PDB in memory.
6. The payload length, which is part of the frame description, is extracted and pushed as a two byte field into the Class 2 CHA for authentication.
7. The ICV length is added to the payload length. This variant of payload length is pushed into the output FIFO and becomes part of the output frame
8. Payload and ICV are encrypted using AES Counter mode; the encryption product is pushed into the output frame.

### 9.2.6.4 DTLS 1.2 Record Encapsulation when using an AEAD Cipher

This version of SEC supports two different AEAD ciphers:

- AES-GCM (using 128, 192, or 256 bit keys)
- AES-CCM (using 128, 192, or 256 bit keys)

Each AEAD cipher suite operates a little differently. Critical to the cipher is the underlying nonce. In each case, nonce / IV construction is performed identically to how built for TLS 1.2, as described in Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation; particularly as shown in Figure 9-27. RFC 6347 specifies that DTLS 1.2 AEAD cipher suites operate identically the same as with TLS 1.2.

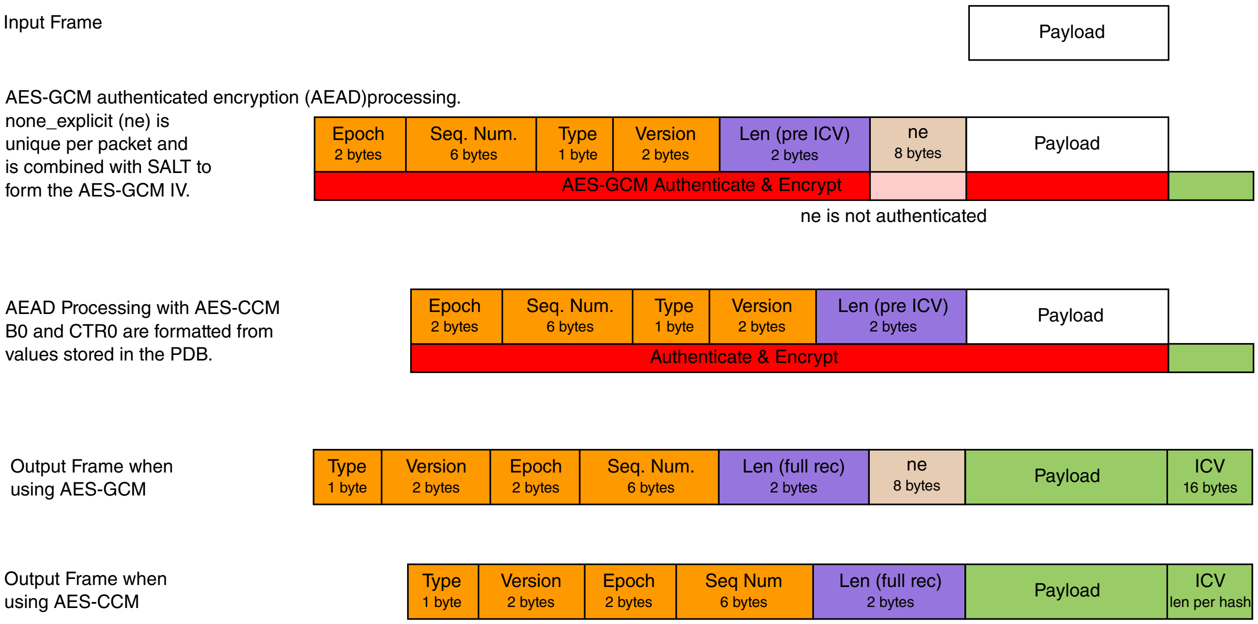


**Figure 9-36. DTLS Record Encapsulation when using an AEAD Cipher**

For AEAD ciphers, the authenticate-only segment consists of a reordered record header, and is constructed by extracting the epoch, sequence number, type and version from the PDB, and then by using the record length, which reflects the header and payload length, but not the length of the ICV, nor for AES-GCM, of the Nonce Explicit. This construct is passed to the class 1 CHA as type AAD.

The record header is passed to the output frame in proper order, consisting of type, version, epoch, sequence number, and for AES-GCM, nonce_explicit. Nonce Explicit is a random number. Included in the record header is the full record version of the record length, which is the length of the encapsulated record, and includes the length of ICV, and for AES-GCM, the length of nonce_explicit.

## 9.2.7 Process for DTLS record decapsulation

This version of SEC supports DTLS versions 1.0 and 1.2, and supports several cipher suites, including:

- Triple-DES-CBC with HMAC-SHA-256
- AES-256-CBC with HMAC-SHA-1
- AES-128-CCM-16

Note this list is not complete. For a complete list, please see Table 7-55.

#### 9.2.7.1 Differences between DTLS and TLS

DTLS 1.0 is a variant of TLS 1.1, and DTLS 1.2 is a variant of TLS 1.2. The DTLS procedure for record decapsulation is different from TLS in that:

- DTLS requires the reception of an explicit sequence number.
- DTLS authenticates the header fields in a different order than received The field order for authentication matches how TLS performs authentication.

The explicit Sequence Number is necessary to support the DTLS-specific requirement of support for out-of-order reception of records. Because TLS uses implicit sequence numbers, it cannot support out-of-order reception.

Because DTLS support out-of-order reception, anti-replay checking is available as part of SEC DTLS decapsulation processing. Please refer to Anti-Replay built-in checking for more details on how anti-replay works. Anti-Replay operation is controlled by the PDB options byte, and state is maintained in the PDB. Details can be found in PDB use and format for SSL, TLS, and DTLS encapsulation and decapsulation.

### 9.2.7.2 Process of DTLS Record Decapsulation when using a Block Cipher

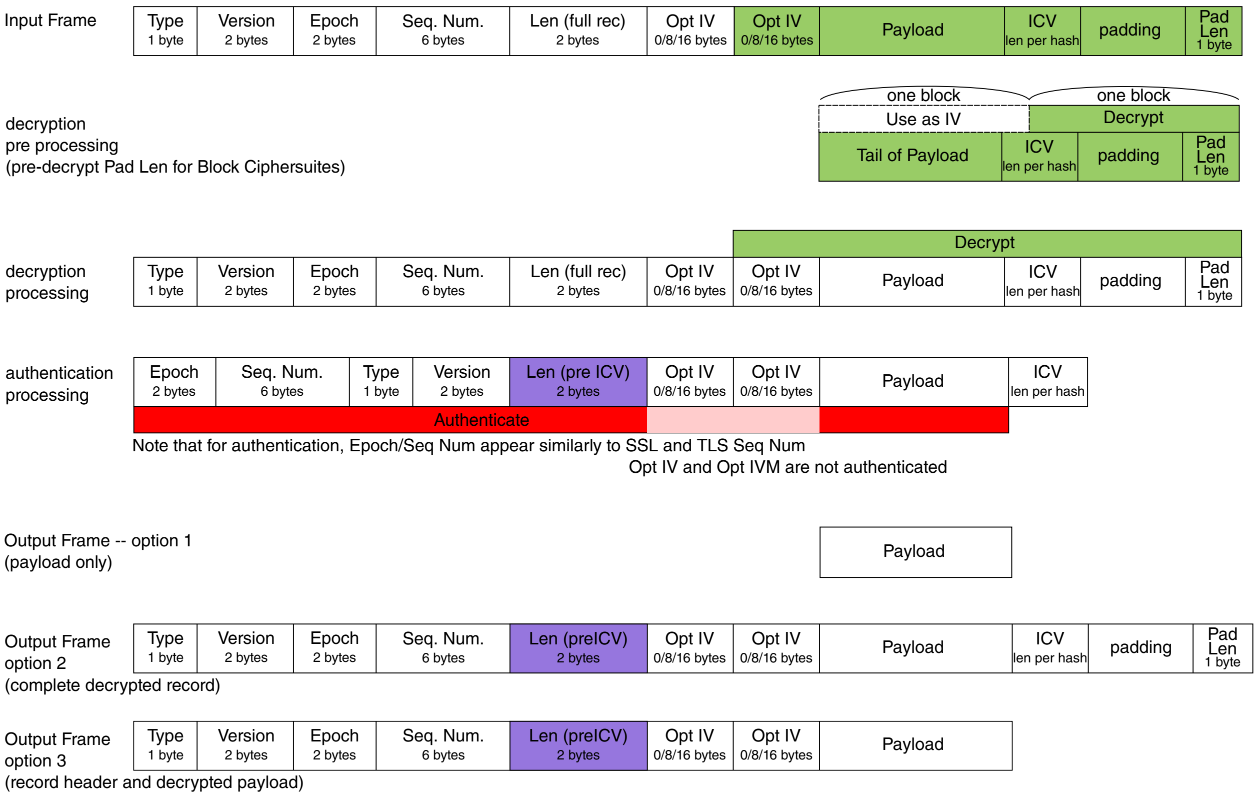


**Figure 9-37. DTLS Record Decapsulation when using a Block Cipher**

The DTLS block cipher decapsulation procedure is as follows:

1. SEC begins decapsulation when it receives an input frame containing the encapsulated payload.
2. SEC determines the number of bytes of padding by fast-forwarding to the last two cipher blocks of the message, and uses the second to last block as an IV to decrypt the final block, the last byte of which is the pad length byte.
3. SEC rewinds to beginning of the frame, and extracts the type, version, epoch, sequence number, and record length fields from the input frame.
4. SEC pushes the concatenated epoch, sequence number, record type, and version into the Class 2 CHA.

**NOTE**

The order of these fields for authentication is different than received.

5. SEC takes the received record length field, which reflects the length of the encapsulated frame, and subtracts the length of ICV, padding, and the pad length byte. This computed record length (shown in diagrams as "length (pre ICV)") is passed into the class 2 CHA for authentication.
6. If the PDB Options byte field OutFMT so selects, the record header, including the adjusted record length field, is written to the output frame.
7. If enabled, anti-replay checking is performed, updating state to the PDB.
8. Decryption is performed. The decrypted payload is passed into the class 2 CHA for authentication, and is written to the output frame.
9. Once the HMAC computation is complete, the recieved ICV is compared to the computed ICV. Any mismatch between the two ICVs generates an error and is signalled back.

### 9.2.7.3 Process of DTLS Record Decapsulation when using a Stream Cipher

SEC performs DTLS AES-Counter mode decapsulation very similarly to TLS 1.1 record decapsulation as described in Decapsulation of TLS 1.1 and TLS 1.2 records when a stream cipher is used. Like DTLS Block cipher decapsulation, the major difference between TLS AES-Counter and DTLS AES-Counter processing is authentication placement of the explicit sequence number.

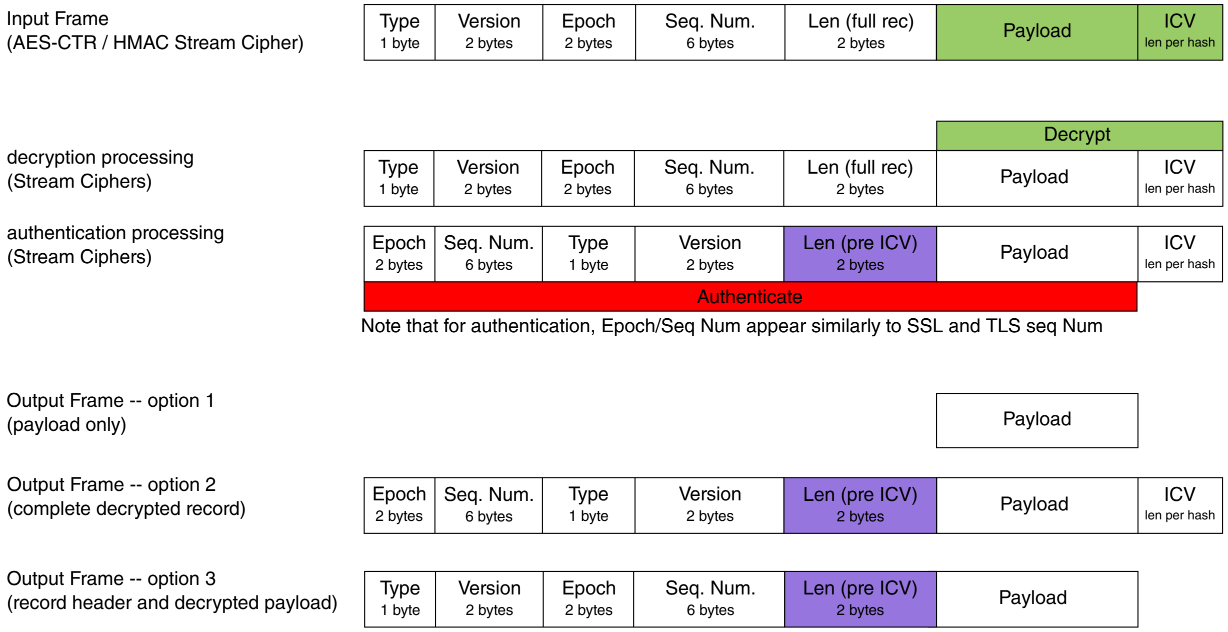


**Figure 9-38. DTLS Record Decapsulation when using a Stream Cipher**

The DTLS stream-cipher decapsulation procedure is as follows:

1. SEC begins decapsulation when it receives an input frame containing the encapsulated record.
2. SEC extracts the version, epoch, sequence number, and record length fields from the input frame.
3. SEC pushes the concatenated epoch, sequence number, record type, and version into the Class 2 CHA for authentication and then onto the output frame.

**NOTE**

The order of these fields for authentication is different than from the input frame.

4. The received record length is adjusted by subtracting the length of ICV from it. The adjusted record length is then pushed into the class 2 CHA for authentication.
5. If so selected by the PDB Options byte field outFMT, the record header, including the adjusted record length, is writen to the output frame.
6. The sequence number and Write_IV are extracted from the PDB and written into Class 1 Context to become the packets' initial counter value for AES-Counter Decryption.
7. If enabled, SEC performs anti-replay checking, and updates PDB state as a result.
8. Payload and ICV are decrypted using AES Counter mode; the decrypted payload is pushed into the output frame and into the class 2 CHA for authentication processing.
9. Once the HMAC computation is complete, it is compared to the received HMAC, and an error is signalled if not identical.
10. If enabled by outFMT, then the decrypted ICV is written to the output frame.

### 9.2.7.4 DTLS 1.2 Record Decapsulation when using an AEAD Cipher

This version of SEC supports two different AEAD ciphers:

- AES-GCM (using 128, 192, or 256 bit keys)
- AES-CCM (using 128, 192, or 256 bit keys)

Each AEAD cipher suite operates a little differently. Critical to the cipher is the underlying nonce. In each case, nonce / IV construction is performed identically to how built for TLS 1.2, as described in Support for IV generation in TLS 1.1 and TLS 1.2 record encapsulation; particularly as shown in Figure 9-27. RFC 6347 specifies that DTLS 1.2 AEAD cipher suites operate identically the same as with TLS 1.2.

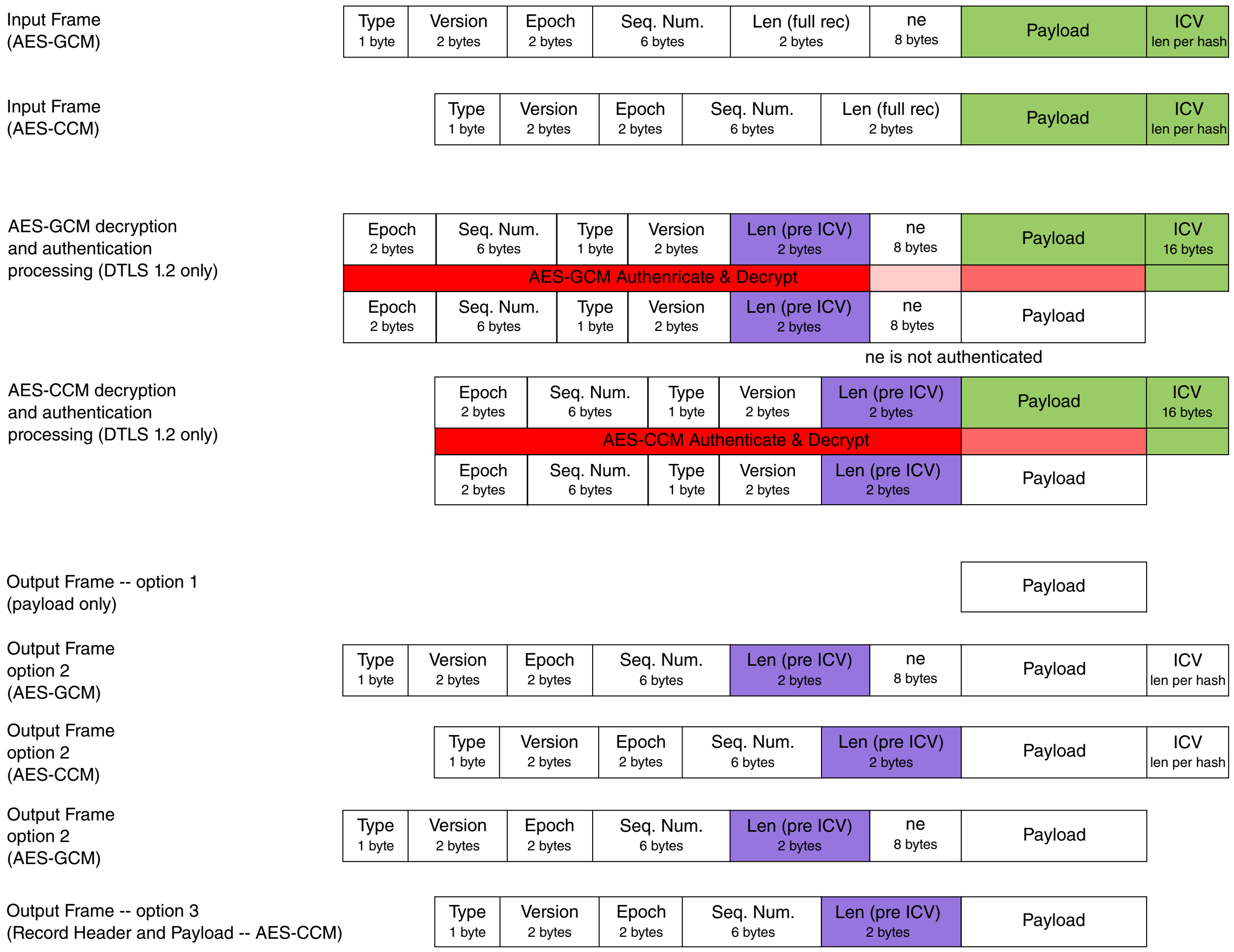

Input Frame (AES-GCM)
Type 1 byte | Version 2 bytes | Epoch 2 bytes | Seq. Num. 6 bytes | Len (full rec) 2 bytes | ne 8 bytes | Payload | ICV len per hash
Input Frame (AES-CCM)
Type 1 byte | Version 2 bytes | Epoch 2 bytes | Seq. Num. 6 bytes | Len (full rec) 2 bytes | Payload | ICV len per hash
AES-GCM decryption and authentication processing (DTLS 1.2 only)
Epoch 2 bytes | Seq. Num. 6 bytes | Type 1 byte | Version 2 bytes | Len (pre ICV) 2 bytes | ne 8 bytes | Payload | ICV 16 bytes
AES-GCM Authenricate & Decrypt
Epoch 2 bytes | Seq. Num. 6 bytes | Type 1 byte | Version 2 bytes | Len (pre ICV) 2 bytes | ne 8 bytes | Payload
ne is not authenticated
AES-CCM decryption and authentication processing (DTLS 1.2 only)
Epoch 2 bytes | Seq. Num. 6 bytes | Type 1 byte | Version 2 bytes | Len (pre ICV) 2 bytes | Payload | ICV 16 bytes
AES-CCM Authenticate & Decrypt
Epoch 2 bytes | Seq. Num. 6 bytes | Type 1 byte | Version 2 bytes | Len (pre ICV) 2 bytes | Payload
Output Frame -- option 1 (payload only)
Payload
Output Frame option 2 (AES-GCM)
Type 1 byte | Version 2 bytes | Epoch 2 bytes | Seq. Num. 6 bytes | Len (pre ICV) 2 bytes | ne 8 bytes | Payload | ICV len per hash
Output Frame option 2 (AES-CCM)
Type 1 byte | Version 2 bytes | Epoch 2 bytes | Seq. Num. 6 bytes | Len (pre ICV) 2 bytes | Payload | ICV len per hash
Output Frame option 2 (AES-GCM)
Type 1 byte | Version 2 bytes | Epoch 2 bytes | Seq. Num. 6 bytes | Len (pre ICV) 2 bytes | ne 8 bytes | Payload
Output Frame -- option 3 (Record Header and Payload -- AES-CCM)
Type 1 byte | Version 2 bytes | Epoch 2 bytes | Seq. Num. 6 bytes | Len (pre ICV) 2 bytes | Payload

**Figure 9-39. DTLS Record Decapsulation when using an AEAD Cipher**

The DTLS AEAD-cipher decapsulation procedure is as follows:

1. For AEAD ciphers, the authenticate-only segment consists the received record header, reordered to put epoch and sequence number first and is constructed like for TLS, and with the length field is adjusted to remove the length of the ICV. This modified record header is passed to the class 1 CHA as type AAD.
2. For AES-GCM, the nonce_explicit (ne) field is extracted from the input frame and is used to construct the nonce. Nonce_explicit is not authenticated.
3. If selected by the PDB options byte field outFMT, the record header is passed to the output frame in same order as received in the input frame, consisting of type, version, epoch, sequence number, the *adjusted* record length, and for AES-GCM, nonce_explicit.
4. The encrypted message is passed into the class 1 CHA with type set to message data. The ICV is passed into the class 1 CHA with type set to ICV . Decrypted payload is

written to the output frame. If outFMT is selected to do so, then SEC writes the ICV to the output frame.

## 9.3 SRTP packet encapsulation and decapsulation

SRTP, which stands for secure real-time transport protocol, is defined in RFC 3711 as a cryptographic encapsulation of RTP, which stands for real time protocol. RFC 3711 defines two ciphers and one authentication function for use in SRTP encapsulation and decapsulation. SEC supports the following:

- AES-Counter Mode for confidentiality
- SHA-1 for authentication.
- AEAD AES-GCM for confidentiality and authentication
- AEAD AES-CCM for confidentiality and authentication

SEC does not support AES-f8, which is the remaining cipher specified by RFC 3711.

SEC's built-in SRTP protocol supports data encapsulation, encryption, and data integrity checking. The PROTINFO field codes enumerated in Table 7-54 define the specific encryption and data integrity algorithms to be used by the protocol, and the hardware handles the remaining details.

**Table 9-51. SRTP protocol descriptors**

| Encapsulation | | Decapsulation |
|---|---|---|
| Header | | Header |
| Protocol data block | | Protocol data block |
| Class 2 key data block | | Class 2 key data block |
| Class 1 key data block | | Class 1 key data block |
| Protocol = SRTP encrypt | | Protocol = SRTP decrypt |

**NOTE**

Any bulk-data protocol using a cipher suite that includes any HMAC uses MDHA and for performance requires the use of a split key. Therefore for proper operation when using IPsec with HMAC, the KDEST field in the Class 2 KEY command must be set to MDHA Split Key. For first invocation, the Derived Key Protocol may be used to create both the split key form of the HMAC key as well as the actual key command loading the split key.

## 9.3.1 Building the initial counter value (Counter IV)

The first 16 to 80 bytes of an RTP packet consists of a series of fields that are authenticated, but not encrypted, by the encapsulation process. This series of fields is referred to as the SRTP header.

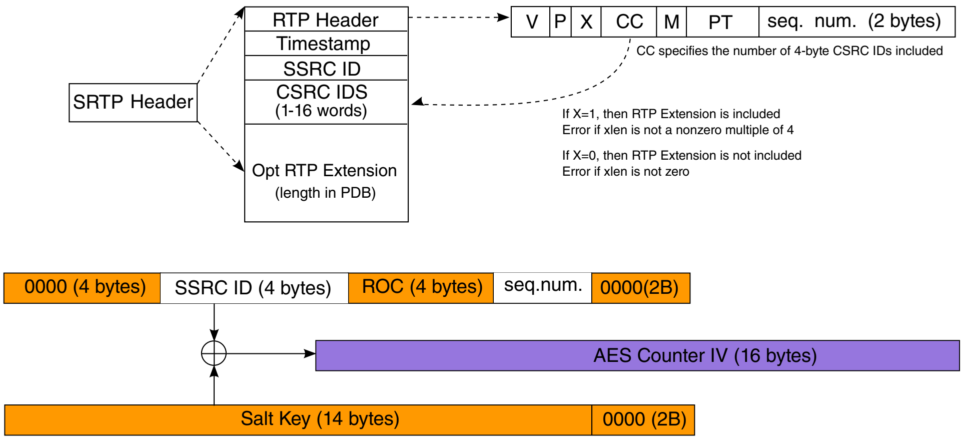

SRTP Header
RTP Header
Timestamp
SSRC ID
CSRC IDS
(1-16 words)
Opt RTP Extension
(length in PDB)
V
P
X
CC
M
PT
seq. num. (2 bytes)
CC specifies the number of 4-byte CSRC IDs included
If X=1, then RTP Extension is included
Error if xlen is not a nonzero multiple of 4
If X=0, then RTP Extension is not included
Error if xlen is not zero
0000 (4 bytes)
SSRC ID (4 bytes)
ROC (4 bytes)
seq.num.
0000(2B)
AES Counter IV (16 bytes)
Salt Key (14 bytes)
0000 (2B)

**Figure 9-40. SRTP encapsulation and decapsulation AES Counter IV Preparation**

AES counter mode requires an initial counter value (Counter IV). SRTP specifies that the initial counter value is obtained by performing a 112-bit bitwise XOR function of the Salt key (14 bytes of extra key material) with a concatenation of the following three fields (found on the input frame): the 4-byte SSRC ID, the 2-byte sequence number, and the 4-byte ROC.

- The SSRC ID is found in bytes 9-13 of the SRTP Header.
- The sequence number is found in bytes 2-3 of the SRTP Header.
- The rollover counter (ROC) is an RTP parameter that is incremented each time the sequence number rolls over; each RTP packet is generated with a monotonically increasing sequence number).

## 9.3.2 Building the AEAD Nonce

Both AEAD cipher suites supported by SRTP require the construction of a 12-byte Nonce. AES-GCM uses the 12-byte nonce as the 12-byte GCM-IV; effectively becoming the initial counter value for encrypting the packet. AES-CCM requires construction of 16-byte words $B_0$ and Initial Counter from the 12-byte Nonce.

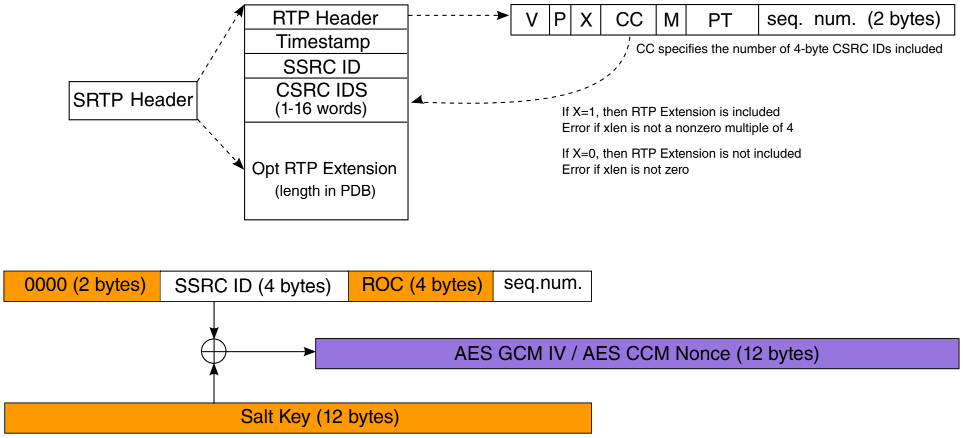

SRTP Header
RTP Header
Timestamp
SSRC ID
CSRC IDS
(1-16 words)
Opt RTP Extension
(length in PDB)
V
P
X
CC
M
PT
seq. num. (2 bytes)
CC specifies the number of 4-byte CSRC IDs included
If X=1, then RTP Extension is included
Error if xlen is not a nonzero multiple of 4
If X=0, then RTP Extension is not included
Error if xlen is not zero
0000 (2 bytes)
SSRC ID (4 bytes)
ROC (4 bytes)
seq.num.
AES GCM IV / AES CCM Nonce (12 bytes)
Salt Key (12 bytes)

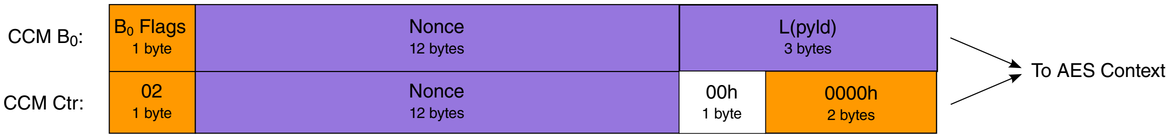
**Figure 9-41. SRTP AEAD Nonce Preparation**

The Nonce is constructed by taking the 12-byte Salt Key from the PDB, and XORing that value with a 12-byte value constructed from the 4-byte SSRC from the input frame, the 4-byte ROC from the PDB, and the 2-byte Sequence Number from the input frame (the two most significant bytes are treated as zero to construct a 12-byte value).

- The SSRC ID is found in bytes 9-13 of the SRTP Header.
- The sequence number is found in bytes 2-3 of the SRTP Header.
- The rollover counter (ROC) is an RTP parameter that is incremented each time the sequence number rolls over; each RTP packet is generated with a monotonically increasing sequence number).

## 9.3.3 Constructing the AESA context from the SRTP AEAD Nonce for AES-CCM mode

This figure shows the construction of SRTP AESA context for AES-CCM mode.



Note: $B_0$ Flags to be programmed into PDB word 6 field $B_0$ Flags
use 5A with 8-byte ICV, 6A with 12-byte ICV, and 7A with 16-byte ICV
02 to be programmed into PDB word 6 field $CTR_0$ Flags
0000h to be programmed into PDB word 6 field $CTR_0$ Constant

**Figure 9-42. SRTP CCM Context construction**

SEC uses the nonce to construct both the CCM Initial Counter value and CCM $B_0$, both of which are written to the Class 1 CHA Context Register.

- The CCM Initial Counter value includes 3 bytes extracted from the protocol data block.
- The CCM $B_0$ includes one byte extracted from the protocol data block and the length of the payload (as determined by SEC).

## 9.3.4 SRTP encapsulation

SEC interprets the input frame as:

- A 16-80 byte SRTP Header
- An arbitrary-length RTP payload
- RTP padding that ensures the packet ends on a 4-byte boundary; therefore SEC does not perform padding for SRTP encapsulation.
- A pad length field indicating how many bytes after payload is padding.

Note that the SRTP header contains the following:

- 4-byte RTP header, consisting of a field indicating the number of CSRC IDs included (CC) and an RTP sequence number
- 4-byte timestamp field
- SSRC ID field
- Field from 1 to 16 CSRC IDs, each 4 bytes
- Optional RTP extension header

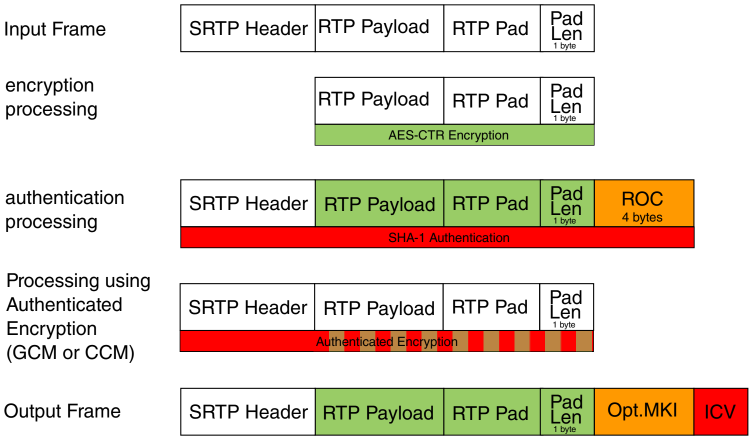


**Figure 9-43. SRTP encapsulation process**

SRTP Header
RTP Payload
RTP Pad
Pad Len

**Figure 9-44. SRTP encapsulation input frame**

### 9.3.4.1 Process for SRTP encapsulation

This figure shows SRTP encapsulation with AES Counter encryption and HMAC-SHA-1 authentication.

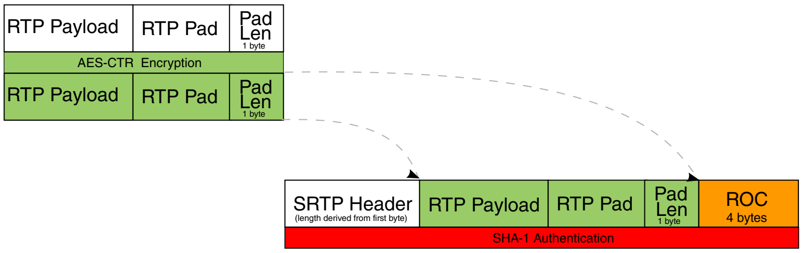


**Figure 9-45. SRTP encapsulation encryption and authentication when using AES-Counter and HMAC-SHA-1**

1. To begin encapsulation for the AES-Counter / HMAC-SHA-1 cipher suite, SEC builds the Counter IV as described in Building the initial counter value (Counter IV).
2. To begin encapsulation for AEAD cipher suites, SEC builds the Nonce as described in Building the AEAD Nonce.
3. SEC processes the input frame as follows:
   - SEC passes the SRTP header to the Class 2 CHA for HMAC-SHA-1 authentication and then to the output frame.
   - SEC passes the RTP payload, padding, and pad length fields to the Class 1 CHA for AES-Counter encryption.
   - For AEAD Cipher Suites, SEC passes the RTP payload, padding, and pad length fields to the Class 1 CHA for Authenticated Encryption. The SRTP header is processed as AAD, and the RTP Payload, RTP Pad, and Pad Length are processed as plaintext. Note that the ROC is not included as part of AAD for Authenticated Encryption. Instead, the ROC is used in Nonce formation.
   - The encrypted result is passed to the output frame immediately following the SRTP header and then to the Class 2 CHA for authentication.
4. When HMAC-SHA-1 is used, the ROC field is the last item authenticated although it appears first in the PDB; it is not passed to the output frame. As noted above, for AEAD encapsulation the ROC is not treated as data for any authentication computation and instead is part of the IV.
5. ROC is incremented whenever the Seq Num in the SRTP Header rolls over (so after use, ROC should be incremented and written back to PDB if the Seq Num value is FFFF).
6. Once authentication completes, the number of bytes of ICV selected is passed to the output frame. For HMAC-SHA-1, the selection is made by the PDB n_tag field. For AEAD cipher suites, the ICV size is defined by the chosen cipher suite.

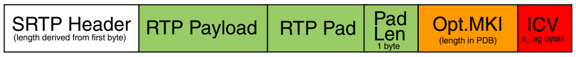


**Figure 9-46. SRTP encapsulation output frame**

### 9.3.4.2 Handling the optional MKI

If present, the optional MKI is copied from the PDB to the output frame immediately following the encrypted pad length and prior to the ICV. However as MKI is not authenticated, it is not copied to the input-data FIFO. The length of the MKI (in 4-byte words) is also stored in the PDB. Note that the length of MKI supported by SEC is limited by the total size of the descriptor buffer as well as the size of the other contents required to be stored in the descriptor buffer.

### 9.3.4.3 SRTP encapsulation PDB format descriptions

**Table 9-52. SRTP encapsulation PDB, formats for AES-CTR, AES-CCM and AES-GCM**

<table>
<tr><td>PDB word 0</td><td>(8 bits)<br>x-len</td><td>(8 bits)<br>length of MKI</td><td>(8 bits)<br>for AES-CTR: n_tag<br>otherwise: reserved (00h)</td><td>(8 bits)<br>options<br>[see table below]</td></tr>
<tr><td>PDB word 1</td><td colspan="2">(16 bits)<br>for AES-CTR: constant=0000h<br>otherwise: constant</td><td colspan="2">(16 bits)<br>for AES-CTR: constant=0000h<br>otherwise: reserved (0000h)</td></tr>
<tr><td>PDB word 2</td><td colspan="2">(16 bits)<br>reserved (0000h)</td><td colspan="2">(16 bits)<br>for AES-CTR: constant=0000h<br>otherwise: reserved (0000h)</td></tr>
<tr><td>PDB word 3</td><td colspan="4">salt 1</td></tr>
<tr><td>PDB word 4</td><td colspan="4">salt 2</td></tr>
<tr><td>PDB word 5</td><td colspan="4">salt 3</td></tr>
<tr><td rowspan="2">PDB word 6</td><td colspan="2">(16 bits)<br>for AES-CTR: salt 4<br>for AES-GCM: reserved (0000h)<br>for AES-CCM:</td><td colspan="2" rowspan="2">(16 bits)<br>for AES-CTR: constant=0000h<br>for AES-GCM: reserved (0000h)<br>for AES-CCM: $Ctr_0$ constant</td></tr>
<tr><td>$B_0$ flags (8 bits)</td><td>$Ctr_0$ flags (8 bits)</td></tr>
<tr><td>PDB word 7</td><td colspan="4">reserved (00000000h)</td></tr>
<tr><td>PDB word 8[1]</td><td colspan="4">ROC</td></tr>
<tr><td>PDB word 9</td><td colspan="4">optional MKI</td></tr>
</table>

[1] Written back to PDB in memory, as needed.

Table 9-53. SRTP encapsulation PDB, format of the options byte

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | MKI | Reserved | | |

Table 9-54. SRTP encapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| 7-4 | Reserved |
| 3<br>MKI | MKI included in Output Frame<br>0b - MKI not included in Output Frame.<br>1b - MKI copied from PDB into Output Frame |
| 2-0 | Reserved |

### 9.3.4.4 SRTP encapsulation error conditions

This table lists the conditions under which SRTP encapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine.

Table 9-55. SRTP encapsulation error conditions

| Condition | Error status |
|---|---|
| Reserved bit set to 1 in the PDB options byte | Protocol PDB error |
| OPERATION Command PROT ID selects SRTP Encap, and PROTINFO is not a valid protocol | Protocol Command Error |
| n_tag in the PDB = 0, or n_tag > 20 and cipher suite is AES-Counter with HMAC-SHA-1 | Protocol Command Error |
| [ROC, SEQNUM] overflows | Protocol Sequence Number Overflow |

## 9.3.5 SRTP decapsulation overview

This figure shows SRTP decapsulation.

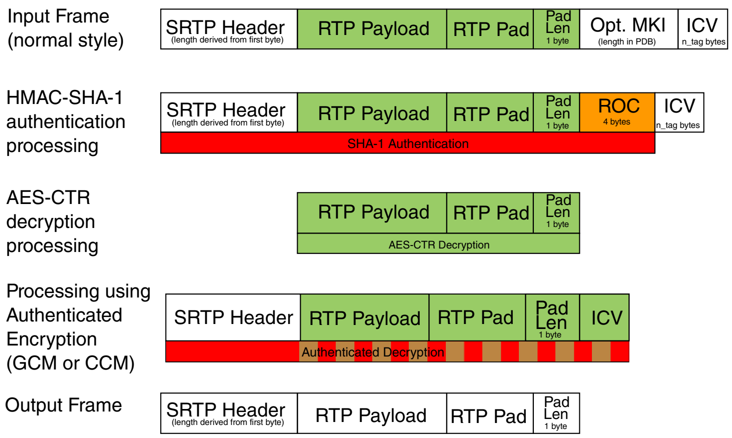


**Figure 9-47. SRTP decapsulation overview**

To perform decapsulation, SEC receives an input frame interpreted similarly to the encapsulation input frame, with the main differences being the inclusion of an ICV field and an optional MKI field. If present, the optional MKI is located after the padding length and before the ICV.

### 9.3.5.1 Process for SRTP decapsulation

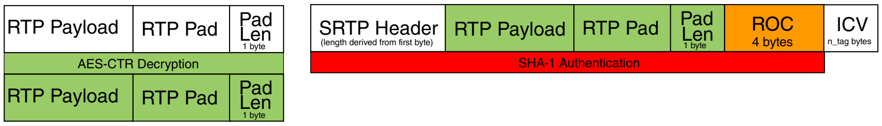


**Figure 9-48. SRTP decapsulation decryption and authentication with AES-Counter and HMAC-SHA-1**

The SRTP decapsulation procedure for the AES-Counter / HMAC-SHA-1 cipher suite is as follows:

1. SEC grabs the appropriate fields from the SRTP header to generate the Counter IV as described in Building the initial counter value (Counter IV).
2. SEC pushes the SRTP header into the Class 2 CHA for authentication and then onto the output frame.
3. The encrypted payload and padding are passed from the input frame to both CHAs for decryption and authentication.
4. The decrypted results are passed to the output frame.
5. The ROC is pushed into the Class 2 CHA for authentication; if the MKI, is present on the input frame, it is ignored.

6. MDHA completes computation of the ICV across the input frame and compares the result to that received from the input frame. If any difference is detected, a failure is reported.

The SRTP decapsulation procedure for Authenticated Encryption cipher suites is as follows:

1. SEC constructs the AEAD Nonce as described in Building the AEAD Nonce, using SSRC and Sequence Number from the input frame, and ROC and Salt Key from the protocol data block.
2. For AES-GCM cipher suites, the nonce is passed to AESA as IV
3. For AES-CCM cipher suites, the nonce is used to construct $B_0$ and $Ctr_0$, which are then written to Class 1 context register
4. The received SRTP header is passed to AESA as AAD
5. The encrypted part of the received frame is passed to AESA as message data
6. MKI, if present, is stripped
7. ICV is passed to AESA as ICV type, to be compared with the computed ICV

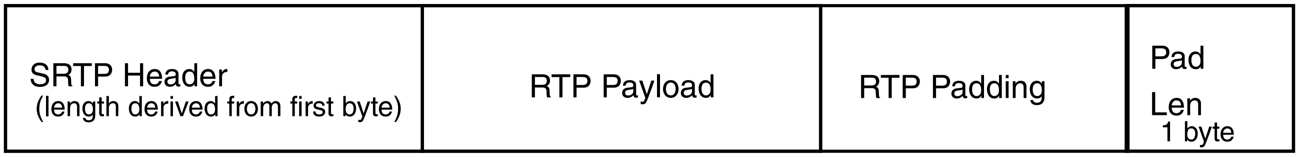


**Figure 9-49. SRTP decapsulation output frame**

SEC can perform anti-replay checking for SRTP decapsulation, using a window of 64 or 128 packets. A replayed or late PDU is rejected, and the Job Completion Status Word written to the Output Frame Queue is tagged as REPLAY or LATE as appropriate.

SEC does not update the PDB with the anti-replay status until after the ICV check has passed. If the ICV check fails, the PDU is rejected and the anti-replay status in the PDB is not updated. SEC manages a local copy of the sequence number found in the SRTP Header, as well as the ROC. The ROC needs to be incremented whenever the sequence number rolls over; sometimes ROC needs to be adjusted prior to use.

### 9.3.5.2 SRTP decapsulation PDB format descriptions

**Table 9-56. SRTP decapsulation PDB, formats for AES-CTR, AES-CCM and AES-GCM**

| | | | | |
|---|---|---|---|---|
| PDB word 0 | (8 bits)<br>x-len | (8 bits)<br>length of MKI | (8 bits)<br>for AES-CTR: n_tag<br>otherwise: reserved (00h) | (8 bits)<br>options<br>[see table below] |

*Table continues on the next page...*

### Table 9-56. SRTP decapsulation PDB, formats for AES-CTR, AES-CCM and AES-GCM (continued)

<table>
<tr><td>PDB word 1</td><td colspan="2">(16 bits)<br>for AES-CTR: constant=0000h<br>otherwise: constant</td><td>(16 bits)<br>for AES-CTR: constant=0000h<br>otherwise: reserved (0000h)</td></tr>
<tr><td>PDB word 2</td><td colspan="2">(16 bits)<br>reserved (0000h)</td><td>(16 bits)<br>for AES-CTR: constant=0000h<br>otherwise: reserved (0000h)</td></tr>
<tr><td>PDB word 3</td><td colspan="3">salt 1</td></tr>
<tr><td>PDB word 4</td><td colspan="3">salt 2</td></tr>
<tr><td>PDB word 5</td><td colspan="3">salt 3</td></tr>
<tr><td rowspan="2">PDB word 6</td><td colspan="2">(16 bits)<br>for AES-CTR: salt 4<br>for AES-GCM: reserved (0000h)<br>for AES-CCM:</td><td rowspan="2">(16 bits)<br>for AES-CTR: constant=0000h<br>for AES-GCM: reserved (0000h)<br>for AES-CCM: $Ctr_0$ constant</td></tr>
<tr><td>$B_0$ flags (8 bits)</td><td>$Ctr_0$ flags (8 bits)</td></tr>
<tr><td>PDB word 7 [1]</td><td colspan="2">(16 bits) reserved</td><td>(16 bits) sequence number</td></tr>
<tr><td>PDB word 8</td><td colspan="3">ROC</td></tr>
<tr><td>PDB word 9</td><td colspan="3">anti-replay scorecard 1 [present if ARS= 01b or 10b]</td></tr>
<tr><td>PDB word 10</td><td colspan="3">anti-replay scorecard 2 [present if ARS= 01b or 10b]</td></tr>
<tr><td>PDB word 11</td><td colspan="3">anti-replay scorecard 3 [present if ARS= 10b]</td></tr>
<tr><td>PDB word 12</td><td colspan="3">anti-replay scorecard 4 [present if ARS= 10b]</td></tr>
</table>

[1] Shaded rows are written back to PDB in memory, as needed.

### Table 9-57. SRTP decapsulation PDB, format of the options byte

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ARS | | Reserved | | MKI | Reserved | | |

### Table 9-58. SRTP decapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| 7-6<br>ARS | anti-replay checking scorecard<br>00b - Anti-replay checking disabled<br>01b - 64-bit anti-replay checking enabled<br>10b - 128-bit anti-replay checking enabled<br>11b - Reserved |
| 5-4 | Reserved |
| 3<br>MKI | MKI included in Output Frame<br>0b - MKI not included in Output Frame.<br>1b - MKI copied from PDB into Output Frame |

*Table continues on the next page...*

Table 9-58. SRTP decapsulation PDB, description of the options byte (continued)

| Field | Description |
|---|---|
| 2-0 | Reserved. Must be zero. |

### 9.3.5.3 SRTP decapsulation error conditions

This table lists the conditions under which SRTP decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine. Authentication failure can produce an ICV check error.

Table 9-59. SRTP decapsulation error conditions

| Condition | Error Status |
|---|---|
| Reserved bit set to 1 in the PDB options byte | Protocol PDB error |
| OPERATION Command PROT ID selects SRTP Decap, and PROTINFO is not a valid protocol | Protocol Command Error |
| n_tag in the PDB = 0, or n_tag > 20 for HMAC-SHA-1 cipher suites | Protocol Command Error |
| [ROC, SEQNUM] overflows | Protocol Sequence Number Overflow |
| Anti-Replay detects a LATE packet | Protocol LATE error |
| Anti-Replay detects a REPLAY packet | Protocol REPLAY error |

## 9.4 IEEE 802.1AE MACsec encapsulation and decapsulation overview

SEC supports MACsec encapsulation and decapsulation as described in the IEEE 802.1AE-2006 specification and the IEEE 802.1AEbn-2011 and 802.1AEbw-2013 amendments, using AES-GCM for authentication and confidentiality.

SEC optionally supports the following:

- CRC generation and insertion of the resulting frame check sequence (FCS).
- Insertion of an optional AAD (up to 96 bytes for encapsulation and up to 112 bytes for decapsulation) that is not part of the current IEEE 802.1AE specification.

Both options are controlled by the descriptor's operation command (see PROTOCOL OPERATION commands).

Table 9-60. IEEE 802.1AE MACsec protocol descriptors

| Encapsulation | | Decapsulation |
|---|---|---|
| Header | | Header |
| Protocol data block | | Protocol data block |
| Class 1 key data block | | Class 1 key data block |
| Protocol = MACsec encrypt | | Class 2 key data block |
| | | Protocol = MACsec decrypt |

## 9.4.1 Process for 802.1AE MACsec encapsulation

This figure shows 802.1AE MACsec encapsulation procedure.

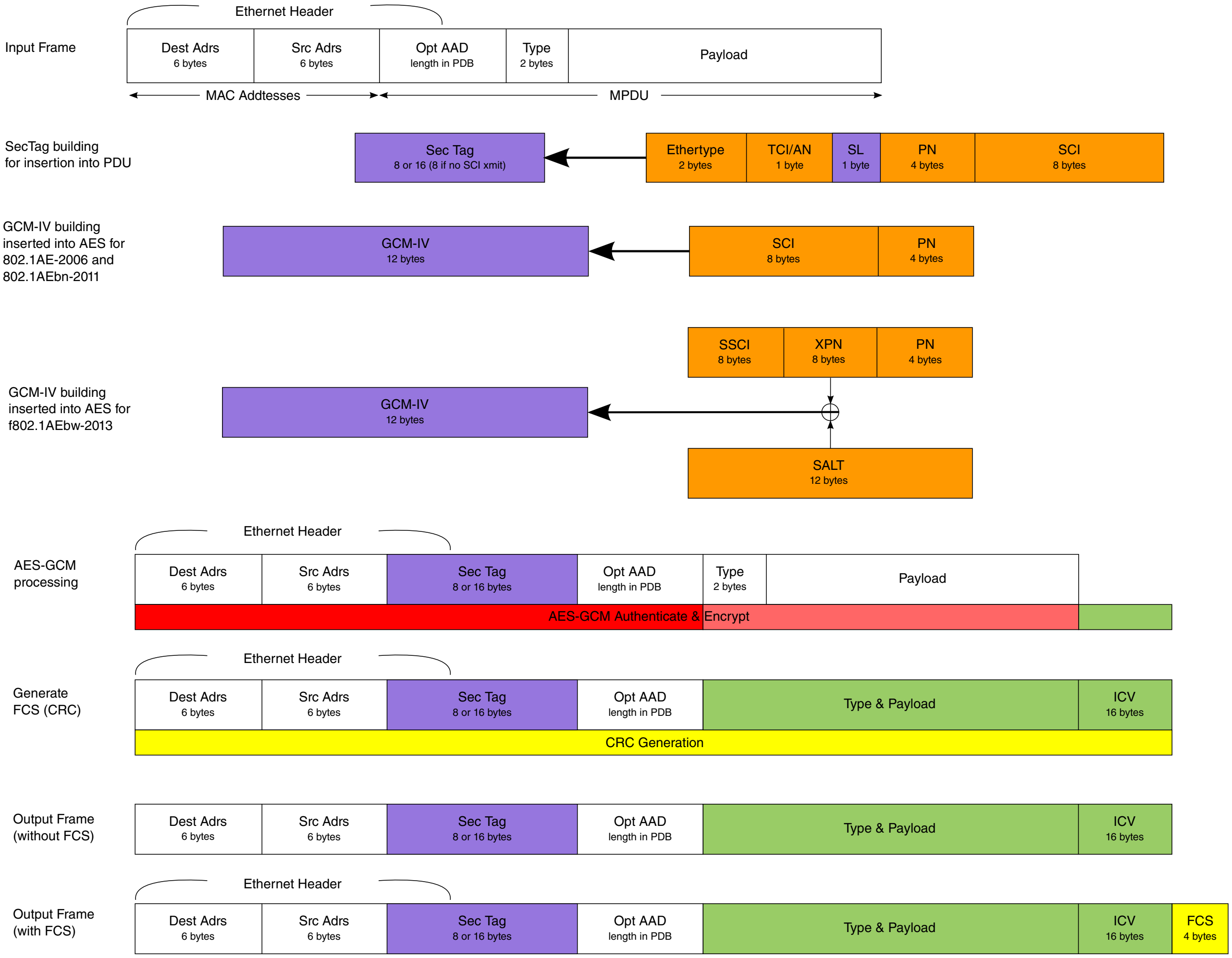


Figure 9-50. 802.1AE MACsec encapsulation

This figure shows the encapsulation input frame.

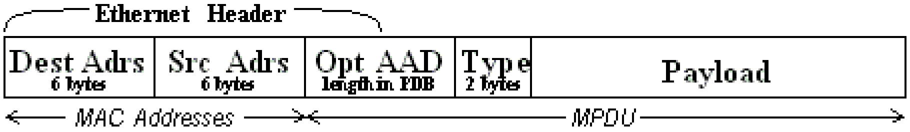


**Figure 9-51. 802.1AE MACsec encapsulation input frame**

The standard MACsec encapsulation procedure is:

1. SEC receives an Ethernet header from the input frame; the header consists of:
   - A destination address (Dest Adrs)
   - A source address (Src Adrs)
   - Optional AAD (of up to 96 bytes)

### NOTE

> In Figure 9-51, a second 2-byte ethertype field is assumed to part of the optional AAD because during the encapsulation process a new ethertype field (assumed part of the SecTag) is inserted into the Ethernet header.

   - An ethertype field that refers to the type of the frame (Type)
2. SEC receives the payload from the input frame.
3. The destination address and source address portions of the Ethernet header are copied to both AESA and CRCA, tagged for authentication (AAD) and CRC (if enabled), and also copied to the output frame.
4. SEC builds the 8 or 16-byte SecTag as shown in the following figure:

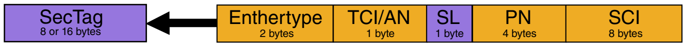


**Figure 9-52. Building the MACsec SecTag**

   a. SCI transmission is determined by bit 1 of the TCI/AN byte.
      - If set, the 16-byte SecTag includes the SCI.
      - If not set, the 8-byte SecTag omits the SCI.

   b. The 4-byte PN, or packet number, is incremented and written back to one word of the PDB.

   c. SEC computes the 1-byte SL as the length of payload if the payload is less than 48 bytes in length and as zero otherwise.

   d. EtherType and TCI/AN occupy one word of the PDB.
5. (optional) Rollover results in a ROLLOVER status indication in the Job Completion Status Word.
6. The SecTag is pushed onto output frame and into both CHAs, where it is tagged for authentication and CRC.

7. SEC constructs the GCM-IV from the SCI and PN for AES-GCM, or from the SSCI, XPN, PN, and SALT for AES-GCM-XPN; note that SCI is used to construct the GCM-IV even if bit 1 of the TCI/AN byte indicates that SCI is not transmitted.

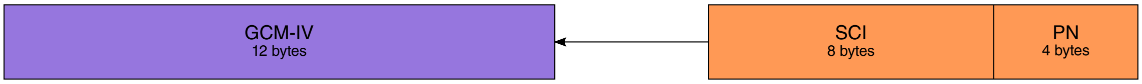


**Figure 9-53. Building the GCM initialization vector (GCM IV) for AES-GCM cipher suites**

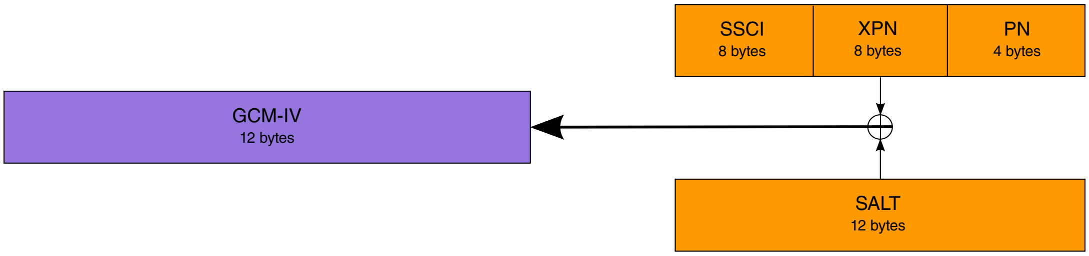


**Figure 9-54. Building the GCM initialization vector (GCM IV) for AES-GCM-XPN cipher suites**

8. SEC pads the GCM IV with trailing zeros and pushes it to AESA (tagged as IV) prior to AAD, ethertype, and payload.
9. SEC treats the two byte ethertype field as if it were part of payload and pushes the payload into AESA for encryption and authentication, and into CRCA.
10. The resulting encrypted payload (including the encrypted type) is pushed onto the output frame.
11. AESA computes the GCM ICV automatically, and that encrypted ICV is also pushed onto the output frame.

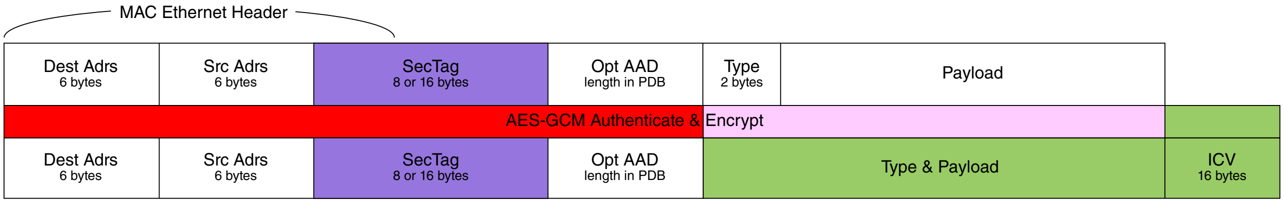


**Figure 9-55. 802.1AE MACsec packet encapsulation using AES-GCM**

### 9.4.1.1 Using the frame check sequence (FCS)

SEC's in-built CRCA CHA can be enabled to compute the frame check sequence (FCS). IEEE 802.3 (Ethernet) specifies a minimum frame size of 64 bytes, and the last four bytes of the frame are the FCS.

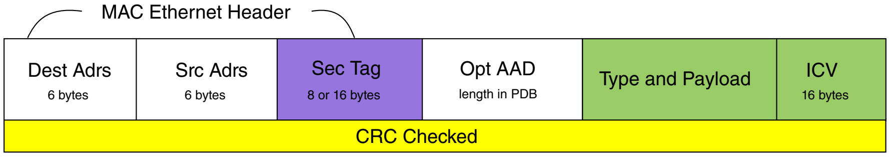


**Figure 9-56. Optional 802.1AE MACsec packet encapsulation using CRC**

If the FCS is enabled, the CRCA receives all data authenticated. If the output frame is less than 60 bytes, SEC pads the output frame with zeros to bring the frame size to 60 bytes. These additional bytes of zero padding are input to the CRC calculation.

Additionally, the ICV computed by AESA is recirculated back as a final input to Class 2 CHA CRCA. The FCS produced is then a CRC of the entire authenticated and encrypted packet, as shown in Figure 9-57. Figure 9-58 shows the result of appending the FCS to the end of the encapsulated packet.

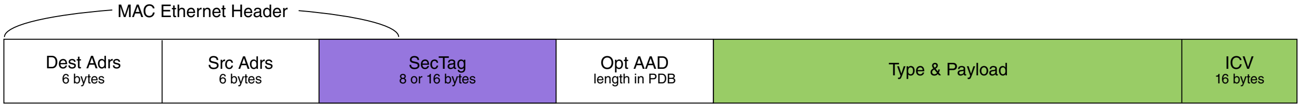


**Figure 9-57. 802.1AE MACsec encapsulation output packet option 1 (no FCS)**

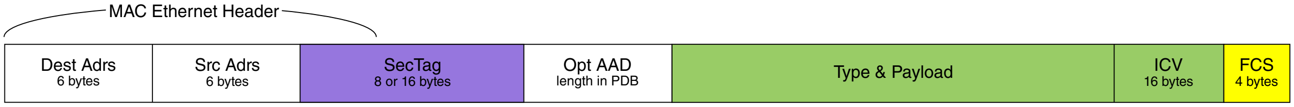


**Figure 9-58. 802.1AE MACsec encapsulation output packet option 2 (with FCS)**

**NOTE**

At this point, the Ethernet header consists of the destination address, the source address, and the ethertype that is part of the SecTag.

### 9.4.1.2 Additional notes for GMAC support

Process for 802.1AE MACsec encapsulation describes MACsec with GCM. SEC also supports GMAC. MACsec for GMAC is processed like GCM except that the input frame is not encrypted. The ICV is computed in the same fashion.

SEC determines whether to use GCM or GMAC based on the E bit in the TCI field of the SecTag programmed into the Shared Descriptor PDB. The TCI field is found in the next-to-the-rightmost byte of word 4 of the PDB. IEEE standard 802.1AE numbers the TCI octet with msb 8 and lsb 1, and identifes E as bit 4. Therefore, the E bit is programmed into ms bit of the third-from-the-rightmost nibble of word 4 of the PDB.

## 9.4.2 MACsec encapsulation PDB format descriptions

**Table 9-61. MACsec Encapsulation PDB for AES-GCM cipher suites**

| | | | |
|---|---|---|---|
| PDB word 0 | AAD Length<br>(16 bits) | reserved<br>(8 bits) | options<br>(8 bits)<br>[see table below] |
| PDB word 1 | SCI 1 | | |
| PDB word 2 | SCI 2 | | |
| PDB word 3 | Ethertype<br>(16 bits) | TCI/AN<br>(8 bits) | reserved<br>(8 bits) |
| PDB word 4 | PN | | |

**Table 9-62. MACsec Encapsulation PDB for AES-GCM-XPN cipher suites**

| | | | |
|---|---|---|---|
| PDB word 0 | AAD Length<br>(16 bits) | reserved<br>(8 bits) | options<br>(8 bits)<br>[see table below] |
| PDB word 1 | SCI 1 | | |
| PDB word 2 | SCI 2 | | |
| PDB word 3 | Ethertype<br>(16 bits) | TCI/AN<br>(8 bits) | reserved<br>(8 bits) |
| PDB word 4 | salt 1 | | |
| PDB word 5 | salt 2 | | |
| PDB word 6 | salt 3 | | |
| PDB word 7 | EPN | | |
| PDB word 8 | PN | | |
| PDB word 9 | SSCI | | |

**Table 9-63. MACsec encapsulation PDB, format of the options byte**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | SAinSCI | Reserved | Reserved | Reserved | Reserved | FCS |

### Table 9-64. MACsec Encapsulation PDB - Description of the Options Byte

| Field | Description |
|---|---|
| 7-6 | Reserved |
| 5<br>SAinSCI | 0: SCI used as-is from PDB for encapsulation<br>1: SCI for encapsulation constructed from SA from input frame, with portID from lower 16 bits of SCI2 in PDB |
| 4-1 | Reserved |
| 0<br>FCS | FCS included in Output Frame<br>0 FCS not included in Output Frame.<br>1 FCS computed by CRCA and copied into Output Frame |

## 9.4.3 Process for 802.1AE MACSec decapsulation

This figure shows the stages for the 802.1AE MACsec decapsulation process.

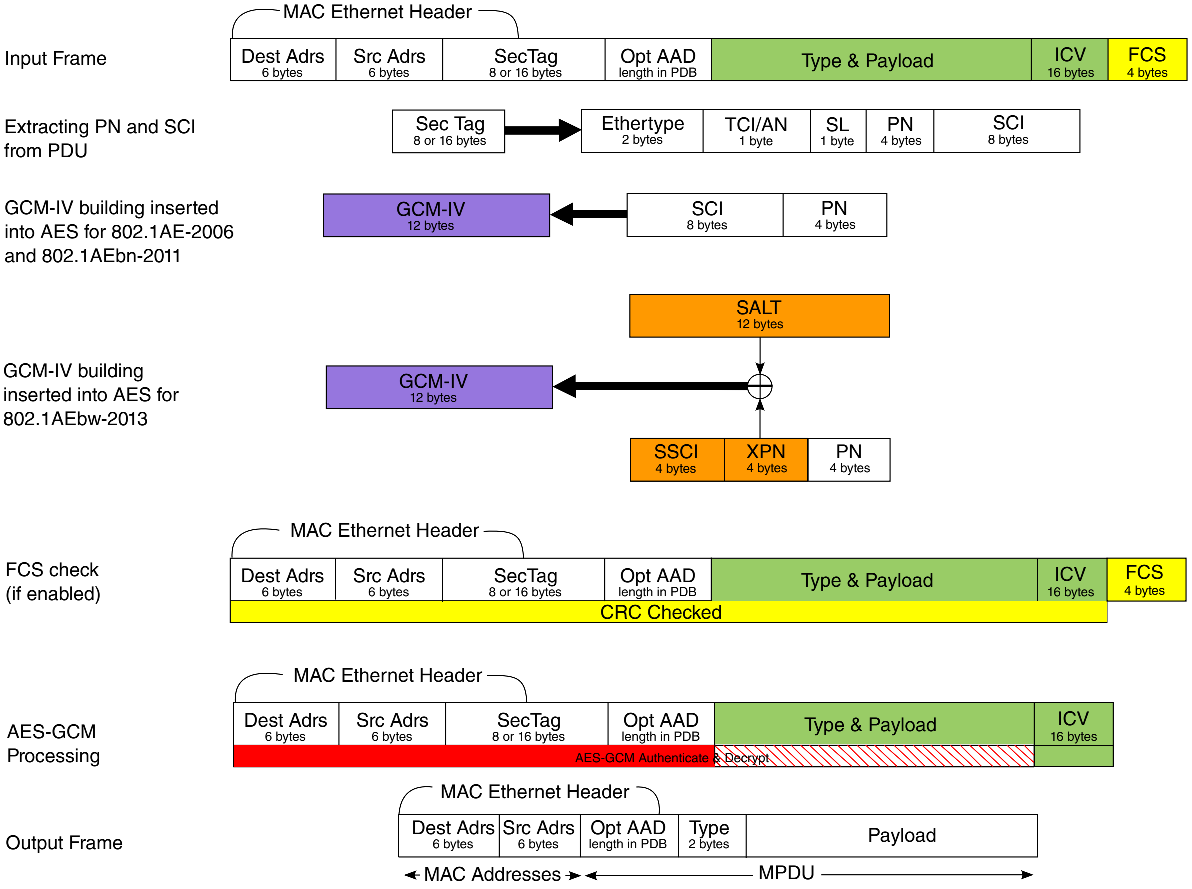


**Figure 9-59. 802.1AE MACsec decapsulation process**

SEC decapsulates MACsec encapsulated packets as specified by the IEEE 802.1AE specification, with the addition of support for an optional AAD value. This optional AAD is specified to sit in the frame between the SecTag and the encrypted type & payload field (see the following figure).

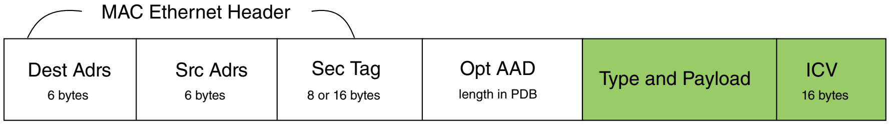


**Figure 9-60. 802.1AE MACsec decapsulation input frame**

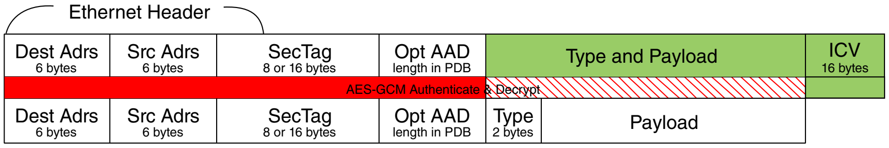


**Figure 9-61. 802.1AE MACsec decapsulation**

The procedure for MACsec decapsulation is:

1. SEC pulls the MAC Ethernet header and the SecTag from the input frame and pushes them into the AESA tagged for authentication.
2. The destination address and source address fields of the MAC Ethernet header are also copied to the output frame.
3. (optional) If present, the optional AAD is tagged for authentication by AESA.
4. For AES-GCM cipher suites, SCI and PN are pulled out of the SecTag to form the GCM-IV, as shown in the following figures.

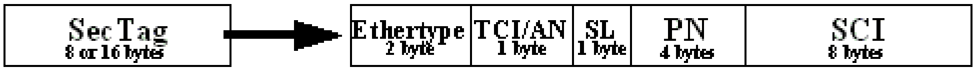


**Figure 9-62. 802.1AE MACsec SecTag elements**

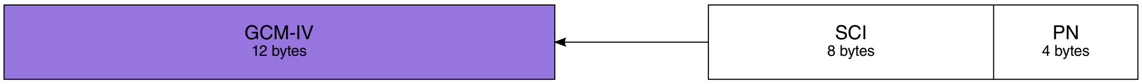


**Figure 9-63. 802.1AE MACsec GCM IV building**

**NOTE**

If SCI is not transmitted (this is indicated by bit 1 not set in TCI/AN byte of SecTag), the GCM-IV must be built using a SCI extracted from the PDB.

5. For the GCM-XPN cipher suites, PN is pulled out of the received frame's SecTag and combined with SSCI, XPN and SALT as programmed into the PDB to form the GCM-IV, as shown in the following figures.

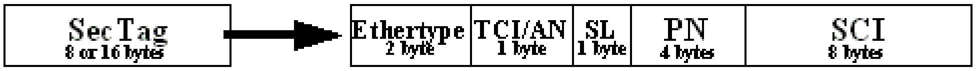


**Figure 9-64. 802.1AE MACsec SecTag elements**

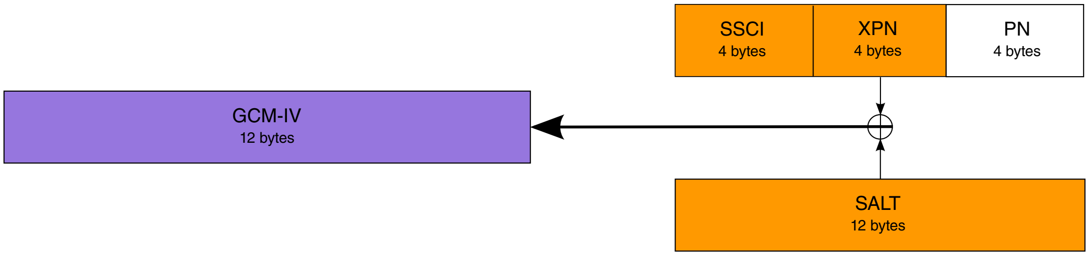


**Figure 9-65. 802.1AE MACsec GCM-XPN IV building**

6. The GCM-IV is padded with trailing zeros to a 16-byte boundary and pushed to AESA.
7. (optional) If present, the optional AAD is also pushed to the output frame.
8. The encrypted type and payload are pushed into AESA and tagged for authentication and decryption.
9. The decrypted result is pushed onto the output frame.
10. SEC compares the received ICV to the computed ICV; if it detects differences, it indicates ICV CHECK FAIL in the Job Completion Status Word.

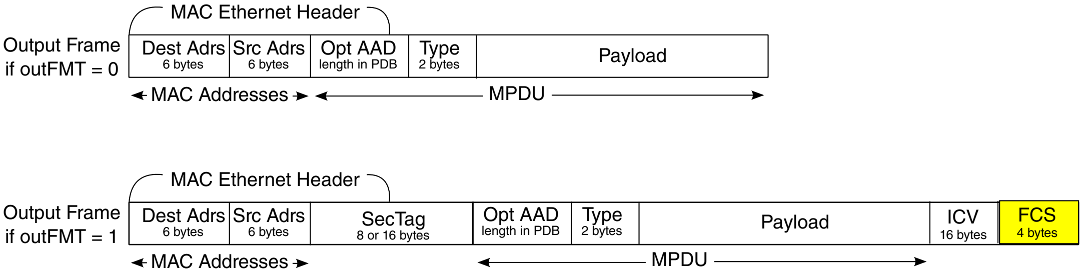


**Figure 9-66. 802.1AE MACsec Decapsulation Output Frame**

### 9.4.3.1 Automatically switching between two keys

SEC MACsec decapsulation has the ability to automatically switch between two keys. If AKS (Automatic Key Switching) is set in the PDB, SEC uses the least significant bit of the AN field of the SecTag received in the input frame to select which key is used:

- If the bit is set, the Class 2 Key is used.
- If the bit is cleared, the Class 1 Key is used.

This feature allows reception of frames without software needing to preprocess the frames to determine which key is being used.

### 9.4.3.2 Additional notes for GMAC support (decapsulation)

Process for 802.1AE MACSec decapsulation describes MACsec with GCM. SEC also supports GMAC. MACsec for GMAC is processed like GCM, except that the input frame is not encrypted. The same ICV is computed in the same fashion.

During MACsec decapsulation, SEC determines whether to use GCM or GMAC based on the E bit in the TCI field of the SecTag received in the input frame. The TCI field is found in bits 16-23 of word 4 of the PDB. IEEE standard 802.1AE numbers the TCI octet with msb 8 and lsb 1 and identifes E as bit 4.

## 9.4.4 MACsec decapsulation PDB format descriptions

Table 9-65. MACsec decapsulation PDB

| | Descriptor Header (1 or 2 words) | | | |
|---|---|---|---|---|
| PDB word 0 | AAD Length<br>(16 bits) | reserved<br>(8 bits) | Options<br>(8 bits)<br>[see table below] | |
| PDB word 1 | SCI 1 | | | |
| PDB word 2 | SCI 2 | | | |
| PDB word 3 | reserved | | ARlen<br>(8 bits) | |
| PDB word 4 | PN | | | DECO updates PDB in descriptor buffer and external memory as needed |
| PDB word 5 | Anti Replay Scorecard 1 [present if AR=1] | | | |
| PDB word 6 | anti-replay scorecard 2 [present if AR=1 and ARlen>32] | | | |
| PDB word 7 | anti-replay scorecard 3 [present if AR=1 and ARlen>64] | | | |
| PDB word 8 | anti-replay scorecard 4 [present if AR=1 and ARlen>96] | | | |

Table 9-66. MACsec decapsulation PDB for GCM-XPN cipher suites

| | Descriptor Header (1 or 2 words) | | | |
|---|---|---|---|---|
| PDB word 0 | AAD Length<br>(16 bits) | ARLen<br>(8 bits) | Options<br>(8 bits)<br>[see table below] | |

Table continues on the next page...

## Table 9-66. MACsec decapsulation PDB for GCM-XPN cipher suites (continued)

| | | |
|---|---|---|
| PDB word 1 | SCI 1 | |
| PDB word 2 | SCI 2 | |
| PDB word 3 | SSCI | |
| PDB word 4 | salt 1 | |
| PDB word 5 | salt 2 | |
| PDB word 6 | salt 3 | |
| PDB word 7 | EPN | DECO updates PDB in descriptor buffer and external memory as needed |
| PDB word 8 | PN | |
| PDB word 9 | anti-replay scorecard 1 [present if AR=1] | |
| PDB word 10 | anti-replay scorecard 2 [present if AR=1 and ARlen>32] | |
| PDB word 11 | anti-replay scorecard 3 [present if AR=1 and ARlen>64] | |
| PDB word 12 | anti-replay scorecard 4 [present if AR=1 and ARlen>96] | |

## Table 9-67. MACsec decapsulation PDB, format of the options byte

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Reserved | AR | SAinSCI | outFMT | Reserved | Reserved | AKS | FCS |

## Table 9-68. MACsec decapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| 7 | Reserved |
| 6<br>AR | Anti-replay enable<br>0: Anti-replay checking not enabled.<br>1: Anti-replay checking enabled |
| 5<br>SAinSCI | 0: use SCI as is from PDB for decapsulation<br>1: construct SCI using SA from input frame plus lower 16bits of SCI2 from PDB (for portID) |
| 4 | Reserved |
| 3<br>outFMT | 0: SECTAG, ICV and FCS stripped from output frame<br>1: SECTAG, ICV, and FCS left in output frame |
| 2 | Reserved |
| 1<br>AKS | Automatic Key Switching performed<br>0: Automatic Key Switching not performed -- Class 1 Key is always used<br>1: Automatic Key Switching performed -- input frame SecTag AN field LSB selects key. |
| 0 | FCS included in Output Frame<br>0: FCS not included in Output Frame.<br>1: FCS computed by encapsulator and included into Input Frame |

## 9.5 IEEE 802.11-2012 WPA2 MPDU encapsulation and decapsulation

IEEE 802.11 "WiFi" is a popular standard that provides wireless LAN services. Security has evolved within the WiFi standard; Wired-Equvalent Privacy, or WEP, was the original security service. WEP was replaced by Temporal Key Integrity Protocol (TKIP) in 2002, and by WiFi Protected Access (WPA) in 2003. The 80211i-2004 amendment settled upon WPA2, utilizing CCMP.

SEC supports WPA2 CCMP encapsulation and decapsulation of WiFi MPDUs in support of the IEEE 802.11-2012 standard.

**Table 9-69. IEEE 802.11i protocol descriptors**

| Encapsulation | | Decapsulation |
|---|---|---|
| Header | | Header |
| Protocol data block | | Protocol data block |
| Class 1 key data block | | Class 1 key data block |
| Protocol = WiFi encrypt | | Class 2 key data block |
| | | Protocol = WiFi decrypt |

### 9.5.1 Processing Common to WPA2 Encapsulation and Decapsulation

In order for SEC to support WPA2 CCMP encapsulation and decapsulation of WiFi MPDUs, several preprocessing steps have to occur to prepare AESA to receive the MPDU:

- Additional Authentication Data (AAD) requires preparation
- Nonce requires preparation
- AESA Context must be prepared

#### 9.5.1.1 Constructing the AAD for WPA2 encapsulation and decapsulation

The MAC Header is not used as-is as Additional Authentication Data (AAD). Bits in several fields require masking, and the Dur/ID field is not protected at all. In addition, for CCM, AESA requires AAD to be formatted; the 2-byte AAD Length field must be prepended before it is presented to AESA.

This figure shows the construction of formatted Additional Authenticated Data (AAD).

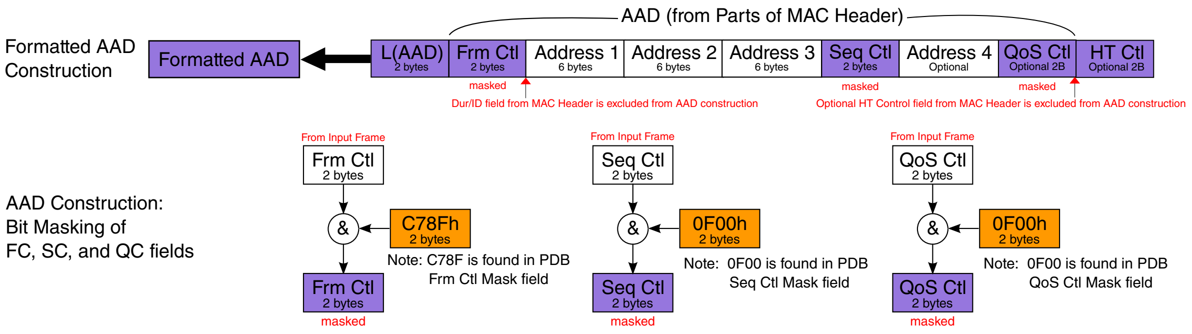

AAD (from Parts of MAC Header)
Formatted AAD Construction
Formatted AAD
L(AAD) 2 bytes
Frm Ctl 2 bytes
Address 1 6 bytes
Address 2 6 bytes
Address 3 6 bytes
Seq Ctl 2 bytes
Address 4 Optional
QoS Ctl Optional 2B
HT Ctl Optional 2B
masked
masked
masked
Dur/ID field from MAC Header is excluded from AAD construction
Optional HT Control field from MAC Header is excluded from AAD construction
AAD Construction: Bit Masking of FC, SC, and QC fields
From Input Frame
Frm Ctl 2 bytes
&
C78Fh 2 bytes
Note: C78F is found in PDB Frm Ctl Mask field
Frm Ctl 2 bytes
masked
From Input Frame
Seq Ctl 2 bytes
&
0F00h 2 bytes
Note: 0F00 is found in PDB Seq Ctl Mask field
Seq Ctl 2 bytes
masked
From Input Frame
QoS Ctl 2 bytes
&
0F00h 2 bytes
Note: 0F00 is found in PDB QoS Ctl Mask field
QoS Ctl 2 bytes
masked

**Figure 9-67. 802.11 CCMP construction of formatted AAD**

SEC builds the AAD as shown in the figure above, using fields in the MAC Header. The first 2 bytes of the formatted AAD contain the length of the AAD proper.

Note that SEC bit masks the Frame Control and Sequence Control, and QoS Control fields of the MAC header per the IEEE 802.11 spec (as illustrated above), using masks stored in the descriptor's Protocol Data Block.

The AAD is authentication-only data provided to the AES engine. That is, the parts of the MAC header protected as AAD are authenticated, but are not encrypted. AAD is constructed for AESA only; the unmodified MAC header is copied from the input frame to the output frame.

### 9.5.1.2 Constructing the CCMP Nonce for WPA2 encapsulation and decapsulation

SEC constructs the AES-CCM nonce using:

- A constant priority (Pri) field extracted from the PDB
- The Address2 field found in the MAC header from the input frame
- A packet number (PN) extracted from the PDB (and for decapsulation, matched against the PN found in the input frame).

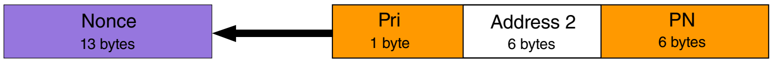


**Figure 9-68. CCMP nonce construction**

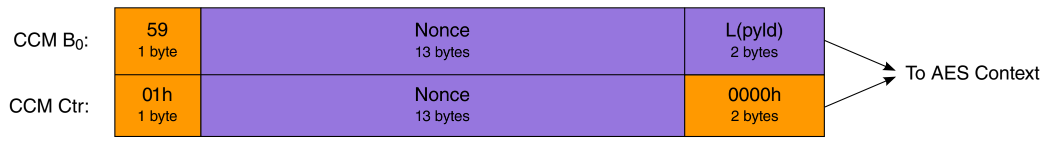
The PN is incremented following use, and the incremented value is written back to the protocol data block.

### 9.5.1.3 Constructing the AESA context for WPA2 CCMP encapsulation and decapsulation

For AES-CCM, AESA requires $B_0$ and $CTR_0$ to be constructed and written to the Class 1 context register before it can start processing the input frame, per the example formatting function described in Appendix A of NIST Special Publication 800-38C. The 13-byte nonce is the bulk of both 16-byte values, and is prepared first.

To construct $B_0$, SEC first computes a 2-byte representation of the length of the payload. The $B_0$; flags byte is extracted from the PDB, and the three are concatenated together as shown below.

To construct $CTR_0$, SEC extracts the CTR flags byte and the Counter Init Count field from the PDB, and concatenates the three as shown below.



Note: 59 to be programmed into PDB field $B_0$ Flags
01h to be programmed into PDB field CTR Flags
0000h to be programmed into PDB field CTR Init Count

**Figure 9-69. WPA2 CCMP context construction**

## 9.5.2 Process for WPA2 encapsulation

This figure shows the WPA2 encapsulation process.

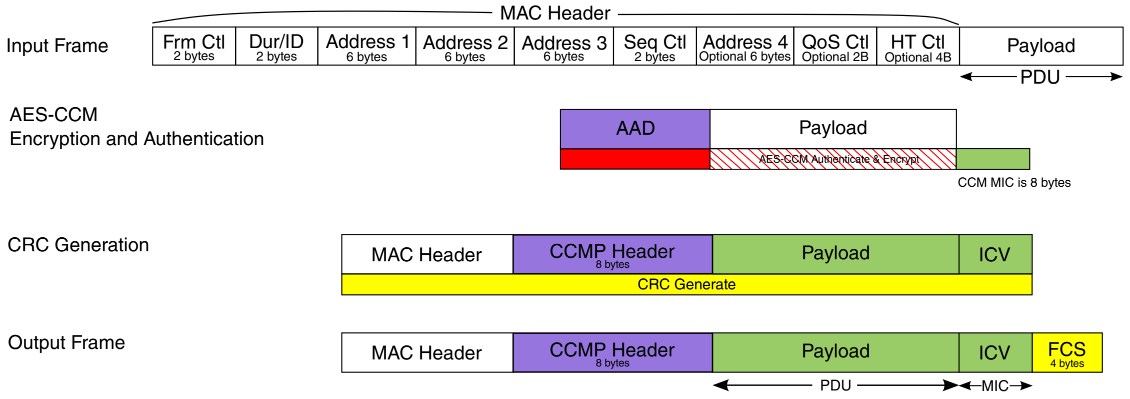


**Figure 9-70. WPA2 encapsulation process**

For WPA2 encapsulation, SEC requires that the input frame consist of an 802.11 MPDU, including the MAC header and the payload requiring cryptographic protection. The MAC Header Length field in the PDB can be used to determine the contents of the MAC header. If HTE is set in the PDB Options field, then the HT Control field is expected. Alternatively, by setting the DFC bit in the PDB Options field, the MPDU header will be parsed to determine the contents of the header.

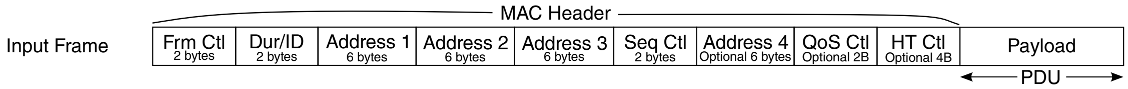


**Figure 9-71. WPA2 encapsulation input frame (802.11 MPDU)**

SEC performs the encapsulation procedure by doing the following:

1. Receives the input frame.
2. Uses fields from the MAC Header to construct the AAD (see Constructing the AAD for WPA2 encapsulation and decapsulation for more details).
3. Uses fields from the PDB and MAC header to construct the nonce (see Constructing the CCMP Nonce for WPA2 encapsulation and decapsulation for more details).
4. Uses the nonce to construct both the CCM Initial Counter value and CCM $B_0$, both of which are written to the Class 1 CHA Context Register (see Constructing the AESA context for WPA2 CCMP encapsulation and decapsulation for more details).
5. Constructs a CCMP header and pushes it onto the output frame immediately following the MAC header; note that this header is not cryptographically protected (see Constructing the CCMP header for WPA2 encapsulation for more details).
6. Extracts the payload from the input frame and pushes it into AESA. (see WPA2 Payload Encapsulation for more details)
7. AESA automatically produces the appropriate MAC and encrypts it to produce the ICV.
8. Pushes the encrypted payload and ICV onto the output frame.

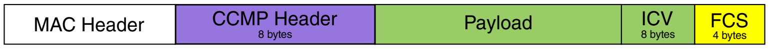


**Figure 9-72. WPA2 CCMP-encapsulated output frame**

All data pushed onto the output frame (up through the ICV) may also be pushed into CRCA for CRC computation. If you have CRC computation enabled, see Computing the FCS for WPA2 encapsulation.

### 9.5.2.1 Constructing the CCMP header for WPA2 encapsulation

SEC constructs a CCMP header, using the original packet number (PN1, PN2) and Key ID found in the protocol data block, plus a constant 00h byte. This construction is pushed onto the output frame immediately following the MAC header, but is not cryptographically protected.

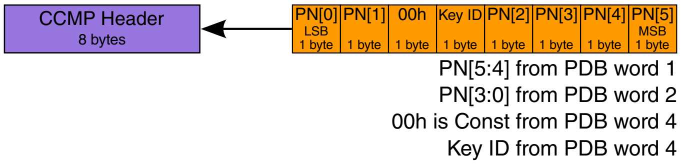


**Figure 9-73. WPA2 CCMP header construction**

### 9.5.2.2 WPA2 Payload Encapsulation

The payload is extracted from the input frame and pushed into AESA. As part of the WPA2 authentication and encryption computation, AESA automatically produces the appropriate CBC-MAC and encrypts it to produce the ICV. The encrypted payload and ICV are pushed onto the output frame.

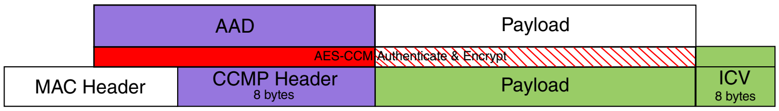


**Figure 9-74. WPA2 cryptographic encapsulation**

### 9.5.2.3 Computing the FCS for WPA2 encapsulation

As produced, all data pushed onto the output frame (up through the ICV) may also be pushed into CRCA for CRC computation. If CRC computation is enabled, SEC appends the resulting frame check sequence (FCS) to the output frame.

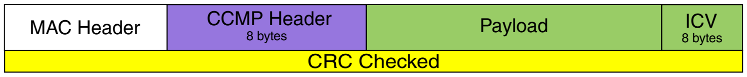


**Figure 9-75. WPA2 checksum encapsulation**

### 9.5.2.4 WPA2 encapsulation PDB format descriptions

**Table 9-70. 802.11 WPA2 encapsulation PDB**

| | Descriptor Header (1 or 2 words) | | | | |
|---|---|---|---|---|---|
| PDB Word 0 | MAC Header Length<br>(16 bits) | | rsv<br>(8 bits) | Options<br>(8 bits) | |
| PDB Word 1 | $B_0$ Flags<br>(8 bits) | Pri<br>(8 bits) | PN 1<br>(16 bits) | | DECO writes PN back to PDB as needed |
| PDB Word 2 | PN 2 | | | | |
| PDB Word 3 | Frm Ctl Mask<br>(16 bits) | | Seq Ctl Mask<br>(16 bits) | | |
| PDB Word 4 | QoS Ctl Mask<br>(16 bits) | | Const<br>(8 bits) | KeyID<br>(8 bits) | |
| PDB Word 5 | CTR Flags<br>(8 bits) | rsv<br>(8 bits) | CTR Init Count<br>(16 bits) | | |

**Table 9-71. WPA2 encapsulation PDB, format of the options byte**

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Reserved | Reserved | DFC | Reserved | Reserved | HTE | Reserved | FCS |

**Table 9-72. WPA2 encapsulation PDB, description of the options byte**

| Field | Description |
|---|---|
| 7-6 | Reserved |
| 5<br>DFC | Decode Frame Control<br>0: do not decode Frame Control field; determine format of MAC Header from Mac Header Length and HTE bit.<br>1: use Frame Control field to determine format of MAC Header (HTE must be programmed to 0; PDB MAC Header Length field is ignored). |
| 4-3 | Reserved |
| 2 HTE | HT Enable<br>0: When DFC=0, HT Control field is not present in MAC Header<br>1: When DFC=0, HT Control field may be present in MAC Header. SEC will use Frame Control field from input frame to determine if MAC Header includes an HT Control field. If an HT Control field is determined to be present, it is skipped from the authentication processing that most of the MAC header undergoes.<br>*Note: HTE must be 0 if DFC=1* |
| 1 | Reserved |
| 0<br>FCS | FCS included in Output Frame<br>0 FCS not included in Output Frame.<br>1 FCS computed by CRCA and copied into Output Frame |

### 9.5.2.5 WPA2 encapsulation error conditions

This table lists the conditions under which WPA2 encapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine.

**Table 9-73. WPA2 encapsulation error conditions**

| Condition | Error Status |
|---|---|
| Reserved bit set to 1 in the PDB options byte | Protocol PDB error |
| DFC=1 and HTE=1 in the PDB Options byte | Protocol PDB error |
| OPERATION Command PROT ID selects IEEE 802.11 WiFi WPA2 Encap, and PROTINFO is not a valid protocol | Protocol Command Error |
| PN overflows | Protocol Sequence Number Overflow |

## 9.5.3 Process for WPA2 decapsulation

SEC performs WPA2 decapsulation by doing the following:

1. Receives an input frame containing the original MAC header, a CCMP header, an encrypted payload and ICV, and an optional frame check sequence (FCS) (see the following figure).

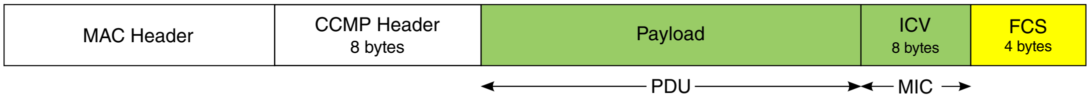


**Figure 9-76. WPA2 decapsulation input frame**

2. Constructs the Nonce as described for encapsulation, using the MAC header from the input frame, and various resources programmed in the PDB.
3. Constructs the AAD as described for encapsulation, by masking and removing fields from the MAC header.
4. For AES-CCM mode, constructs the formatted AAD from the constructed AAD, and constructs the AESA context, using the Nonce, and various resources programmed in the PDB.
5. Copies the MAC header from the input frame to the output frame.
   - If PDB Option DFC=0, the MAC Header Length field in the PDB determines the number of bytes to copy.
   - If DFC=1, the Frame Control field of the MAC header is parsed to determine the length and contents of the MAC Header.

6. Optional: If enabled, writes the entire input frame into CRCA for CRC computation (see the following figure).

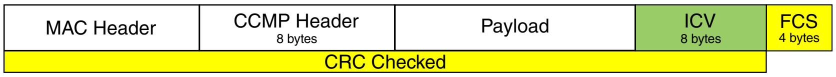


**Figure 9-77. WPA2 decapsulation checksum verification**

7. If the computed CRC does not validate the frame check sequence (FCS) found at the end of the input frame, a CRC fail is indicated in the Job Completion Status Word.
8. Optional: If anti-replay checking is enabled, each received PN is compared to the packet number maintained in the PDB, and if the values are different, REPLAY is indicated in the Job Completion Status Word (see WPA2 Decapsulation Anti-replay checking).
9. The AAD is pushed, tagged for authentication only, to AESA.
10. Following AAD, the encrypted payload and ICV are pushed into AESA and CRCA; the payload is tagged for authentication and decryption.
11. AESA automatically computes an ICV and compares it to the received ICV. If the two ICVs do not match, an ICV CHECK FAIL is indicated in the Job Completion Status Word.

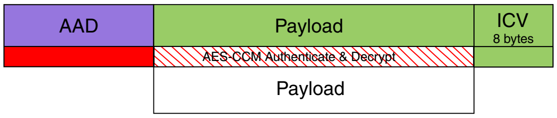


**Figure 9-78. WPA2 cryptographic decapsulation**

12. The decrypted payload is pushed onto the output frame immediately following the MAC header.

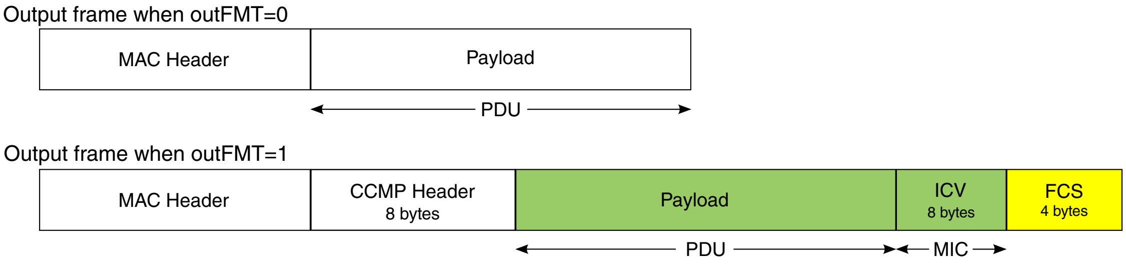


**Figure 9-79. WPA2 decapsulation Output Frame (802.11 MPDU)**

### 9.5.3.1 WPA2 Decapsulation Anti-replay checking

If enabled by the options byte AR field, each received PN is compared to the packet number maintained in the PDB. If the value of PN received is not equal to that previously stored, REPLAY is indicated in the Job Completion Status Word. If Anti-Replay is disabled, a packet received with an out-of-sequence PN produces an ICV error and does not decapsulate correctly.

The use of the AR option allows the user to know whether an ICV error was caused by an out-of-sequence PN.

### 9.5.3.2 Using automatic key-switching

SEC WPA2 decapsulation has the ability to automatically switch between two keys. If Automatic key-switching (AKS) is set in the PDB, SEC uses the least significant bit of the KeyID field of the CCMP Header received in the input frame to select which key is used.

- If the bit is set, the Class 2 Key is used.
- If the bit is cleared, the Class 1 Key is used.

This feature allows reception of frames without software having to preprocess to determine which key is being used.

### 9.5.3.3 WPA2 decapsulation PDB format descriptions

**Table 9-74. 802.11 WPA2 decapsulation PDB**

<table>
<tr><td></td><td colspan="4">Descriptor Header (1 or 2 words)</td><td></td></tr>
<tr><td>PDB Word 0</td><td colspan="2">MAC Header Length<br>(16 bits)</td><td>rsv<br>(8 bits)</td><td>Options<br>(8 bits)</td><td></td></tr>
<tr><td>PDB Word 1</td><td>$B_0$ Flags<br>(8 bits)</td><td>Pri<br>(8 bits)</td><td colspan="2">PN 1<br>(16 bits)</td><td rowspan="2">DECO writes PN back to PDB as needed</td></tr>
<tr><td>PDB Word 2</td><td colspan="4">PN 2</td></tr>
<tr><td>PDB Word 3</td><td colspan="2">Frm Ctl Mask<br>(16 bits)</td><td colspan="2">Seq Ctl Mask<br>(16 bits)</td><td></td></tr>
<tr><td>PDB Word 4</td><td colspan="2">QoS Ctl Mask<br>(16 bits)</td><td colspan="2">rsv<br>(16 bits)</td><td></td></tr>
<tr><td>PDB Word 5</td><td>CTR Flags<br>(8 bits)</td><td>rsv<br>(8 bits)</td><td colspan="2">CTR Init Count<br>(16 bits)</td><td></td></tr>
</table>

Table 9-75. WPA2 decapsulation PDB, format of the options byte

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | AR | DFC | Reserved | outFMT | HTE | AKS | FCS |

Table 9-76. WPA2 decapsulation PDB, description of the options byte

| Field | Description |
|---|---|
| 7 | Reserved |
| 6<br>AR | Anti-replay enable<br>0 Anti-replay checking not enabled.<br>1 Anti-replay checking enabled |
| 5<br>DFC | Decode Frame Control field<br>0: do not decode Frame Control field; determine format of MAC Header from Mac Header Length and HTE bit.<br>1: use Frame Control field to determine format of MAC Header (HTE must be programmed to 0; PDB MAC Header Length field is ignored). |
| 4 | Reserved. Must be zero. |
| 3<br>outFMT | Output Frame Format<br>0: CCMP / GCMP Header, ICV, and FCS stripped from output frame<br>1: CCMP Header, ICV, and FCS included in output frame |
| 2<br>HTE | HT Field Enable<br>0: When DFC=0, HT Control field is not present in MAC Header<br>1: When DFC=0, HT Control field may be present in MAC Header. SEC will use Frame Control field from input frame to determine if MAC Header includes an HT Control field. If an HT Control field is determined to be present, it is skipped from the authentication processing that most of the MAC header undergoes.<br>*Note: HTE must be 0 if DFC=1* |
| 1<br>AKS | Automatic Key Switching<br>0 Automatic Key Switching not performed -- Class 1 Key is always used<br>1 Automatic Key Switching performed -- input frame CCMP Header KEYID field selects key. |
| 0<br>FCS | FCS included in input Frame<br>0 FCS not included in inputFrame.<br>1 FCS computed by encapsulator and included into Input Frame |

### 9.5.3.4 WPA2 decapsulation error conditions

This table lists the conditions under which WPA2 Decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine; authentication failure can produce an ICV check error.

Table 9-77. WPA2 decapsulation error conditions

| Condition | Error status |
|---|---|
| Reserved bit set to 1 in the PDB options byte | Protocol PDB error |
| DFC=1 and HTE=1 | Protocol PDB error |
| OPERATION Command PROT ID selects WPA2 Decap, and PROTINFO is not a valid protocol | Protocol Command Error |
| PN overflows | Protocol Sequence Number Overflow |
| AR = 1, and PN is not received Sequentially (PN PDB doesn't match received PN) | Protocol REPLAY Error |

## 9.6 IEEE 802.16 WiMAX encapsulation and decapsulation overview

The IEEE 802.16 standard describes how the generic MAC header (or GMH) is to be modified during the encapsulation process. SEC assumes the input frame includes the GMH appropriate for transmission with the encapsulated payload. If applying AES-CCM, SEC uses fields from the GMH to build the initial counter $CTR_0$ and $B_0$ values required to be written into Class 1 Context for AESA to perform AES-CCM processing, but does not perform any of the modifications specified in IEEE 802.16. Software is required to include the GMH with the value of the EC bit changed, with the length field updated to include the ICV and FCS additions, and with the HCS updated.

The IEEE 802.16 standard specifies the use of AES-CCM for confidentiality and data integrity protection, without any Additional Authentication Data (AAD). The standard does not require confidentiality or data integrity.

The PROTINFO field of the Protocol Operation command specifies whether or not AES-CCM is to be applied to the flow.

- If AES-CCM is applied to encapsulation, the payload is encrypted and an ICV is computed and inserted into the output frame.
- If AES-CCM is not applied to encapsulation, then no encryption, ICV computation, or ICV insertion is performed.
- If AES-CCM is not applied to decapsulation, then decryption is not performed, and no ICV check is performed.

Table 9-78. IEEE 802.16 protocol descriptors

| Encapsulation | | Decapsulation |
|---|---|---|
| Header | | Header |
| Protocol data block | | Protocol data block |

Table continues on the next page...

Table 9-78. IEEE 802.16 protocol descriptors (continued)

| Encapsulation | | Decapsulation |
|---|---|---|
| Class 1 key data block | | Class 1 key data block |
| Protocol = WiMAX encrypt | | Class 2 key data block |
| | | Protocol = WiMAX decrypt |

### 9.6.1 Process for IEEE 802.16 WiMAX encapsulation

SEC performs IEEE 802.16 WiMAX encapsulation by doing the following:

1. Receives an input frame including the encapsulation-appropriate GMH and payload

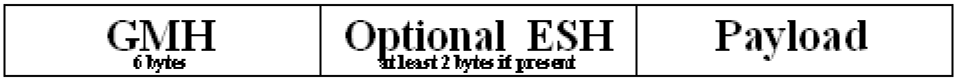
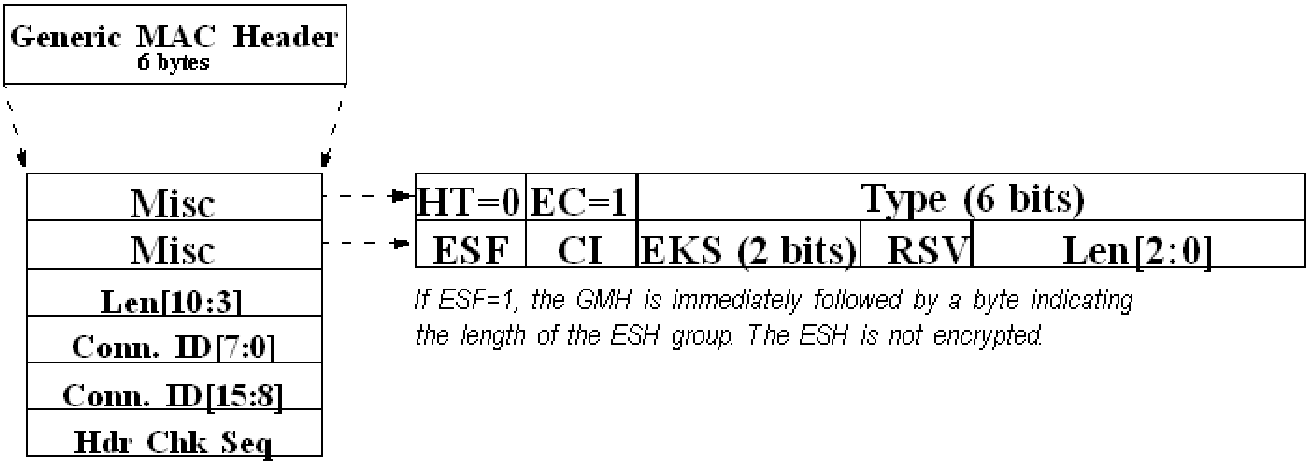


Figure 9-80. 802.16 WiMAX encapsulation input frame



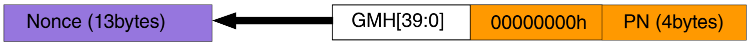
*If ESF=1, the GMH is immediately followed by a byte indicating the length of the ESH group. The ESH is not encrypted.*

Figure 9-81. 802.16 WiMAX generic MAC header detail

2. Takes this input frame and performs the following actions:
   - Optional: If selected, encrypts and authenticates the payload
   - Inserts a packet number
   - Optional: If enabled, inserts a frame check sequence (FCS) generated using a CRC32 algorithm.
3. Optional: If the ESF bit in the GMH is set, indicating that extended subheaders (ESH) are present in the input frame, the ESH is passed to the output frame and FCS protected (if CRC checking is enabled), but is not passed to the AESA for authentication or confidentiality.

- The first byte of the ESH is the length, in bytes, of the ESH.
- There must be at least one more byte of ESH after the ESH length or an error is returned.

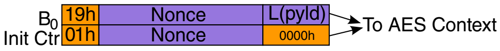
4. Uses the first 5 bytes of the GMH, along with a PDB-maintained 4-byte constant and 4-byte packet number (PN), and builds a nonce.
   - The packet number is incremented and written back to the PDB after use.
   - (optional) If the most significant bit of PN changes upon incrementing, the Job Completion Status Word contains the ROLLOVER indicator.
5. Writes the GMH, Optional ESH, and original PN to the output frame and passes them to CRCA



Note: 0000 0000h to be programmed into PDB field Nonce Constant

**Figure 9-82. 802.16 WiMAX nonce construction**

6. Uses the 13-byte nonce to build the initial counter value $CTR_0$ and the $B_0$, which are provided to the Class 1 Context Register for AES to use in performing CCM computations.
   - The constants used in Nonce construction and in $CTR_0$ and $B_0$ construction are extracted from the PDB, future proofing against the possibility of a simple change to the WiMAX spec.



Note: 19h to be programmed into PDB field $B_0$Flags
01h to be programmed into PDB field CTR Flags
0000h to be programmed into PDB field CTR Init Count

**Figure 9-83. 802.16 WiMAX AES-CCM context construction**

### NOTE

If the Protocol Operation command PROTINFO field indicates AES-CCM is not used, construction of $CTR_0$ and $B_0$ is skipped.

7. After loading the key, $CTR_0$, and $B_0$ into the AES engine, uses AES-CCM mode to encrypt the payload and compute an ICV
8. Writes the encrypted payload and ICV the output frame and passes them to CRCA.

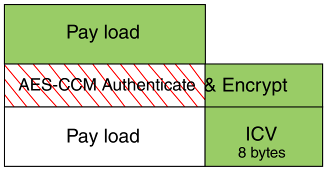


**Figure 9-84. 802.16 WiMAX cryptographic encapsulation**

9. CRC is computed across the entire output frame, and the FCS is appended to the output frame (see the following figure).

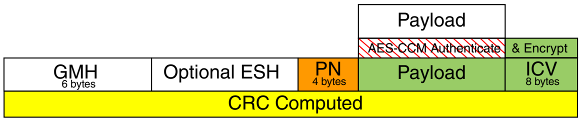


**Figure 9-85. 802.16 WiMAX cryptographic and checksum encapsulation**

After the FCS has been appended, the output frame resulting from WiMAX encapsulation consists of:

- the software modified GMH (6 bytes)
- an optional ESH (present if the ESF bit is set in the GMH)
- the packet number (4 bytes)
- the encrypted payload
- he AES-CCM produced ICV
- the CRC-produced FCS

**NOTE**

The WiMAX spec uses two different FCS computation schemes depending on the type of connection. The built-in CRC engine is capable of performing either of the required CRC types.

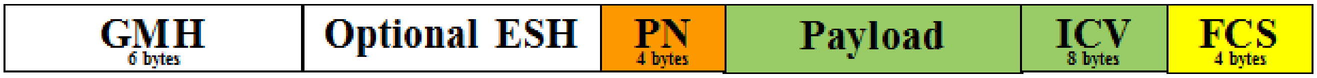


**Figure 9-86. 802.16 WiMAX encapsulation output frame**

The processing that SEC performs presumes processor precomputation of the encapsulation- appropriate Generic MAC Header. As shown in Figure 9-86, HT and EC are shown to be set as appropriate for an encapsulated packet, not a decapsulated one. However, Figure 9-86 fails to show that the GMH processor precomputation includes updating the Len field (adding 4 for the PN, 8 for the ICV, and 4 for the FCS) and updating the Header Check Sequence as appropriate for the other changes made to the GMH.

## 9.6.2 IEEE 802.16 WiMAX encapsulation PDB format descriptions

The IEEE 802.16 WiMAX specification shows two different formats for PN: that labelled PN and that shown as transmitted on the wire. In the PDB, PN is stored (and incremented) in spec PN order, and the order is reversed for building the nonce and for building the output frame.

**Table 9-79. WiMAX encapsulation PDB**

<table>
<tr><td></td><td colspan="4">Descriptor Header (1 or 2 words)</td><td></td></tr>
<tr><td>PDB Word 0</td><td colspan="3">Reserved<br>(24 bits)</td><td>Options<br>(8 bits)</td><td rowspan="3"></td></tr>
<tr><td>PDB Word 1</td><td colspan="4">Nonce Constant</td></tr>
<tr><td>PDB Word 2</td><td>$B_0$ Flags<br>(8 bits)</td><td>$CTR_0$ Flags<br>(8 bits)</td><td colspan="2">Counter Initial Count<br>(16 bits)</td></tr>
<tr><td>PDB word 3</td><td colspan="4">PN</td><td>DECO updates PDB as needed</td></tr>
</table>

**Table 9-80. WiMAX encapsulation PDB, format of the options byte**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | FCS |

**Table 9-81. WiMAX encapsulation PDB, description of the options byte**

| Field | Description |
|---|---|
| 7-1 | Reserved |
| 0<br>FCS | FCS included in Output Frame<br>0 FCS not included in Output Frame.<br>1 FCS computed by encapsulator and included into Input Frame |

## 9.6.3 WiMax encapsulation error conditions

This table lists the conditions under which WiMAX encapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine.

**Table 9-82. WiMAX encapsulation error conditions**

| Condition | Error status |
|---|---|
| Reserved bit set to 1 in the PDB options byte | Protocol PDB error |
| OPERATION Command PROT ID selects WMAX Encap, and PROTINFO is not a valid protocol | Protocol Command Error |
| ESH enabled in GMH, and ESH Length byte < 2 | Protocol Command Error |
| PN overflows | Protocol Sequence Number Overflow |

### 9.6.4 Procedure for IEEE 802.16 WiMAX decapsulation

SEC performs decapsulation by doing the following:

1. Receives the input frame without software modification to the GMH (see following figure)

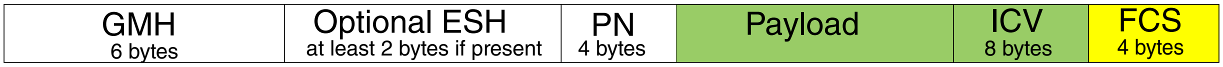
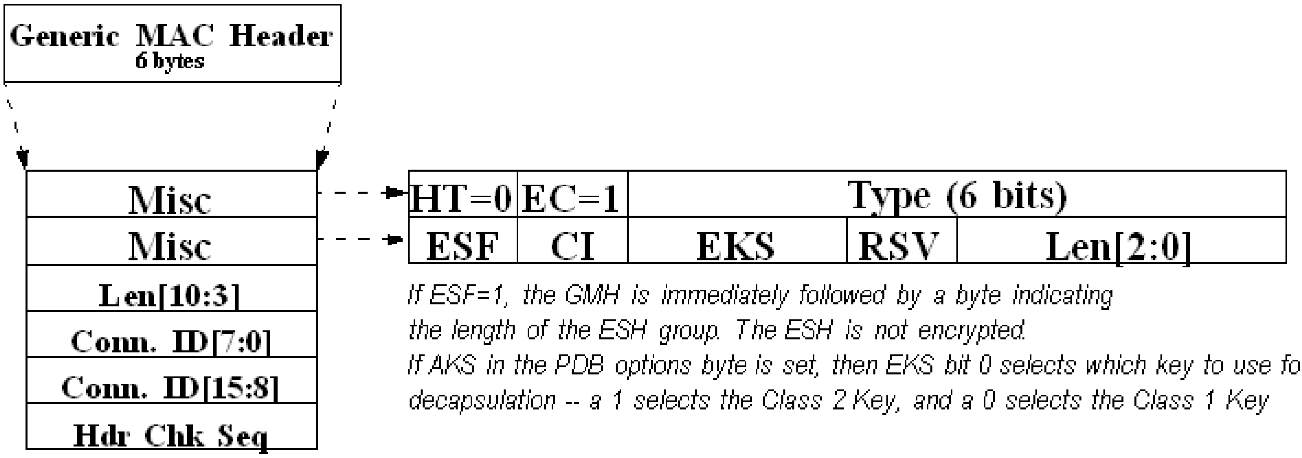


**Figure 9-87. 802.16 WiMAX decapsulation input frame**

2. Optional: If the frame check sequence has not been verified by software or other external hardware prior to SEC receiving the input frame, verifies the FCS, signalling an error if the received FCS does not match the expected result
   - This FCS check occurs in parallel with other SEC processing of the input frame.
   - The FCS is computed across the GMH, the ESH (present if ESF =1 in the GMH), the PN, the Payload, and the ICV.
   - If an ESH is present, the ESH must be at least 2 bytes in length or an error is returned.



*If ESF=1, the GMH is immediately followed by a byte indicating the length of the ESH group. The ESH is not encrypted.*
*If AKS in the PDB options byte is set, then EKS bit 0 selects which key to use for decapsulation -- a 1 selects the Class 2 Key, and a 0 selects the Class 1 Key*

**Figure 9-88. 802.16 WiMAX generic MAC header detail**

3. Constructs the nonce from the GMH and packet number

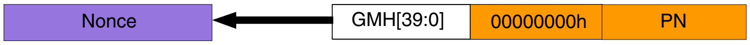
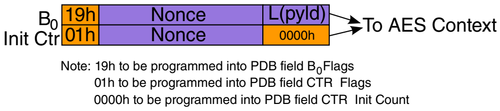


**Figure 9-89. 802.16 WiMAX nonce construction**

4. Builds the AES-CCM initial counter ($CTR_0$) and $B_0$ values using constants extracted from the PDB. Note that if the Protocol Operation command PROTINFO field indicates AES-CCM is not used, then construction of $CTR_0$ and $B_0$ is skipped.



Note: 19h to be programmed into PDB field B$_0$Flags
01h to be programmed into PDB field CTR Flags
0000h to be programmed into PDB field CTR Init Count

**Figure 9-90. 802.16 WiMAX AES-CCM context construction**

5. Completes decapsulation of the packet by decrypting the payload and computing the ICV on the received, decrypted payload.
6. Computes and compares the ICV to the ICV value received, and if the two ICVs fail to match, an error is asserted.

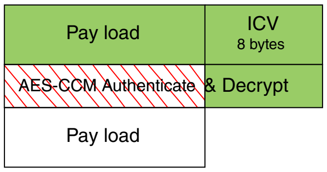


**Figure 9-91. 802.16 WiMAX cryptographic decapsulation**

7. After the received ICV is checked against the computed ICV, performs anti-replay checking on the Packet Number field.
   - The PDB is configurable for either 32, 64, or 128-packet windows (size determined by the 2-byte Anti-replay length field), and all state information is stored back into the PDB before SEC finishes decapsulating the packet.
   - SEC does not pass the PN onto the output frame.
8. After decryption, pushes the payload to the output frame.

The packet in the output frame consists of the unmodified GMH, the ESH if present in the input frame, and the decrypted payload. Unless option outFMT is set, The ICV, FCS, and PN fields are not included in the Output Frame.

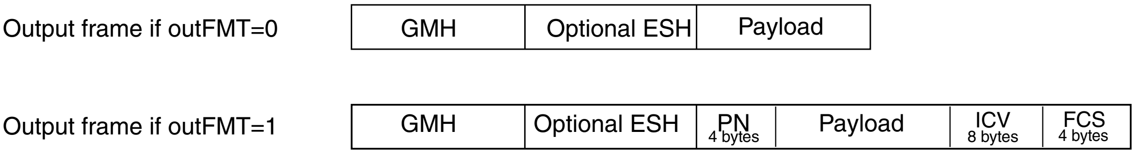


**Figure 9-92. 802.16 WiMAX decapsulation output frame**

### 9.6.4.1 Transforming the GMH (WiMAX decapsulation)

The decapsulated output frame still contains the encapsulation GMH. Transformation of the GMH (setting EC to 0, reducing the length as appropriate, and recomputing the header check sequence) requires processor intervention after SEC finishes processing the PDU.

### 9.6.4.2 Automatic key switching (WiMAX decapsulation)

SEC WiMAX decapsulation has the capability to automatically switch between two keys. If AKS (Automatic Key Switching) is set in the PDB, then SEC uses the least significant bit of the EKS field of the GMH received in the input frame to select which key is used; if the bit is asserted, then the Class 2 Key is used. If the bit is negated, then the Class 1 Key is used. This feature allows reception of frames without software having to preprocess to determine which key is being used.

## 9.6.5 IEEE 802.16 WiMAX decapsulation PDB format descriptions

**Table 9-83. WiMAX decapsulation PDB**

<table>
<tr><td></td><td colspan="4">Descriptor Header (1 or 2 words)</td><td></td></tr>
<tr><td>PDB Word 0</td><td colspan="3">Reserved<br>(24 bits)</td><td>Options<br>(8 bits)</td><td rowspan="3"></td></tr>
<tr><td>PDB Word 1</td><td colspan="4">Nonce Constant</td></tr>
<tr><td>PDB Word 2</td><td>$B_0$ Flags<br>(8 bits)</td><td>$CTR_0$ Flags<br>(8 bits)</td><td colspan="2">Counter Initial Count<br>(16 bits)</td></tr>
<tr><td>PDB word 3</td><td colspan="4">PN</td><td rowspan="6">DECO updates PDB in descriptor buffer and external memory as needed</td></tr>
<tr><td>PDB Word 4</td><td colspan="2">reserved</td><td colspan="2">Anti-Replay Length<br>(16 bits)</td></tr>
<tr><td>PDB word 5</td><td colspan="4">Anti Replay Scorecard 1 [present if AR=1]</td></tr>
<tr><td>PDB word 6</td><td colspan="4">anti-replay scorecard 2 [present if AR=1 and Anti-Replay Length > 32]</td></tr>
<tr><td>PDB word 7</td><td colspan="4">anti-replay scorecard 3 [present if AR=1 and Anti-Replay Length > 64]</td></tr>
<tr><td>PDB word 8</td><td colspan="4">anti-replay scorecard 4 [present if AR=1 and Anti-Replay Length > 96]</td></tr>
</table>

**Table 9-84. WiMAX decapsulation PDB, format of the options byte**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | AR | Reserved | Reserved | outFMT | Reserved | AKS | FCS |

**Table 9-85. WiMAX Decapsulation PDB - Description of the Options Byte**

| Field | Description |
|---|---|
| 7 | Reserved |
| 6<br>AR | Anti-replay enable<br>0 Anti-replay checking not enabled.<br>1 Anti-replay checking enabled; length determined by PDB field Anti-Replay Length |
| 5-4 | Reserved. Must be zero. |

*Table continues on the next page...*

**Table 9-85. WiMAX Decapsulation PDB - Description of the Options Byte (continued)**

| Field | Description |
|---|---|
| 3 | output frame format<br>0 PN, ICV, and FCS stripped from output frame.<br>1 PN, ICV, and FCS included in output frame. |
| 2 | Reserved. Must be zero. |
| 1<br>AKS | Automatic Key Switching performed<br>0 Automatic Key Switching not performed; Class 1 Key is always used.<br>1 Automatic Key Switching performed; input frame EKS field of GMH selects key. |
| 0<br>FCS | FCS included in input Frame<br>0 FCS not included in input Frame.<br>1 FCS computed by encapsulator and included into Input Frame |

## 9.6.6 WiMAX decapsulation error conditions

This table lists the conditions under which WiMAX decapsulation generates an error status. Note that these are the error conditions directly detected by the protocol engine. Authentication failure can produce an ICV check error.

**Table 9-86. WiMAX decapsulation error conditions**

| Condition | Error Status |
|---|---|
| Reserved bit set to 1 in the PDB options byte | Protocol PDB error |
| AR = 1, and Anti-Replay Length > 64 | Protocol PDB error |
| AR = 0, and Anti-Replay Length != 0 | Protocol PDB error |
| OPERATION Command PROT ID selects WiMAX Decap, and PROTINFO is not a valid protocol | Protocol Command Error |
| ESH enabled in GMH, and ESH Length byte < 2 | Protocol Command Error |
| PN overflows | Protocol Sequence Number Overflow |
| Anti-Replay detects a LATE packet | Protocol LATE Error |
| Anti-Replay detects a REPLAY packet | Protocol REPLAY Error |

## 9.7 Anti-Replay built-in checking

Several network protocol decapsulation commands include the ability to perform Anti-Replay checking. This capability is actually built as a separate command that can be invoked separately. As a separate command, Anti-Replay supports a packet number of 16, 32, 48, or 64 bits in length, and an anti-replay window sized anywhere between 1 and 128 entries.

The Anti-Replay operation compares the Packet Number stored in the PDB to the Packet Number stored right-justified in MATH0. It is the responsibility of other descriptor commands to put the Packet Number into MATH0 in the correct form. Four checks can be performed by the Anti-Replay operation:

1. Late Check determines if the Packet Number found in MATH0 is smaller than the Packet Number found in the PDB by at least the size of the Anti Replay Window. That is, MATH0's Packet Number is so old that it cannot be checked for a replay. If determined LATE, then either an error or a warning will be signaled as determined by PDB options bit RLST.
2. Replay Check determines if the Packet Number found in MATH0 is already reflected by appropriate entry in the Anti Replay Window. In particular, the window entry associated with the value in MATH0 is checked. If already set, then it is determined to be REPLAY. If not yet set, then it is not REPLAY, and the entry is set. If determined REPLAY, then either an error or a warning will be signaled as determined by PDB options bit RLST.
3. Packet Number Overflow check (optional; occurs if COF=1) detects if Packet Number in MATH0 has rolled past the maximum Packet Number, as determined by the length of the Packet Number. An Overflow indicates the Packet Number in the PDB is close to $2^{PNLen}$ - 1, but the Packet Number in MATH0 is close to zero. When an overflow is detected, either an error or a warning is signalled, as determined by Option OUST.
4. Packet Number Underflow check (optional; occurs if CUF=1) detects if Packet Number in MATH0 has rolled back past zero. For this to be detected, the Packet Number in the PDB must be close to zero and the Packet Number in MATH0 must be close to $2^{PNLen}$ - 1. When an underflow is detected, either an error or a warning is signalled, as determined by Option OUST.

If the value in MATH0 is a not a replay and is newer than the Packet Number in the PDB, then the PDB is updated with the Packet Number value from MATH0, and the Anti Replay Window is shifted such that the newest entry corresponds to the new Packet Number. The Packet Number and Anti Replay window fields are updated back to memory where the descriptor was fetched from.

Upon successful completion of the Anti-Replay operation, the status field of the PDB is updated.

**NOTE**

> If an error is signalled, then descriptor execution is terminated immediately; further tasks will not be completed.

When the Anti-Replay operation is invoked as a stand-alone operation, it requires its own PDB that is of a form different than seen when Anti-Replay is part of a network protocol. ARLen, a field in the first PDB word, control the size of the anti-replay window, and is programmable to any integer between 1 and 128. Options byte field PNLen selects the packet number length. Options bits CUF and COF indicate whether or not to check for packet number underflow (roll-under) or overflow (roll-over). Bit OUST determines whether to signal an error or a warning for any detected packet number underflow or overflow. Bit RLST determines whether to signal an error or a warning if either a LATE or a REPLAY Packet Number is detected.

**Table 9-87. Anti-Replay built-in checking PDB**

| | Descriptor Header (1 or 2 words) | | | | |
|---|---|---|---|---|---|
| PDB Word 0 | Status<br>(4 bits) | Reserved<br>(12 bits) | ARLen<br>(8 bits) | Options<br>(8 bits) | |
| PDB Word 1 | Upper Packet Number (unused if PNLen is 32 or 16) | | | | DECO writes back to PDB as needed |
| PDB Word 2 | Lower Packet Number | | | | |
| PDB Word 3 | Anti Replay Window entries 31-0 (always present) | | | | |
| PDB Word 4 | Anti Replay Window entries 63-32 (present if ARLen > 32) | | | | |
| PDB Word 5 | Anti Replay Window entries 95-64 (present if ARLen > 64) | | | | |
| PDB Word 6 | Anti Replay Window entries 127-96 (present if ARlen > 96) | | | | |

**Table 9-88. Anti-Replay built-in checking PDB, format of the status nibble**

| 31 | 30 | 29 | 28 |
|---|---|---|---|
| PNUpdate | LATE | REPLAY | OUFD |

**Table 9-89. Anti-Replay built-in checking PDB, description of the status nibble**

| Field | Description |
|---|---|
| 31<br>PNUpdate | Indicates if the Packet Number field in the PDB was updated by the Anti Replay command<br>0 : was not updated<br>1 : was updated |
| 30<br>LATE | LATE was detected but no error was issued<br>0 : no LATE detected<br>1 : LATE Packet Number detected |
| 29<br>REPLAY | REPLAY was detected but no error was issued<br>0 : no REPLAY detected |

*Table continues on the next page...*

Table 9-89. Anti-Replay built-in checking PDB, description of the status nibble (continued)

| Field | Description |
|---|---|
| | 1 : REPLAY Packet Number detected |
| 28<br>OUFD | Underflow or Overflow Packet Number detected but no error issued<br>0 : no Underflow or Overflow detected<br>1 : Underflow or Overflow of Packet Number detected |

Table 9-90. Anti-Replay built-in checking PDB, format of the options byte

| 7-6 | 5-4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| Reserved | PNLen | CUF | COF | OUST | RLST |

Table 9-91. Anti-Replay built-in checking PDB, description of the options byte

| Field | Description |
|---|---|
| 7-6 | Reserved |
| 5-4<br>PNLen | Selects Packet Number Size<br>00 : use 16 bit Packet Number<br>01 : use 32 bit Packet Number<br>10 : use 48 bit Packet Number<br>11 : use 64 bit Packet Number |
| 3<br>CUF | Check for packet number Underflow.<br>0 : do not check for underflow<br>1 : check for underflow -- if Packet Number rolls back past zero, signal per OUE |
| 2<br>COF | Check for packet number Overflow.<br>0 : do not check for overflow<br>1 : check for overflow -- if Packet Number rolls forward past $2^{PNLen}$ - 1, signal per OUST |
| 1<br>OUST | Overflow / Underflow signal type.<br>0 : signal warning upon detection of Packet Number Overflow or Underflow<br>1 : signal error upon detection of Packet Number Overflow or Underflow |
| 0<br>RLST | Replay / Late signal type.<br>0 : signal warning upon detection of late or replay Packet Number<br>1 : signal error upon detection of late or replay Packet Number |

## 9.8 Process for 3G double-CRC encapsulation and decapsulation

SEC includes a double-CRC encapsulation and decapsulation protocol thread designed for 3G MAC-d protection.

The unencapsulated frame contains a packet consisting of:

- A PDU header
- A PDU payload consisting of 0 or more bytes

SEC computes either a 7-bit or 11-bit CRC of the PDU Header (the length of which is in the PDB) and the 16-bit CRC of the PDU Payload.

- The 7-bit CRC computation uses an irreducible polynomial of $D^7+D^6+D^2+1$.
- The 11-bit CRC uses an irreducible polynomial is $D^{11}+D^9+D^8+D^2+D+1$.
- The 16-bit CRC uses an irreducible polynomial of $D^{16}+D^{15}+D^2+1$.

For all three computations, the CRC engine is configured with DIS, DOS, DOC, and IVZ all set.

## 9.8.1 3G double-CRC encapsulation process

The encapsulated output frame consists of the following:

- The PDU header, which is copied from the input frame after being modified by insertion of the header CRC
- The PDU payload, which is copied without modification from the input frame
- The sixteen-bit payload CRC

### 9.8.1.1 Calculating the 7-bit CRC of the PDU header for encapsulation

This figure shows 3G double-CRC encapsulation with 7-bit header CRC.

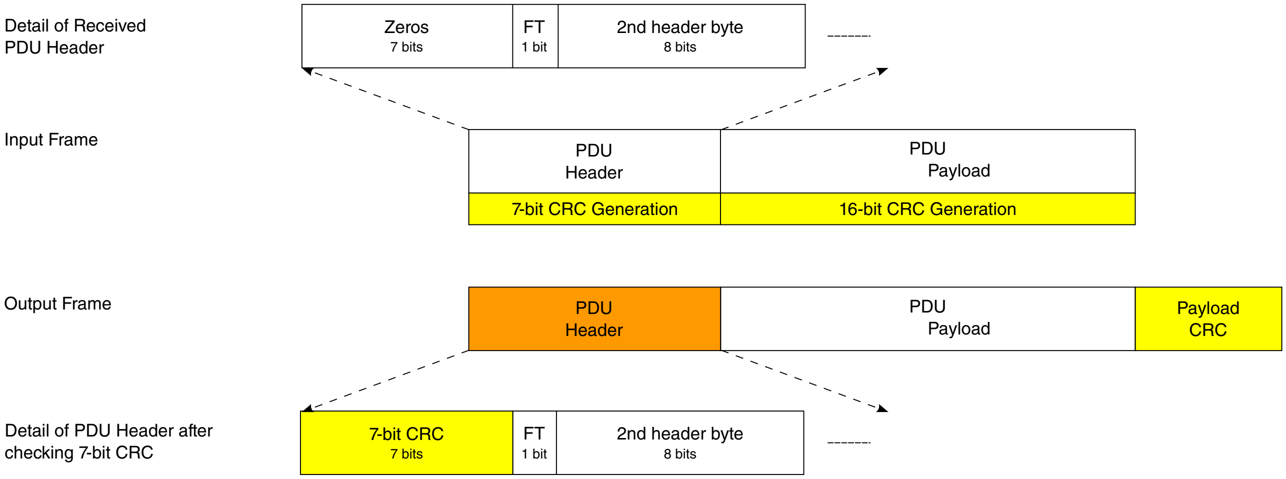


**Figure 9-93. 3G double-CRC encapsulation with 7-bit header CRC**

If the PROTINFO field of the Operation Command selects a 7-bit CRC, it is computed as follows:

1. SEC takes the number of bytes for the PDU Header from the input frame as defined by the PDU Header Length field in the PDB.
2. SEC zeroes the first 7 bits as the header is passed to the CRC engine.
3. After the 7-bit CRC computation is completed, the computed value is moved to a DECO register, along with the FT bit that completes the byte containing the CRC.

### 9.8.1.2 Calculating the 11-bit CRC of the PDU header for encapsulation

This figure shows 3G double-CRC encapsulation with 11-bit header CRC.

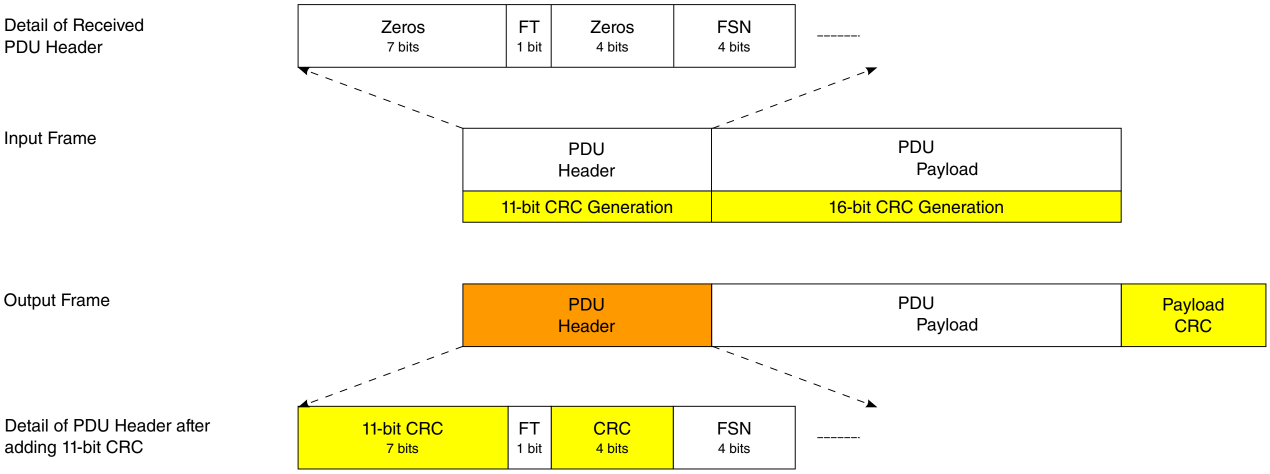


**Figure 9-94. 3G double-CRC encapsulation with 11-bit header CRC**

If the PROTINFO field of the Operation Command selects an 11-bit CRC, it is computed against a modified PDU Header as follows:

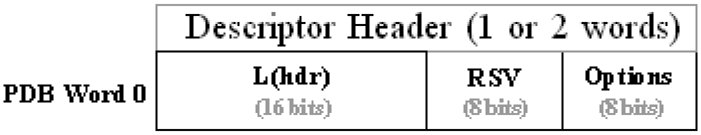
1. SEC takes the number of bytes for the PDU Header from the input frame as defined by the PDU Header Length field in the PDB.
2. The PDU Header bits reserved for the CRC are zeroed.
3. The second byte of the PDU Header is re-ordered before being passed to the CRC engine: the FT bit is moved to be after the entire CRC field.
4. The PDU Header is copied from the input frame to the output frame, unmodified, along with the PDU Payload.
5. The completed 11-bit CRC value is moved to a DECO register, along with the rest of the first two bytes of the PDU Header.

### 9.8.1.3 Calculating the 16-bit payload CRC for encapsulation

After the header CRC computation has been completed, the CRC CHA is reconfigured for the sixteen-bit payload CRC computation.

1. The PDU Payload is passed unmodified from the input frame to the output frame and is also passed to the CRC CHA.
2. The CRC CHA calculates the payload CRC.
3. The CRC value is appended to the end of the output frame by the CRC engine.
4. The saved first byte or two of the PDU Header with inserted CRC is written out, overwriting the start of the output frame.

## 9.8.2 3G double-CRC encapsulation PDB format descriptions



| | Descriptor Header (1 or 2 words) | | |
|---|---|---|---|
| PDB Word 0 | L(hdr)<br>(16 bits) | RSV<br>(8 bits) | Options<br>(8 bits) |

**Figure 9-95. 3G double-CRC encapsulation PDB**

**Table 9-92. 3G double-CRC encapsulation PDB, format of the options byte**

| 7..0 |
|---|
| Reserved |

**Table 9-93. 3G double-CRC encapsulation PDB, description of the options byte**

| Field | Description |
|---|---|
| 0-7 | Reserved |

## 9.8.3 3G double-CRC decapsulation process

The decapsulation process is as follows:

1. The PDU Header CRC is checked and passed as-is from the input frame to the output frame, along with the rest of the PDU Header.
2. If the received header CRC does not match the computed header CRC, an error results.

3. The received payload CRC is checked against the computed payload CRC, but the output frame does not include the payload CRC, meaning that the decapsulated output frame is 2 bytes shorter than the input frame.
4. If the received payload CRC does not match the computed payload CRC, an error is returned.

### 9.8.3.1 Calculating the 7-bit CRC of the PDU header for decapsulation

This figure shows 3G double-CRC decapsulation with 7-bit header.

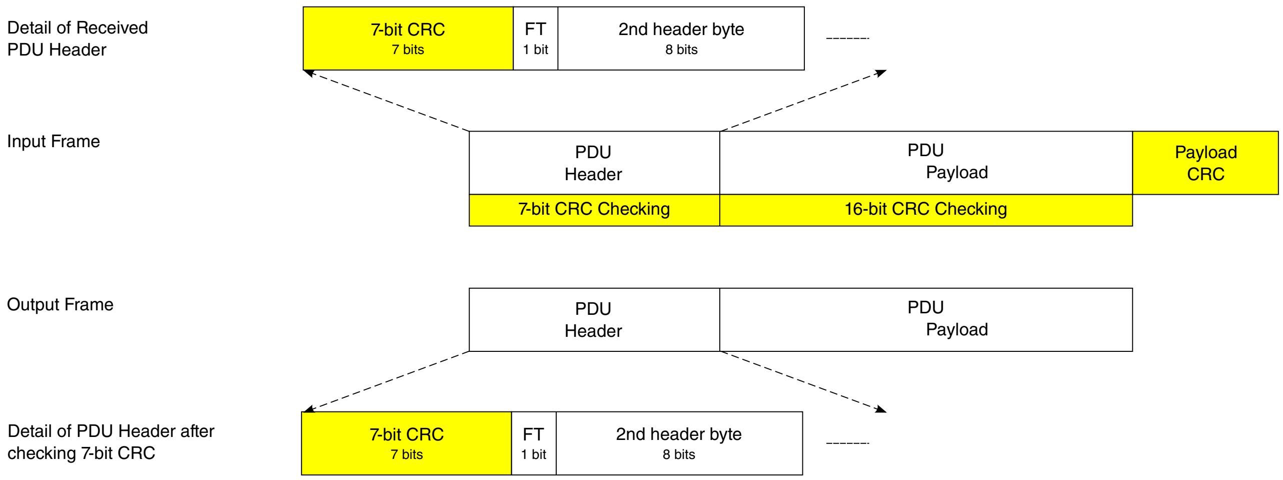


**Figure 9-96. 3G double-CRC decapsulation with 7-bit header**

If the value of the Operation Command PROTINFO field selects a seven-bit CRC computation, the procedure is:

1. The first 7 bits of the PDU header are checked against a CRC computed across the PDU header; the length of the PDU header is determined by the PDU Header Length field in the PDB.
2. The PDU header is copied verbatim from the input frame to the output frame.
3. For the purposes of the CRC computation, the received seven-bit CRC is zeroed prior to passing it to the CRC engine.
4. The seven-bit CRC received from the input frame is written separately into the CRC engine, and is compared to the computed CRC.
5. If the two CRC values do not match, an error is returned.

### 9.8.3.2 Calculating the 11-bit CRC of the PDU header for decapsulation

This figure shows 3G double-CRC decapsulation with a 11-bit header.

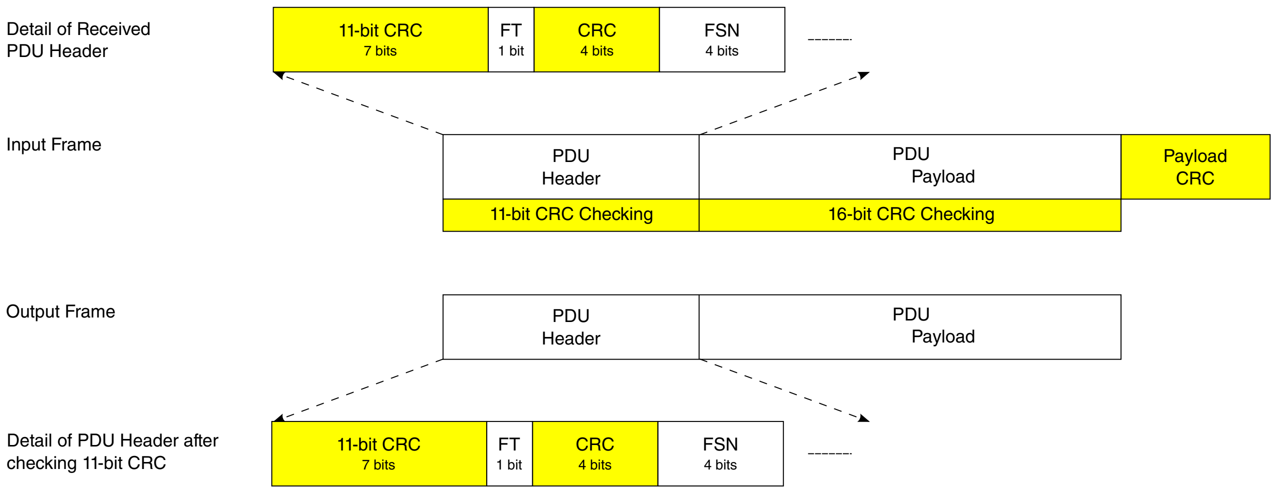


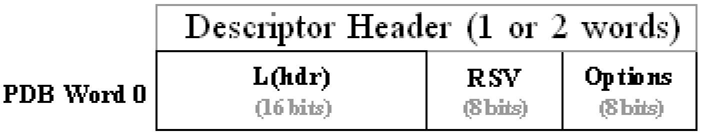
**Figure 9-97. 3G double-CRC decapsulation with 11-bit header**

If the value of the Operation Command PROTINFO field selects an 11-bit CRC computation, the procedure is the same as for the 7-bit CRC except for how the PDU Header is presented to the CRC engine. Because the FT bit appears in the last bit of the first byte of the PDU Header, the first two bytes of the header are reordered going into the CRC engine, in addition to zeroizing the CRC fields. The PDU header is presented such that the FT bit follows eleven zeroed CRC bits, but preceeds all other parts of the PDU header.

### 9.8.3.3 Calculating the 16-bit payload CRC for decapsulation

After the header CRC has been checked, a 16-bit CRC is computed across the PDU payload, which comprises everything after the PDU Header in the input frame except for the final two bytes. The final two bytes of the input frame comprise the payload CRC, which the CRC engine compares to the computed payload CRC. If the two CRC values do not match, an error is returned.

### 9.8.4 3G double-CRC decapsulation PDB format descriptions



| | Descriptor Header (1 or 2 words) | | |
|---|---|---|---|
| PDB Word 0 | L(hdr)<br>(16 bits) | RSV<br>(8 bits) | Options<br>(8 bits) |

**Figure 9-98. 3G double-CRC decapsulation PDB**

**Table 9-94. 3G double-CRC decapsulation PDB, format of the options byte**

| 7..0 |
|---|
| Reserved |

**Table 9-95. 3G double-CRC decapsulation PDB, description of the options byte**

| Field | Description |
|---|---|
| 7-0 | Reserved. |

## 9.9 3G RLC PDU Encapsulation and Decapsulation overview

SEC implements encapsulation and decapsulation for 3G RLC PDUs. For RLC PDUs, only confidentiality is provided; authentication is handled at a different protocol layer. SEC supports both Kasumi-f8 (UEA1) and SNOW-3g f8 (UEA2) for NULL confidentiality (UEA0) as well as for confidentiality.

**Table 9-96. 3G RLC protocol descriptors**

| Encapsulation | | Decapsulation |
|---|---|---|
| Header | | Header |
| Protocol data block, includes HFN | | Protocol data block, includes HFN |
| Class 1 key data block | | Class 1 key data block |
| Protocol = <protocol> encrypt | | Protocol = <protocol> decrypt |

### 9.9.1 3G RLC PDU encapsulation overview

The input frame consists of a single 3G RLC layer PDU header and payload.

- The header is either one or two bytes, depending on the length of the sequence number.
- The payload consists of zero or more bytes.

The encapsulation output frame consists of:

- The PDU header (unmodified)
- The encrypted PDU payload
- Optional padding

Encapsulation can occur in either unacknowledged mode or acknowledged mode:

- If the PDB options byte SNS bit = 1, the PDU is interpreted as an unacknowledged mode PDU, with a 7-bit sequence number in a one-byte PDU Header.
- If the PDB options byte SNS bit is not 1, the PDU is interpreted as an acknowledged mode PDU header, with a twelve-bit sequence number in a two-byte PDU header.

This figure shows 3G RLC PDU encapsulation.

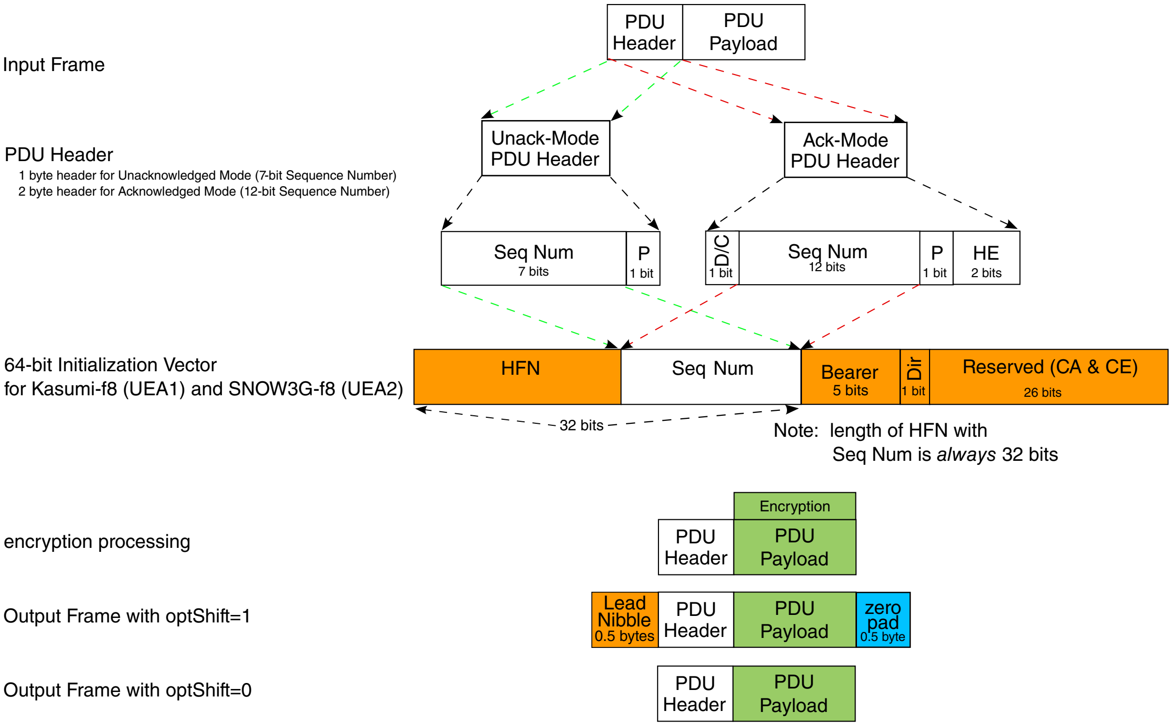

Input Frame
PDU Header
PDU Payload
PDU Header
1 byte header for Unacknowledged Mode (7-bit Sequence Number)
2 byte header for Acknowledged Mode (12-bit Sequence Number)
Unack-Mode PDU Header
Ack-Mode PDU Header
Seq Num 7 bits
P 1 bit
D/C 1 bit
Seq Num 12 bits
P 1 bit
HE 2 bits
64-bit Initialization Vector for Kasumi-f8 (UEA1) and SNOW3G-f8 (UEA2)
HFN
Seq Num
Bearer 5 bits
Dir 1 bit
Reserved (CA & CE) 26 bits
32 bits
Note: length of HFN with Seq Num is *always* 32 bits
encryption processing
Encryption
PDU Header
PDU Payload
Output Frame with optShift=1
Lead Nibble 0.5 bytes
PDU Header
PDU Payload
zero pad 0.5 byte
Output Frame with optShift=0
PDU Header
PDU Payload

**Figure 9-99. 3G RLC PDU encapsulation**

## 9.9.2 Process for 3G RLC PDU encapsulation

The 3G RLC PDU encapsulation procedure is:

1. Prior to starting encryption, the sequence number is extracted and combined with the HyperFrame Number (HFN) field maintained in the PDB.
2. (optional) When the sequence number rolls over from all 1s to zero, the HFN field in the PDB is incremented and written back.
3. Whether incremented or not, the HFN field is checked against the Threshold field in the PDB.
4. If the HFN matches or exceeds the Threshold, a warning is returned when frame processing is complete; this warning indicates keys should be renegotiated at earliest convenience.
5. The PDU Header is copied as-is from the input frame to the output frame. It is used in f8 IV construction, but is not provided to the selected encryption engine (KFHA or SNOW-3G-f8) for encryption.
6. An f8 initialization vector (IV) is built from the following:
   - HFN field as found in the input PDB
   - The PDB word that includes Bearer and Direction
   - The Sequence Number from the input frame
7. The IV is written to the Class 1 Context Register prior to commencing encryption.
8. The entire PDU Payload is moved from the input frame into the input-data FIFO as message data.
9. The resulting encrypted PDU payload is sent to the output frame.

## 9.9.3 3G RLC PDU encapsulation PDB format descriptions

**Table 9-97. 3G RLC PDU encapsulation PDB**

<table>
<tr><td></td><td colspan="5">Descriptor Header (1 or 2 words)</td><td></td></tr>
<tr><td>PDB Word 0</td><td>Resvd<br>(4 bits)</td><td>Lead<br>Nibble<br>(4 bits)</td><td>Reserved<br>(16 bits)</td><td colspan="2">Options<br>(8 bits)</td><td></td></tr>
<tr><td>PDB Word 1</td><td colspan="4">HFN<br>(25 bits)</td><td>Reserved<br>(7 bits)</td><td rowspan="2">DECO writes back to PDB as needed</td></tr>
<tr><td>PDB Word 2</td><td colspan="5">Bearer, Dir, Reserved for CA & CE</td></tr>
</table>

The PDB options byte includes a bit enabling an optional shift. If set, the output frame is 1 byte longer than the input frame, and the entire PDU is offset by 4 bits from the start of the frame. The 4 bits added to the output frame at the front are taken from the Lead Nibble field of the PDB and the 4 bits added at the tail are all zeros.

**Table 9-98. 3G RLC PDU encapsulation PDB, format of the options byte**

| 7-3 | 2-1 | 0 |
|---|---|---|
| Reserved | SNS | optShift |

**Table 9-99. 3G RLC PDU encapsulation PDB, description of the options byte**

| Field | Description |
|---|---|
| 7-3 | Reserved |
| 2-1<br>SNS | Selects Serial Number Size. Ignored for LTE C-Plane.<br>If SNS = 00b : 12-bit Serial Number. This is Acknowledged mode for 3G.<br>If SNS = 01b : 7-bit Serial Number. This is Unacknowledged mode for 3G.<br>If SNS = 10b : 15-bit Serial Number. Not recommended for 3G RLC encapsulation.<br>If SNS = 11b : Reserved. |
| 0<br>optShift | Enables the optional four-bit Output Frame shift<br>If optShift = 0 : No shift<br>If optShift = 1 : Shift the output frame 4 bits, inserting zeros. |

## 9.9.4 3G RLC PDU decapsulation overview

The encapsulated input frame consists of:

- A single 3G RLC layer PDU Header
- An encrypted PDU Payload consisting of zero or more bytes

**NOTE**

If the optShift bit in the PDB Options byte is set, the input frame includes an extra byte beyond the PDU: half the byte (four bits) at the front (before the header) and half the byte (four bits) at the end (after the payload). The frame must be un-shifted before decryption can take place.

The PDU Header is either one or two bytes, depending on the length of the sequence number.

The decapsulation output frame consists of:

- The PDU header (unmodified)
- The decrypted PDU payload

Decapsulation can occur in either unacknowledged or acknowledged mode.

- If the PDB Options byte SNS bit = 1, the PDU is interpreted as an unacknowledged mode PDU, with a 7-bit sequence number in a 1-byte PDU Header.
- If the PDB options byte SNS bit is not 1, the PDU is interpreted as an acknowledged mode PDU header, with a twelve-bit sequence number in a two-byte PDU header.

This figure shows 3G RLC PDU decapsulation.

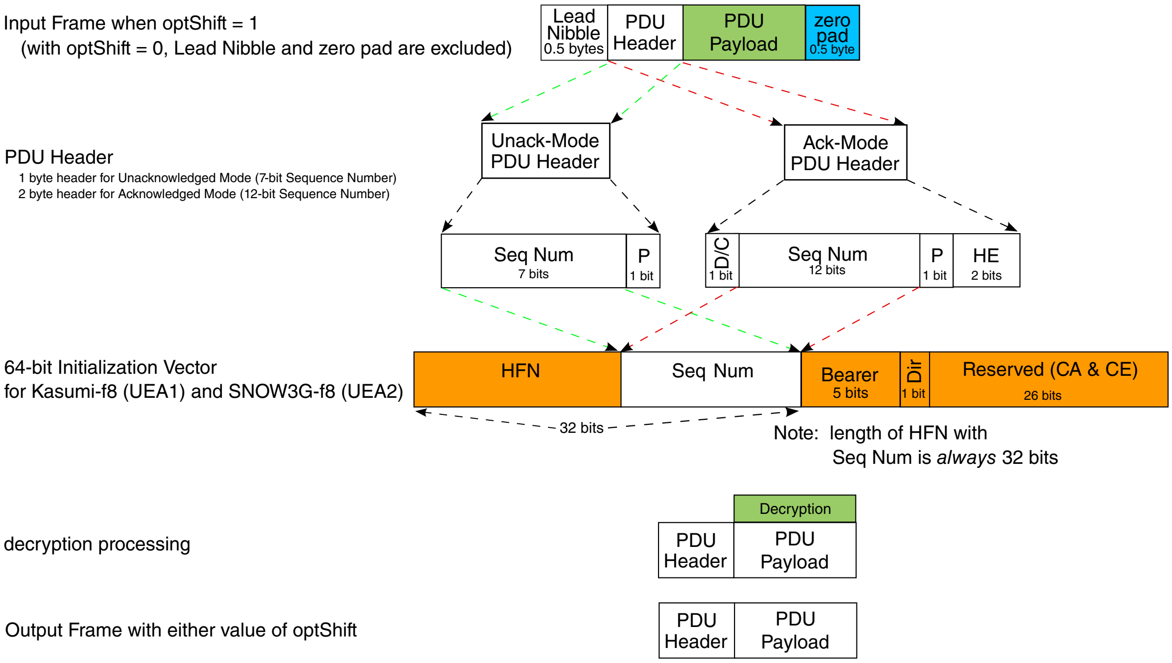

Input Frame when optShift = 1
(with optShift = 0, Lead Nibble and zero pad are excluded)
Lead Nibble 0.5 bytes
PDU Header
PDU Payload
zero pad 0.5 byte
PDU Header
1 byte header for Unacknowledged Mode (7-bit Sequence Number)
2 byte header for Acknowledged Mode (12-bit Sequence Number)
Unack-Mode PDU Header
Ack-Mode PDU Header
Seq Num 7 bits
P 1 bit
D/C 1 bit
Seq Num 12 bits
P 1 bit
HE 2 bits
64-bit Initialization Vector
for Kasumi-f8 (UEA1) and SNOW3G-f8 (UEA2)
HFN
Seq Num
Bearer 5 bits
Dir 1 bit
Reserved (CA & CE) 26 bits
32 bits
Note: length of HFN with
Seq Num is *always* 32 bits
decryption processing
Decryption
PDU Header
PDU Payload
Output Frame with either value of optShift
PDU Header
PDU Payload

**Figure 9-100. 3G RLC PDU decapsulation**

## 9.9.5 Process for 3G RLC PDU decapsulation

The decapsulation procedure is:

1. Prior to starting decryption, the sequence number is extracted and combined with the Hyper Frame Number (HFN) field maintained in the PDB.
2. Whenever the sequence number rolls over from all 1s back to zero, the HFN field in the PDB is incremented and written back; note that there is no provision for rolling back the HFN, so frames must be provided in-order.
3. Whether HFN is incremented or not, the HFN field is checked against the Threshold field in the PDB.
4. If the HFN field matches or exceeds the Threshold field, a warning is returned when frame processing is complete. This warning indicates keys should be renegotiated at earliest convenience.
5. The PDU Header is copied as-is from the input frame to the output frame (shifted if optShift =1). It is used in f8 IV construction, but is not provided to the selected confidentiality engine (KFHA or SNOW-3G-f8) for decryption.
6. An f8 Initialization Vector (IV) is built from:
   - The Hyper Frame Number (HFN) as found in the input PDB

- The PDB word that includes Bearer and Direction
- The Sequence Number from the input frame

7. The IV is written to the Class 1 Context Register prior to commencing decryption.
8. The entire PDU Payload is moved from the Input Frame into the input-data FIFO as message data.
9. The resulting decrypted PDU Payload is sent to the output frame.

## 9.9.6 3G RLC PDU decapsulation PDB format descriptions

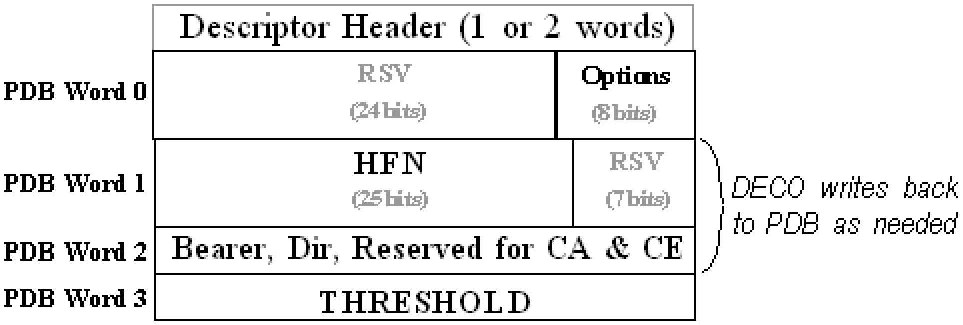


**Figure 9-101. 3G RLC PDU decapsulation PDB**

**Table 9-100. 3G RLC PDU decapsulation PDB, format of the options byte**

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
|  | RSV | RSV | RSV | RSV | RSV | RSV | SNS | optShift |

**Table 9-101. 3G RLC PDU decapsulation PDB, description of the options byte**

| Field | Description |
|---|---|
|  | Reserved. |
| SNS | Selects Serial Number Size. Ignored for LTE C-plane<br>0: 12-bit Serial Number. This is Acknowledged mode for 3G.<br>1: 7-bit Serial Number. This is Unacknowledged mode for 3G. |
| optShift | Selects whether or not to perform a 4-bit shift before decryption.<br>0: No shift<br>1: 4-bit shift performed |

### 9.9.7 Overriding the PDB for 3G RLC PDU encapsulation and decapsulation

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For 3G RLC PDU encapsulation and decapsulation, The HFN is maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired HFN value in the job descriptor. For more information, see Job Ring interface. When using the Queue Manager Interface, QI builds the job descriptor with the LOAD IMMEDIATE to the DPOVRD register with the value of the STATUS/CMD field in the FD. For more information, see Queue Manager Interface (QI).

**Table 9-102. Format of the DPOVRD register when used with the 3G RLC protocol**

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| format when HFN is 20 bits | OVRD | Reserved (12 bits) | | | | | | | | | | | HFN (20 bits) | | | | | | | | | | | | | | | | | | | |
| format when HFN is 25 bits | OVRD | Reserved<br>(7 bits) | | | | | | HFN (25 bits) | | | | | | | | | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 9.10 LTE PDCP PDU encapsulation and decapsulation overview

For LTE, 3GPP moved confidentiality and integrity to both reside in the PDCP layer. Not all modes use authentication.

- PDCP User Plane uses confidentiality only. The encapsulation and decapsulation process is very much like that for 3G RLC PDUs.
- PDCP User Plane for RN uses confidentiality and integrity. The encapsulation and decapsulation process is very much like that for PDCP Control Plane PDUs.
- PDCP Control Plane uses confidentiality and integrity.

The following algorithms are supported for LTE PDCP:

- 128-EEA0 (null-confidentiality) for confidentiality.
- 128-EEA1 (SNOW-3G) for confidentiality.
- 128-EEA2 (AES-CTR) for confidentiality.
- 128-EEA3 (ZUC) for confidentiality.
- 128-EEA0 (null-integrity) for integrity.
- 128-EIA1 (SNOW-3G) for integrity.

- 128-EIA2 (AES-CMAC) for integrity.
- 128-EIA3 (ZUC) for integrity.

**NOTE**

Note that whenever processing the integrity function is included, any combination of confidentiality and integrity algorithm may be selected.

**Table 9-103. LTE control plane protocol descriptors**

| Encapsulation | | Decapsulation |
|---|---|---|
| Header | | Header |
| Protocol data block, includes HFN | | Protocol data block, includes HFN |
| Class 2 key data block | | Class 2 key data block |
| Class 1 key data block | | Class 1 key data block |
| Protocol = <protocol> encrypt | | Protocol = <protocol> decrypt |

## 9.10.1 LTE PDCP PDU IV generation

Each of the algorithms supported by SEC require the generation of initialization vectors (IV). In most cases, a confidentiality IV is written into the Class 1 context register, and a different integrity IV is written into the Class 2 context register to affect processing. EIA2 (AES-CMAC) is the exception; CMAC has no provision for an IV, so the value computed is used as AAD (additonal authenticated data) instead. The various IV values are constructed using Direction, Bearer, some constants programmed into the PDB (in case of future standards work), and Count. Count itself is created by using HFN as the most significant bits and the header's sequence number for least significant bits. Generation of the forms of IV used are shown in the diagrams followiing. These IVs are used for both encapsulation and decapsulation.

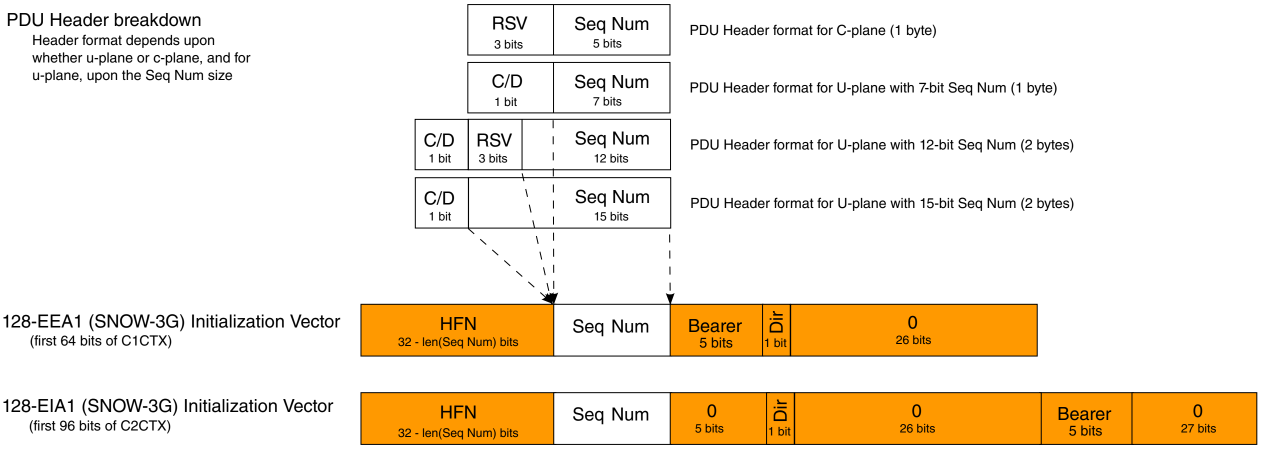

PDU Header breakdown
Header format depends upon whether u-plane or c-plane, and for u-plane, upon the Seq Num size
RSV 3 bits
Seq Num 5 bits
PDU Header format for C-plane (1 byte)
C/D 1 bit
Seq Num 7 bits
PDU Header format for U-plane with 7-bit Seq Num (1 byte)
C/D 1 bit
RSV 3 bits
Seq Num 12 bits
PDU Header format for U-plane with 12-bit Seq Num (2 bytes)
C/D 1 bit
Seq Num 15 bits
PDU Header format for U-plane with 15-bit Seq Num (2 bytes)
128-EEA1 (SNOW-3G) Initialization Vector (first 64 bits of C1CTX)
HFN 32 - len(Seq Num) bits
Seq Num
Bearer 5 bits
Dir 1 bit
0 26 bits
128-EIA1 (SNOW-3G) Initialization Vector (first 96 bits of C2CTX)
HFN 32 - len(Seq Num) bits
Seq Num
0 5 bits
Dir 1 bit
0 26 bits
Bearer 5 bits
0 27 bits

**Figure 9-102. IV generation for 128-EEA1 and 128-EIA1 (SNOW3G)**

Generation of IVs for SNOW3G occurs as follows:

- COUNT is constructed by taking HFN from the PDB and appending the Sequence Number from the input frame
- For the confidentiality IV, Bearer and Direction are taken from the PDB and appended immedlately following COUNT
- The confidentiality IV is completed by appending a 26-bit constant zero value from the PDB (labelled CA & CE).
- For the integrity IV, the Bearer value is moved (and replaced with zeros) from immediately after COUNT to following the end of the confidentiality IV.
- The integrity IV is completed by appending 27 bits of zeros.

## NOTE

The integrity IV is only constructed if integrity processing is being performed.

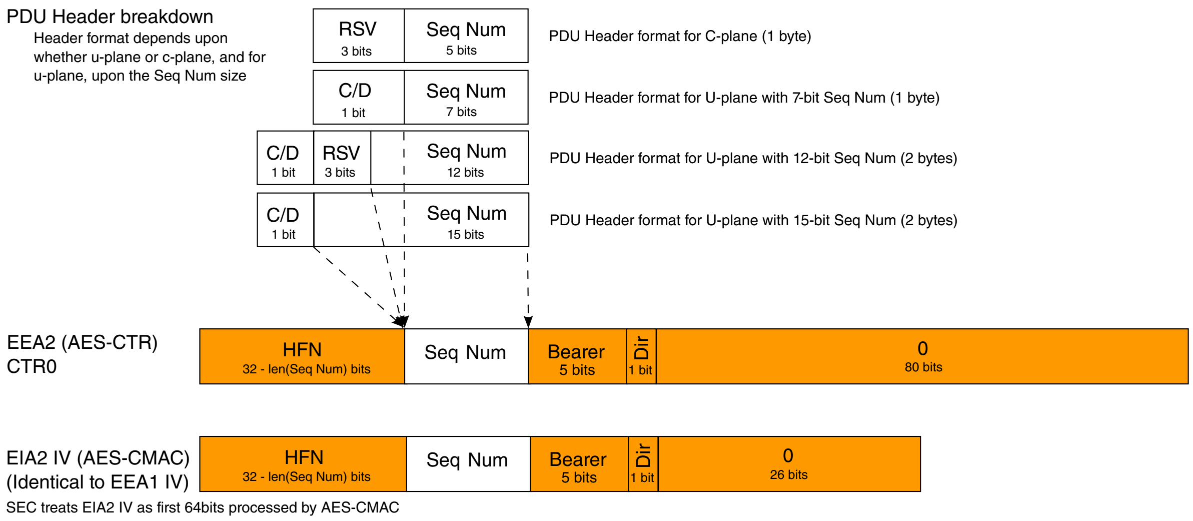

PDU Header breakdown
Header format depends upon whether u-plane or c-plane, and for u-plane, upon the Seq Num size
RSV
3 bits
Seq Num
5 bits
PDU Header format for C-plane (1 byte)
C/D
1 bit
Seq Num
7 bits
PDU Header format for U-plane with 7-bit Seq Num (1 byte)
C/D
1 bit
RSV
3 bits
Seq Num
12 bits
PDU Header format for U-plane with 12-bit Seq Num (2 bytes)
C/D
1 bit
Seq Num
15 bits
PDU Header format for U-plane with 15-bit Seq Num (2 bytes)
EEA2 (AES-CTR)
CTR0
HFN
32 - len(Seq Num) bits
Seq Num
Bearer
5 bits
Dir
1 bit
0
80 bits
EIA2 IV (AES-CMAC)
(Identical to EEA1 IV)
HFN
32 - len(Seq Num) bits
Seq Num
Bearer
5 bits
Dir
1 bit
0
26 bits
SEC treats EIA2 IV as first 64bits processed by AES-CMAC

**Figure 9-103. IV generation for 128-EEA2 and 128-EIA2 (AES)**

Generation of IVs for AES occurs as follows:

- COUNT is constructed by taking HFN from the PDB and appending the Sequence Number from the input frame
- Bearer and Direction are taken from the PDB and appended after COUNT.
- a zero constant is applied to make the confidentiality IV 128 bits, and to make the integrity IV 96 bits.

### NOTE

The integrity IV is only constructed if integrity processing is being performed.

### NOTE

AES-CMAC has no provision for an IV. As a result, the generated integrity IV is applied as additional authenticated data (AAD) prior to the input frame.

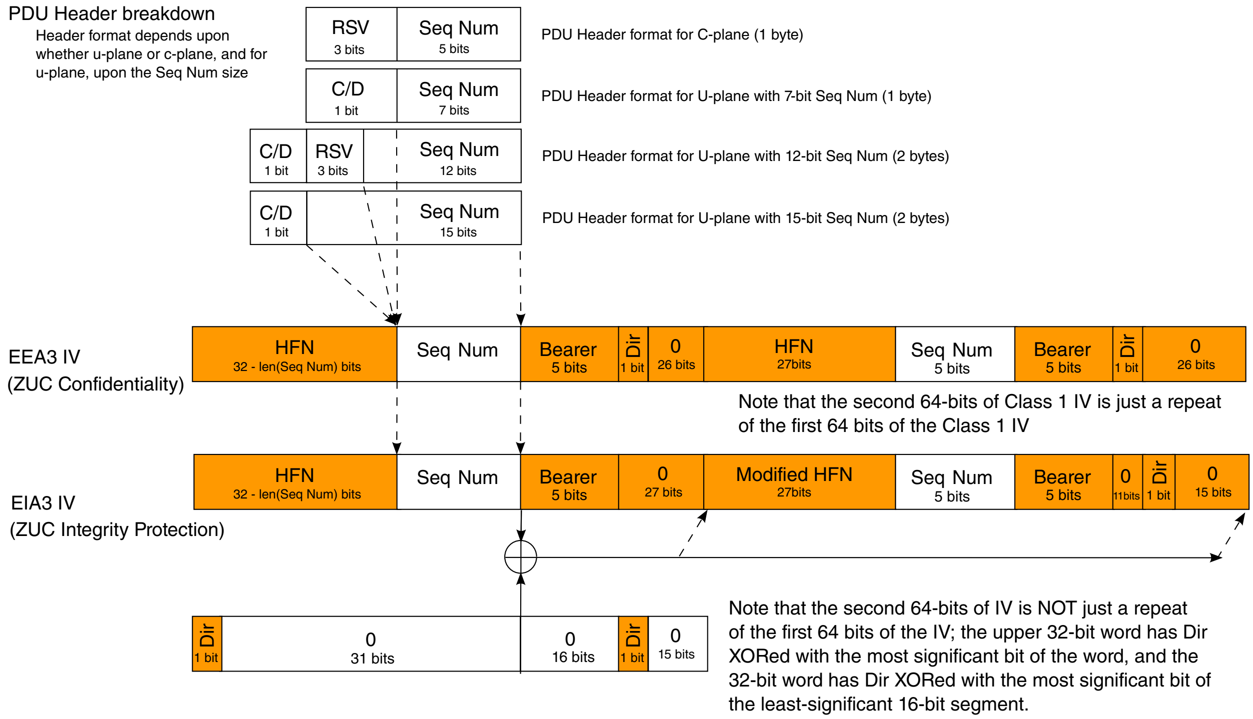

PDU Header breakdown
Header format depends upon whether u-plane or c-plane, and for u-plane, upon the Seq Num size
RSV 3 bits
Seq Num 5 bits
PDU Header format for C-plane (1 byte)
C/D 1 bit
Seq Num 7 bits
PDU Header format for U-plane with 7-bit Seq Num (1 byte)
C/D 1 bit
RSV 3 bits
Seq Num 12 bits
PDU Header format for U-plane with 12-bit Seq Num (2 bytes)
C/D 1 bit
Seq Num 15 bits
PDU Header format for U-plane with 15-bit Seq Num (2 bytes)
EEA3 IV (ZUC Confidentiality)
HFN 32 - len(Seq Num) bits
Seq Num
Bearer 5 bits
Dir 1 bit
0 26 bits
HFN 27bits
Seq Num 5 bits
Bearer 5 bits
Dir 1 bit
0 26 bits
Note that the second 64-bits of Class 1 IV is just a repeat of the first 64 bits of the Class 1 IV
EIA3 IV (ZUC Integrity Protection)
HFN 32 - len(Seq Num) bits
Seq Num
Bearer 5 bits
0 27 bits
Modified HFN 27bits
Seq Num 5 bits
Bearer 5 bits
0 11bits
Dir 1 bit
0 15 bits
Dir 1 bit
0 31 bits
0 16 bits
Dir 1 bit
0 15 bits
Note that the second 64-bits of IV is NOT just a repeat of the first 64 bits of the IV; the upper 32-bit word has Dir XORed with the most significant bit of the word, and the 32-bit word has Dir XORed with the most significant bit of the least-significant 16-bit segment.

**Figure 9-104. IV generation for 128-EEA3 and 128-EIA3 (ZUC)**

Generation of IVs for ZUC occurs as follows:

- The ZUC confidentiality IV is just the SNOW3G confidentiality IV repeated twice.
- The integrity IV is created from the confidentiality IV by removing the Direction bits and
- - A direction bit is XORed into the most significant bit of the lower-half HFN.
  - The direction bit from the lower-half of the confidentiality IV is shifted 10 bits to the right.

**NOTE**

The integrity IV is only constructed if integrity processing is being performed.

## 9.10.2 LTE PDCP PDU encapsulation process for confidentiality only

LTE PDCP PDUs use only confidentiality and no integrity function for user plane operation, when not for relay nodes (RN). A RN may be configured to use integrity protection; for that case see LTE PDCP PDU encapsulation for confidentiality and integrity. The encapsulation procedure is as follows:

1. Prior to starting encryption, Count is created by extracting the sequence number from the input frame header and combining it with the Hyper Frame Number (HFN) maintained in the PDB.
2. Whenever the sequence number rolls over from all 1s back to zero, the HFN field in the PDB is incremented and written back.
3. The HFN field is checked against the Threshold field in the PDB.
4. If the HFN matches or exceeds the Threshold, a warning is returned when frame processing is complete; this warning indicates keys should be renegotiated at earliest convenience.
5. The PDU Header is copied as-is from the input frame to the output frame. It is used in the construction of the IV, but is not provided to the selected encryption engine (SNOW3G or AESA or ZUC) for encryption.
6. A confidentiality initialization vector (IV) is built as shown in LTE PDCP PDU IV generation from:
    - The HFN as found in the input PDB
    - The PDB word that includes Bearer and Direction
    - The Sequence Number from the input frame.
7. This IV is written to the Class 1 Context Register prior to commencing encryption.
8. The entire PDU Payload is moved from the input frame into the input-data FIFO as message data.
9. The resulting encrypted PDU Payload is sent to the output frame.

The process is shown in the figure below.

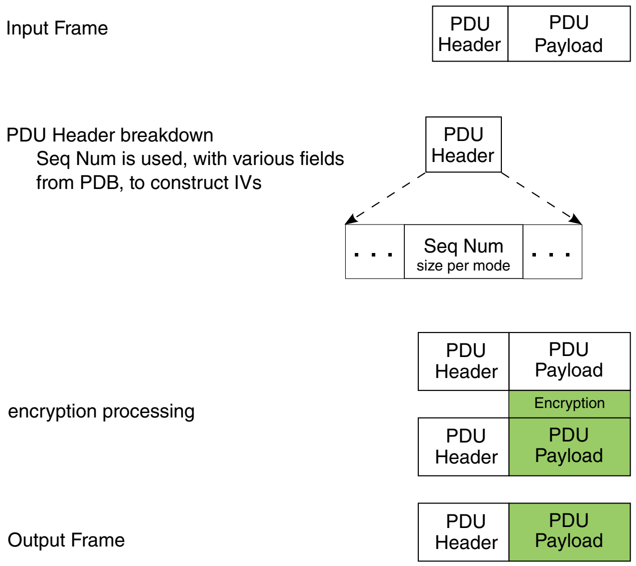


**Figure 9-105. LTE PDCP PDU encapsulation for confidentiality only**

### 9.10.3 LTE PDCP PDU encapsulation for confidentiality and integrity

This figure shows LTE PDCP PDU encapsulation for confidentiality and integrity.

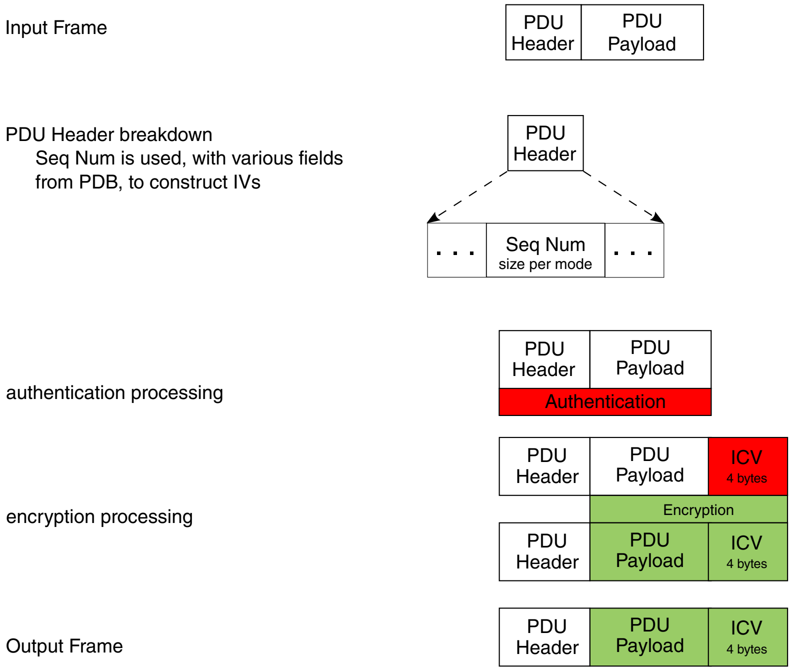


**Figure 9-106. LTE PDCP PDU encapsulation for confidentiality and integrity**

Control plane PDUs include a 1-byte PDU header with a 5-bit sequence number. User plane PDUs for relay nodes (RN) have 7, 12, or 15 bit sequence numbers. User plane headers are 2-bytes for larger sequence numbers, or 1-byte for 7-bit sequence numbers.

1. SEC receives an input frame containing the PDU header and PDU payload.
2. SEC extracts the sequence number from the PDU header.
3. SEC creates the confidentiality IV by taking the two words that contain the Hyper Frame Number (HFN), Bearer, and Direction fields from the PDB and dropping in the sequence number that was extracted from the PDU Header.
4. After the confidentiality IV has been written to the Class 1 Context Register and the integrity IV has been written to the Class 2 Context Register, SEC begins cryptographic computations.
5. The PDU Header is written to the data FIFO to be written to the output frame and also to be processed by the selected integrity (class 2) CHA.

6. The PDU Payload is written to the data FIFO, to be processed by both the confidentiality (class 1) and integrity CHAs. The encrypted PDU Payload is written to the output frame
7. After computation of the integrity MAC-I (Message Authentication Check) value, it is passed back for encryption by the confidentiality CHA.
8. The four-byte encrypted MAC-I is appended to the end of the output frame.

### 9.10.4 LTE PDCP PDU decapsulation process for confidentiality only

LTE PDCP PDUs use only confidentiality for user plane operation, when not for relay nodes (RN). A RN may be configured to use integrity protection; for that case see LTE PDCP PDU decapsulation for confidentiality and integrity. The decapsulation process is:

1. Prior to starting decryption, SEC extracts the sequence number and combines it with the Hyper Frame Number (HFN) maintained in the PDB.
2. Whenever the sequence number rolls over from all 1's back to zero, the HFN field in the PDB is incremented and written back; note that because there is no provision for rolling the HFN back, PDUs must be provided in-order.
3. Whether the HFN is incremented or not, the HFN field is checked against the Threshold field in the PDB.
4. If the HFN matches or exceeds the Threshold, a warning is returned when frame processing is complete; this warning indicates keys should be renegotiated at earliest convenience.
5. The PDU Header is copied as-is from the input frame to the output frame. It is used in the construction of the IV, but is not provided to the selected confidentiality engine (SNOW-3G-f8 or AESA or ZUC) for decryption.
6. A confidentiality Initialization Vector (IV) is built from:
   - The Hyper Frame Number (HFN) as found in the input PDB
   - The PDB word that includes Bearer and Direction
   - The Sequence Number from the input frame.
7. This IV is written to the Class 1 Context Register prior to commencing decryption.

**NOTE**

Note that the IV is constructed slightly differently, depending on the underlying cryptographic algorithm. This is described in LTE PDCP PDU IV generation.

8. The entire PDU Payload is moved from the input frame into the input-data FIFO as message data.
9. The resulting decrypted PDU Payload is sent to the output frame.

The process is shown in the figure below.

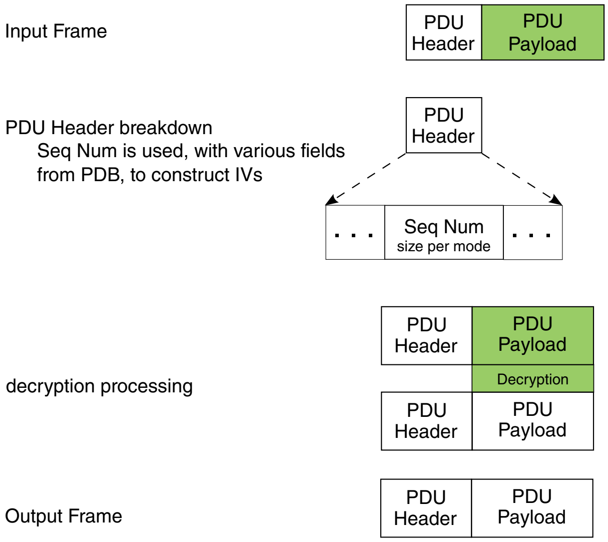


**Figure 9-107. LTE PDCP PDU decapsulation for confidentiality only**

## 9.10.5 LTE PDCP PDU decapsulation for confidentiality and integrity

This figure shows LTE PDCP PDU decapsulation when using confidentiality and integrity.

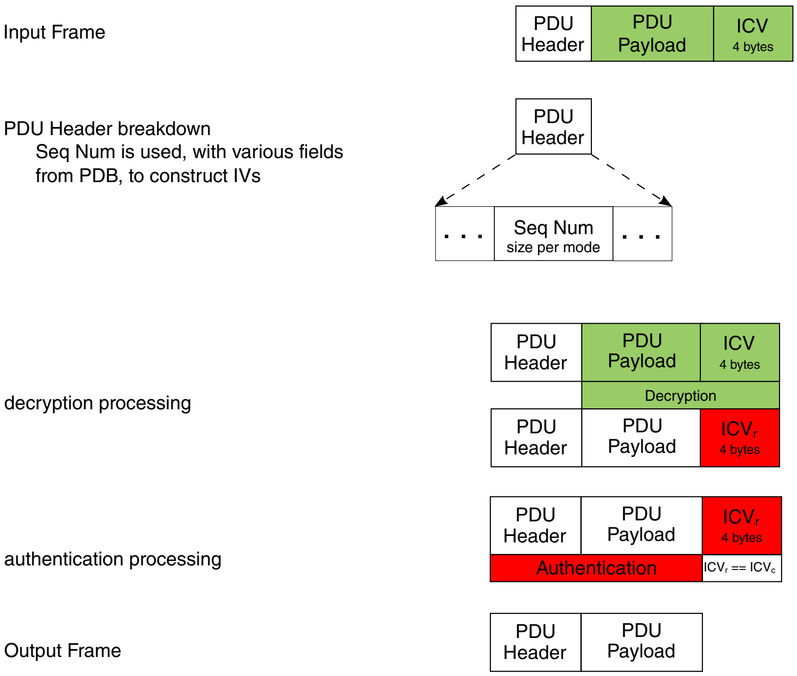


**Figure 9-108. LTE PDCP PDU decapsulation for confidentiality and integrity**

The decapsulation procedure is:

1. SEC receives an input frame that contains the PDU header, PDU payload, and ICV.
2. SEC extracts the sequence number from the PDU header.
3. SEC creates the confidentiality and integrity IVs for decapsulation as described in LTE PDCP PDU IV generation; taking the two words from the PDB containing the Hyper Frame Number (HFN), Bearer, and Direction fields, and inserting in the sequence number that it extracted from the PDU Header.
4. After the Confidentiality IV has been written to the Class 1 Context Register and the integrity IV has been written to the Class 2 Context Register, cryptographic computations begin.
5. Just like for encapsulation, the PDU Header gets written to the data FIFO, to be processed by the selected Class 2 CHA.
6. The PDU Payload also gets written to the data FIFO, to be decrypted. The decrypted data is then passed to the selected Class 2 CHA for the integrity function.
7. After decryption, the received MAC (XMAC-I) is passed into the Class-2 CHA.
8. After computation of the integrity MAC-I (Message Authentication Check) value, it is compared to the XMAC-I value received from the input frame. If the two values are not identical, an error is returned.

The output frame consists of the unmodified PDU Header and the decrypted PDU payload. The XMAC-I is not part of the output frame.

## 9.10.6 LTE PDCP shared descriptor PDB format descriptions

**Table 9-104. LTE PDCP shared descriptor PDB**

| | | | |
|---|---|---|---|
| | Descriptor Header (1 or 2 words) | | |
| PDB Word 0 | Reserved<br>(24 bits) | Options<br>(8 bits) | |
| PDB Word 1 (when 5-bit Sequence Number used) | HFN<br>(27 bits) | Reserved<br>(5 bits) | DECO updates PDB in descriptor buffer and external memory as needed |
| PDB Word 1 (when 7-bit Sequence Number used) | HFN<br>(25 bits) | Reserved<br>(7 bits) | |
| PDB Word 1 (when 12-bit Sequence Number used) | HFN<br>(20 bits) | Reserved<br>(12 bits) | |
| PDB Word 1 (when 15-bit Sequence Number used) | HFN<br>(17 bits) | Reserved<br>(15 bits) | |
| PDB Word 2 | Bearer, Dir, Reserved for CA & CE | | |
| PDB Word 3 | Threshold | | |

**Table 9-105. LTE PDCP shared descriptor PDB, format of the options byte**

| 7-3 | 2-1 | 0 |
|---|---|---|
| Reserved | SNS | Reserved |

**Table 9-106. LTE PDCP shared descriptor PDB, description of the options byte**

| Field | Description |
|---|---|
| 7-3 | Reserved |
| 2-1<br>SNS | Selects Serial Number Size. Ignored for LTE C-Plane.<br>If SNS = 00b : 12-bit Serial Number.<br>If SNS = 01b : 7-bit Serial Number.<br>If SNS = 10b : 15-bit Serial Number.<br>If SNS = 11b : Reserved. |
| | Reserved |

### 9.10.7 Overriding the PDB for LTE PDCP encapsulation and decapsulation

A shared descriptor is created with the intent to provide information required for processing every packet in a flow. Occasionally, it is required to override those standard settings. For LTE PDCP PDU encapsulation and decapsulation, The HFN is maintained in the PDB, but can be overridden through the DPOVRD register, by setting the OVRD bit (see figure below). When using the Job Ring interface, this is achieved by including a LOAD IMMEDIATE to the DPOVRD register of the desired HFN value in the job descriptor. For more information, see Job Ring interface. When using the Queue Manager Interface, QI builds the job descriptor with the LOAD IMMEDIATE to the DPOVRD register with the value of the STATUS/CMD field in the FD. For more information, see Queue Manager Interface (QI).

**Table 9-107. Format of the DPOVRD register when used with the LTE PDCP protocol**

<table>
<tr><td>format when HFN is 17 bits</td><td>OVRD</td><td colspan="14">Reserved (15 bits)</td><td colspan="17">HFN (17 bits)</td></tr>
<tr><td>format when HFN is 20 bits</td><td>OVRD</td><td colspan="11">Reserved (12 bits)</td><td colspan="20">HFN (20 bits)</td></tr>
<tr><td>format when HFN is 25 bits</td><td>OVRD</td><td colspan="6">Reserved (7 bits)</td><td colspan="25">HFN (25 bits)</td></tr>
<tr><td>format when HFN is 27 bits</td><td>OVRD</td><td colspan="4">Resvd (5 bits)</td><td colspan="27">HFN (27 bits)</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

# Chapter 10
# Key agreement functions

The SEC protocol processing capabilities described in Protocol acceleration are centered on bulk data encryption and authentication. This section focuses on key agreement, which is another important part of protocol processing.

Certain protocols specify the use of a pseudo-random function (PRF) as a way that two parties can generate an identical pseudo-random byte string for use as a shared secret key. Internet Key Exchange (IKE) specifies a PRF for use in conjunction with IPsec. SSL and TLS have their own unique PRFs.

Several protocols also specify public key methods for generating or exchanging a shared secret key.

SEC implements the following key agreement methods as built-in functions:

- IKEv2 PRF
- SSL 3.0 PRF
- TLS 1.0, TLS 1.1, DTLS 1.0 PRF
- TLS 1.2, DTLS 1.2 PRF

## 10.1 IKEv2 PRF overview

The Internet Key Exchange v2 pseudo-random function requirements are covered in SEC by two separate but related functions.

- The IKE PRF function can be used to compute SKEYSEED per RFC 5996 section 2.13.
- The IKE PRF+ function can be used to compute IKE Security Association Keying Material per RFC 5996 section 2.14.
- The IKE PRF+ function can also be used to compute Child Security Association Keying Material per RFC 5996 section 2.17.

IKE PRF is a simply a Keyed MAC function -- either one of several selected HMACs based on MD5, SHA-1 or SHA-2; or based on AES-CMAC or AES-XCBC-MAC. As such, it generates a fixed length of material; at least 16 bytes (for MD5, AES-CMAC, and AES-XCBC-MAC); as many as 64 bytes when using SHA-512. IKE PRF+ is a well defined recursive use of PRF. It includes a single byte recursion count such that PRF+ is undefined past the 254th recursion. PRF+ can be used to generate a large pseudo-random byte string, suitable to be used for several keys and / or salting values. The PROTINFO field codes found in Table 7-56 define the cipher suites to be used by the protocol, and SEC's built-in protocol processing sequences handle the remaining details. The protocol permits selected input parameters to be decrypted on the way in, and for results to be encrypted on the way out. By using the IKE PRF function, key material can be generated without allowing any other system resources to have access to unencrypted precursor material.

## 10.1.1 Using IKE PRF to generate SKEYSEED

IKEv2, as described in RFC 5996, specifies generation of a first master key material seed called SKEYSEED:

- SKEYSEED = prf(Ni | Nr, g^ir) for initial setup
- SKEYSEED = prf(SK_d (old), g^ir (new) | Ni | Nr) for rekeying

SKEYSEED is the master seed used by IKE PRF+ to generate IKE SA key material. One of the IKE SA keys generated is subsequently used to generate Child SA key material -- for example, the Child SA material may be for IPsec ESP.

In the initial setup computation of SKEYSEED, Ni and Nr are concatenated to form K -- the key string. Before concatenation, if PDB Option KOV is set, then Ni will be decrypted. The Diffie Hellman shared secret, g^ir, will be decrypted before use as string S if PDB Option IOV is set.

For the rekeying instance, SK_d will be decrypted if PDB Option KOV is set, and g^ir will be decrypted if PDB Option IOV is set. The unencrypted form of g^ir is then concatenated with Ni and Nr to form string S.

## 10.1.2 Using IKE PRF+ to generate keying material for the IKEv2 SA

The IKE PRF+ function can be used as part of IKEv2 to generate IKE key material. The form shown in RFC 5996, is

- {SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr} = PRF+ (SKEYSEED, Ni | Nr | SPIi | SPIr)

In this case, the PRF+ function generates seven separate key blocks. SEC, can generate up to 8 outputs, any of which may be encrypted using the JDKEK. SKEYSEED and Ni may also have been encrypted; SKEYSEED will be decrypted using the JDKEK before use if PDB option KOV is set, and Ni will be decrypted using the JDKEK before use if PDB option IOV is set.

### 10.1.3 Using IKE PRF+ to generate Child SA key material

The IKE PRF+ function can be used as part of IKEv2 to generate Child SA key material. Several forms of this are shown in RFC 5996, including

- KEYMAT = prf+(SK_d, Ni | Nr)
- KEYMAT = prf+(SK_d, g^ir | Ni | Nr)

In both forms, SK_d may have been protected by SEC, using its JDKEK. PDB Option KOV controls whether or not to decrypt SK_d.

In the form with the Diffie-Hellman exchange result, g^ir may have been encrypted using the JDKEK. In the form that excludes the Diffie-Hellman exchange result, Ni might have been encrypted. PDB Option IOV Controls whether or not to decrypt the first segment of string S.

How string KEYMAT is sliced up into keys for child security associations is beyond the scope of RFC 5996. SEC supports splitting this KEYMAT by means of multiple FIFO STORE Commands into up to 8 separate locations as defined by the PDB Output Reference block.

In each case, the resulting PRF output material is passed to another protocol (such as IPsec) for further disposition or is used whole by another element of IKE.

### 10.1.4 Restrictions on programming control blocks

Note the following restrictions on the programming of the control blocks:

- The total length of the key control blocks may not exceed 2046 bytes.
- When not encrypted, the length of either key control block may not exceed 1023 bytes individually.
- Only the first referenced key may be encrypted, and the length of the first key control block when encrypted may not exceed 128 bytes.

- For input material, only the first reference may be encrypted; if encrypted, it is limited to 512 bytes.
- The first two unencrypted referenced inputs may not exceed a total of 512 bytes.
- The remaining three or four unencrypted references may not exceed a total of 512 bytes.

The length of each output segment is limited by the size of the output FIFO. In current designs this is 128 bytes. The total amount of output cannot exceed 512 bytes.

### 10.1.5 IKE PRF PDB format descriptions

Both IKE PRF and PRF+ use the exact same form of PDB. The PDB is used to specify each key block (up to 2), each input block (up to 6), and each output block (up to 8). Restrictions as to length and what may be encrypted are described in Restrictions on programming control blocks. More detail is provided below as to how processing occurs. However, the general idea is that the two Key References can be used to supply up to two separately stored key segments. The two segments are concatenated together and used as a key to the underlying function. The key material fetched from the first address is decrypted before use if PDB option KOV is set. The key material from the second address is never decrypted. In RFC 5996, the resulting concatenated data is called "K."

Similarly, material is fetched from the locations specified by the up-to 6 Input References, the first of which will be decrypted if PDB option IOV is set. All of the fetched data is concatenated to form what RFC 5996 refers to as "S." For output, the PRF-generated string is chopped up into up to 8 different memory locations, as specified by the lengths found in the Output Reference Control Block and the addresses found in the Output Reference Block. Each of the 8 output references controls whether the string segment written to the associated location is encrypted using the KEK in AES-ECB, or using the KEK in AES-CCM.

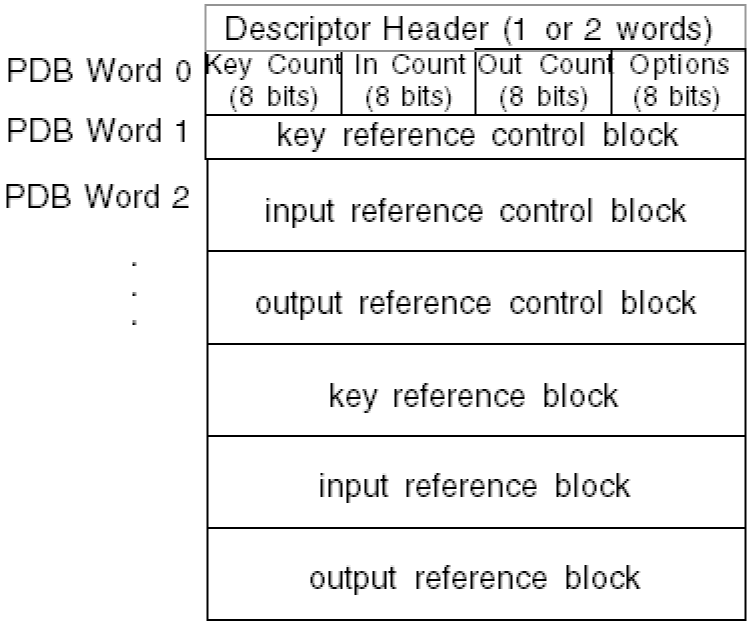


**Figure 10-1. IKE PRF PDB**

### Table 10-1. IKE PRF PDB, description of the PDB

| Field | Description |
|---|---|
| Key Count | Number of key inputs to the PRF function. Valid values are 1 and 2. |
| In Count | Number of inputs to the PRF function. Valid values are 1 - 6. |
| Out Count | Number of outputs from the PRF function. Valid values are 1 - 8. |
| Options | See Table 10-3 |
| Key Reference Control Block | A series of 16-bit fields, called reference controls. There are two Key Reference Controls, one for each Key Reference below. If KOV=1 in the Options field, then the material referenced by the first pointer is considered encrypted, and will be decrypted before use. The ENC, SPLIT and OEKT fields in the Key Reference controls are reserved and must be zero. |
| Input Reference Control Block | A series of 16-bit fields, called Reference controls. There are 6 input reference controls, one for each Input reference. If IOV=1 in the Options field, then the first Input Reference has been encrypted. The ENC, SPLIT and OEKT fields in the Input Reference controls are reserved and must be zero. |
| Output Reference Control Block | A series of 16-bit fields, called reference controls. There are 8 output reference controls, one for each Output Reference. |
| Key Reference Block | Two pointer fields. The width of each field is determined by the PS field of the Master Configuration Register. |
| Input Reference Block | Six pointer fields. The width of each field is determined by the PS field of the Master Configuration Register. |
| Output Reference Block | Eight pointer fields. The width of each field is determined by the PS field of the Master Configuration Register. |

### Table 10-2. IKE PRF PDB, format of the options byte

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | IKEKT | IDEKT | KOV | IOV | Reserved |

### Table 10-3. IKE PRF PDB, description of the options byte

| Field | Description |
|---|---|
| 7-5 | Reserved |
| 4<br>IKEKT | Input Key Encryption Key Type. Note: this field is ignored if keys are not encrypted.<br>0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |
| 3<br>IDEKT | Input Data Encryption Key Type. Note: this field is ignored if data is not encrypted.<br>0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |
| 2<br>KOV | Key input override. If KOV=1, the data referenced by the *first*Key Reference is treated as if it has been encrypted using the Job Descriptor Key Encryption Key.<br>0 Key or keys are not encrypted<br>1 Key Override -- the first referenced key is decrypted before use. |
| 1<br>IOV | Input override. If 1, then the first reference is to a value that has been encrypted using the Job Descriptor Key Encryption Key.<br>0 No encrypted inputs<br>1 Input Override -- one encrypted input is referenced |
| 0 | Reserved |

### Table 10-4. IKE PRF PDB reference controls, format

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SGT | ENC | SPLIT | OEKT | Reserved | | LENGTH | | | | | | | | | |
| Reference Control i | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SGT | ENC | SPLIT | OEKT | Reserved | | LENGTH | | | | | | | | | |
| Reference Control i+1 (if necessary) | | | | | | | | | | | | | | | |

### Table 10-5. IKE PRF PDB reference controls - description of the fields

| Field | Description |
|---|---|
| 31<br>SGT | For Reference Control i, SGT specifies whether the pointer is a direct reference to the data or a pointer to a Scatter/Gather Table.<br>0 Direct reference to data<br>1 Reference to a Scatter/Gather Table |
| 30<br>ENC | (output reference only) - For Reference Control i, ENC=1 specifies that the value is encrypted (with the Job Descriptor Key Encryption Key) before it is written to memory. |
| 29<br>SPLIT | (output reference only) - For Reference Control i, SPLIT=1 specifies that the value is used as an HMAC key and is stored in IPAD/OPAD form. If SPLIT=1, then the HMAC key is encrypted regardless of the ENC value. |
| 28<br>OEKT | (output reference only) - For Reference Control i, OEKT specifies the type of algorithm used for encrypting the output |

*Table continues on the next page...*

Table 10-5. IKE PRF PDB reference controls - description of the fields (continued)

| Field | Description |
|---|---|
| | 0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |
| 27-26 | Reserved |
| 25-16<br>LENGTH | For Reference Control i, LENGTH specifies the number of bytes of data when in plaintext form. Note that if ENC=1, the encrypted form of the data may be longer than the length specified here. |
| 15-0 contain the same fields as bits 31-16, but are used for the next reference. Note that if there is an odd number of input references in the PDB, then the fields in the least-significant half of the PDB word are ignored. | |
| 15<br>SGT | For Reference Control i+1, SGT specifies whether the pointer is a direct reference to the data or a pointer to a Scatter/Gather Table.<br>0 Direct reference to data<br>1 Reference to a Scatter/Gather Table |
| 14<br>ENC | (output reference only) - For Reference Control i+1, ENC =1 specifies that the value is encrypted (with the Job Descriptor Key Encryption Key) before it is written to memory. |
| 13<br>SPLIT | (output reference only) - For Reference Control i+1, SPLIT=1 specifies that the particular value is used as an HMAC key and is to be stored in IPAD/OPAD form. If SPLIT =1, then ENC is ignored. |
| 12<br>OEKT | (output reference only) - For Reference Control i+1, OEKT specifies the type of algorithm used for encrypting the output<br>0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |
| 11-10 | Reserved |
| 9-0<br>LENGTH | For Reference Control i, LENGTH specifies the number of bytes of data when in plaintext form. Note that if ENC=1, the encrypted form of the data may be longer than the length specified here. |

## 10.1.6 Implementation details for IKE PRF function

This figure shows the IKE PRF-material in system memory.

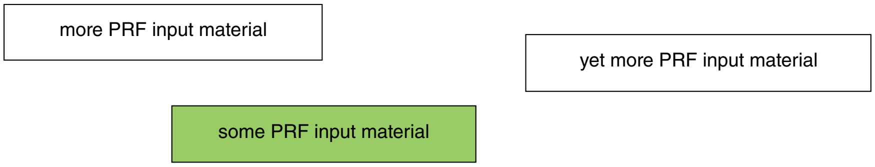


Figure 10-2. IKE PRF-material in system memory

The procedure is as follows:

1. SEC collects the key, as shown in the following figure.

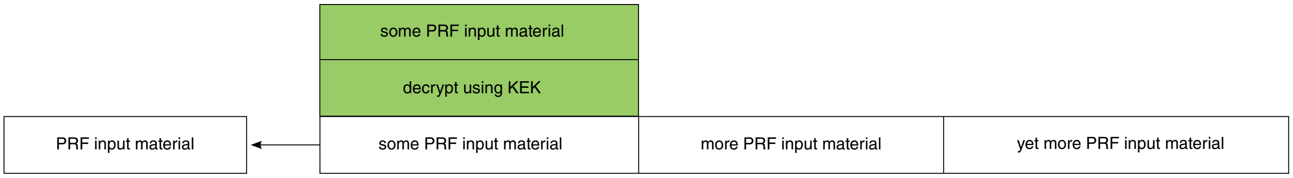


**Figure 10-3. Collecting IKE PRF-material in SEC**

2. If decryption is required, the decryption procedure is applied to the whole first segment.
3. For Key material, up to two segments may be fetched. RFC 5996 refers to the concatenated result as K.
4. The procedure is repeated for input material. Up to six segments may be fetched. RFC 5996 refers to the concatenated result as S.
5. A MAC of the resultant string is then generated.
    - For IKE, legitimate MAC functions include all HMACs supported by MDHA (except SHA-224), as well as AES-CMAC and AES-XCBC-MAC.

## 10.1.7 Implementation Details for IKE PRF+ function

The IKE PRF (Pseudo Random Function) Protocol is a method defined by RFC 5996 for generating session keys using cryptographic algorithms to create pseudo-random data. The SEC's built in protocol processing sequences handle most of the details, so the PROTINFO field codes enumerated in Table 7-56 are used to define the ciphersuites to be used by the Protocol.

The IKE PRF+ is similar to the IKE PRF described in Implementation details for IKE PRF function, with the additional step of adding a byte of counter value to the end of the input material for every recursion of the PRF (see the following figure).

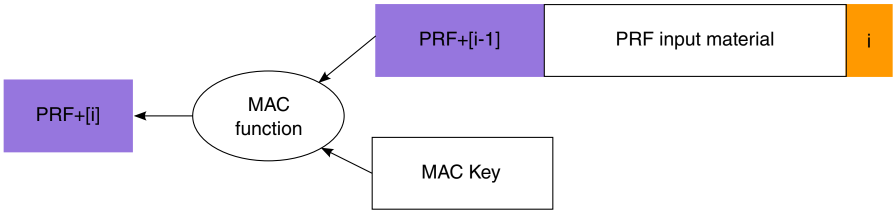


**Figure 10-4. Recursive IKE PRF+ material generation**

SEC uses PRF input material and the HMAC key in the same way as for the IKE PRF. However, it manages the byte counter i. Each PRF+ invocation starts with i = 01h; each iteration increments i by 1.

Valid IKE PRF+ underlying MAC functions include all HMACs supported by MDHA (except SHA-224), AES-CMAC, and AES-XCBC-MAC.

## 10.2 SSL/TLS/DTLS pseudo-random functions (PRF)

The TLS revisions have the following differences in how they define the PRFs.

- SSL 3.0 uses simple hash functions, relying on both MD5 and SHA-1.
- TLS 1.0, TLS 1.1, and DTLS use MD5 and SHA-1, but with HMAC enabled instead of a straight hash.
- TLS 1.2 drops the requirement for any particular hash function and does not combine the output of multiple hash functions to produce the PRF result.

Note that all SSL/TLS/DTLS PRFs use only hash functions supported by MDHA.

See Table 7-55 for a description of the PROTINFO field as used with SSL and TLS PRF Commands.

### 10.2.1 SSL 3.0 PRF overview

This figure shows a functional diagram of SSL 3.0 PRF.

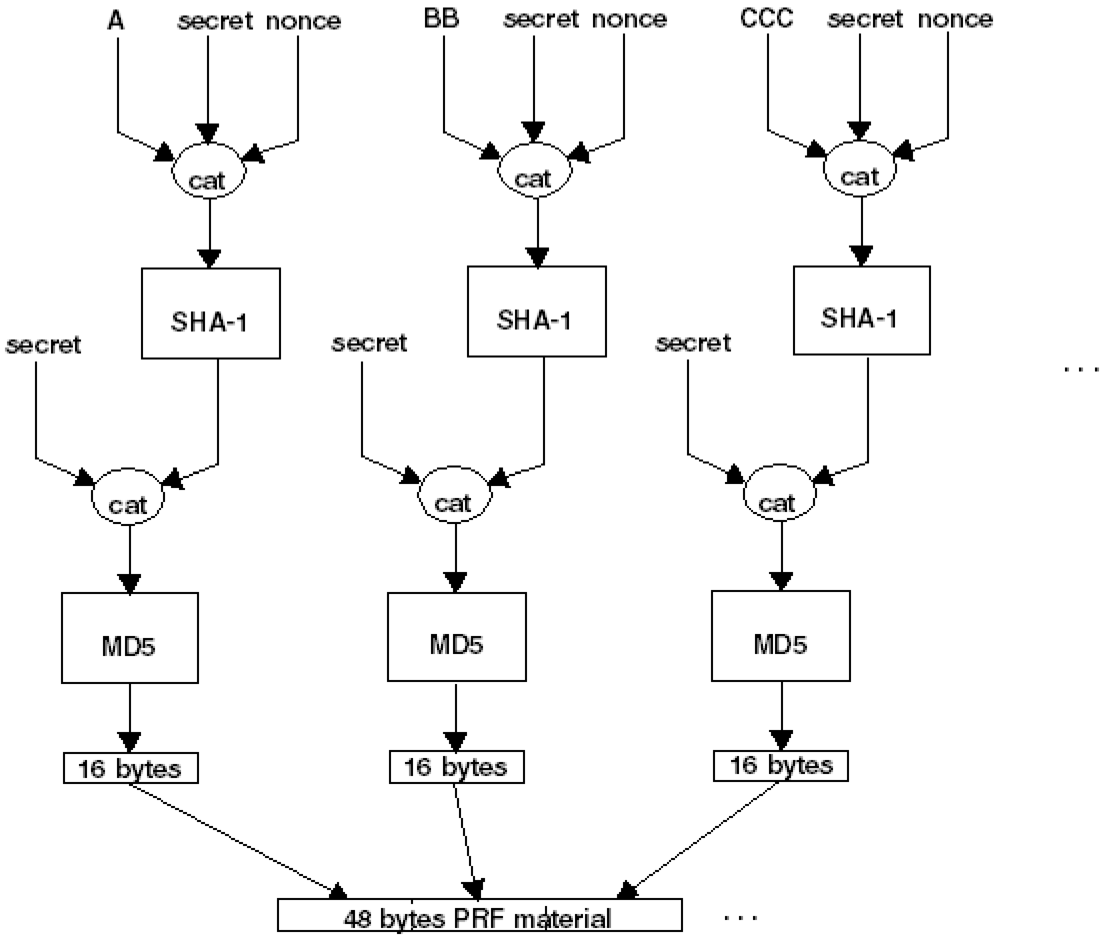


**Figure 10-5. SSL 3.0 PRF**

The SSL 3.0 PRF is referred to as an ad-hoc PRF because it does not use the typical constructs, such as an HMAC, used by other PRFs. Instead, the SSL 3.0 PRF uses straight MD5 and straight SHA-1 to produce PRF material.

### 10.2.1.1 SSL 3.0 PRF definitions

The secret's value is typically the master_secret, which is 48 bytes in length. If the secret's value is the premaster_secret, SSL has no explicit bounds on the length of the premaster_secret. However, SEC does not support a premaster_secret greater than 512 bytes in length.

The seed is composed of two values. Per standards, the values typically used are the 32-byte client_random and server_random values.

- When generating the master_secret value, seed1 is clientHello.random, and seed2 is serverHello.random.

- When generating key material, that is reversed: seed1 is serverHello.random and seed2 is clientHello.random.
- For FINISHED message generation, typical use is that seed1 is a 16-byte MD5 hash of all the handshake messages that the FINISHED message is covering, and seed2 is the 20-byte SHA-1 hash of all the same handshake messages the FINISHED message is covering. Note that the SSL 3.0 PRF function does not support the computation of those hashes.

Whatever the type of seed value, the descriptor PDB can either include the immediate values or can reference them in external memory.

### 10.2.2 Process for SSL 3.0 PRF

For every 16 bytes of PRF material that SEC needs to generate, SEC performs the following actions:

1. Concatenates an iteration constant, a secret, and a nonce
2. Produces a SHA-1 hash
3. Takes the same secret (from the concantenation step) and postpends the SHA-1 hash result to it
4. Produces an MD5 hash of that concatenation.

The descriptor provides the secret to SEC. The secret typically has already been encrypted with the key encryption key. The descriptor also provides the nonce to SEC, but the nonce is not considered sensitive.

SEC iterates to produce as many bytes of PRF material as required. The iteration constant for the first iteration is 41, which is ASCII for "A". Note that with each iteration, both the value and the number of instances of the iteration constant increments; that is the second iteration uses 42, 42 and the seventh iteration uses seven copies of the byte 47.

SEC divides an arbitrary number of bytes of SSL PRF material into up to six distinct memory locations, meaning that the SSL PRF descriptor can have between one and six output pointers for returning PRF material. Note the following:

- Each of these output pointers may be considered sensitive and require encryption of that memory location's PRF material using the key encryption key.

### 10.2.3 SSL 3.0 PRF PDB format descriptions

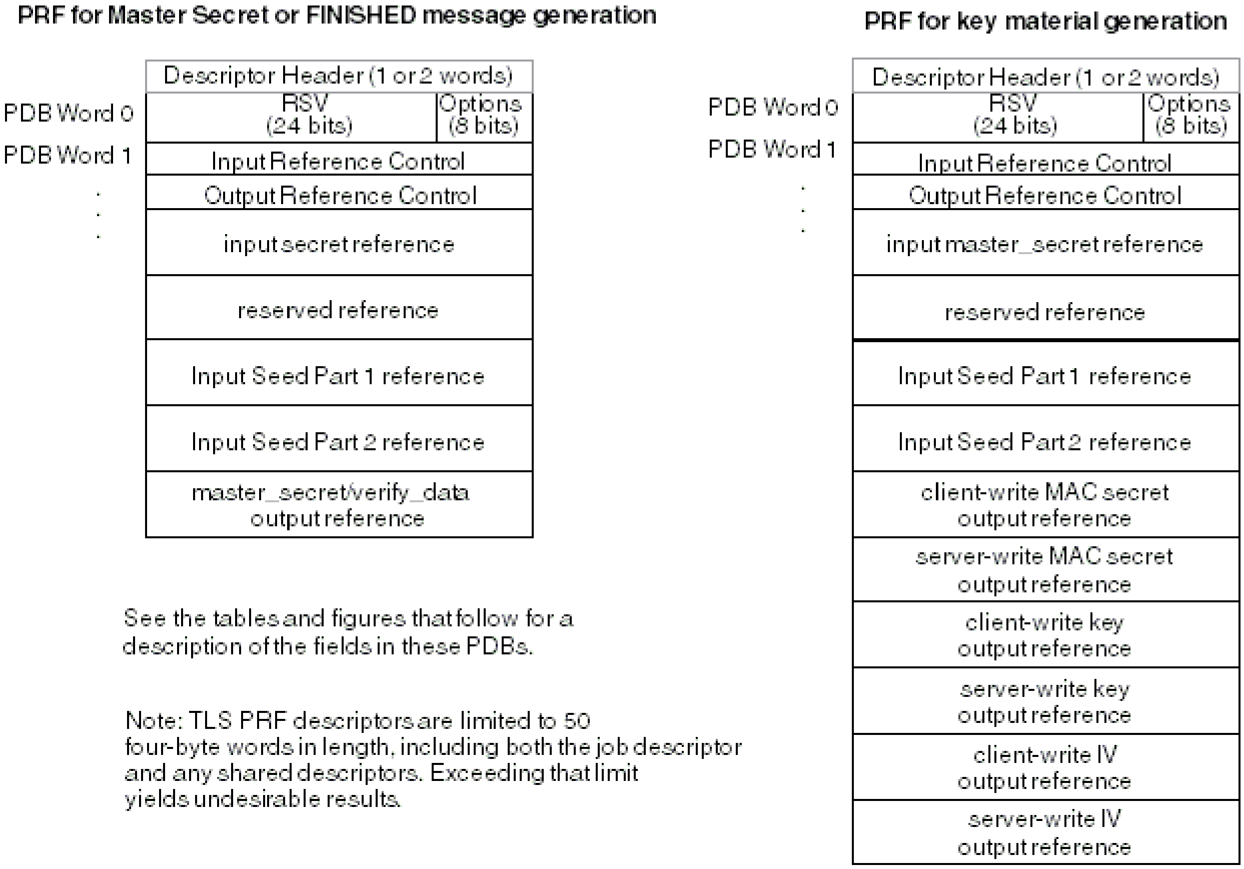

PRF for Master Secret or FINISHED message generation
Descriptor Header (1 or 2 words)
PDB Word 0
RSV (24 bits)
Options (8 bits)
PDB Word 1
Input Reference Control
Output Reference Control
input secret reference
reserved reference
Input Seed Part 1 reference
Input Seed Part 2 reference
master_secret/verify_data output reference
See the tables and figures that follow for a description of the fields in these PDBs.
Note: TLS PRF descriptors are limited to 50 four-byte words in length, including both the job descriptor and any shared descriptors. Exceeding that limit yields undesirable results.
PRF for key material generation
Descriptor Header (1 or 2 words)
PDB Word 0
RSV (24 bits)
Options (8 bits)
PDB Word 1
Input Reference Control
Output Reference Control
input master_secret reference
reserved reference
Input Seed Part 1 reference
Input Seed Part 2 reference
client-write MAC secret output reference
server-write MAC secret output reference
client-write key output reference
server-write key output reference
client-write IV output reference
server-write IV output reference

**Figure 10-6. SSL 3.0 PRF PDB**

**Table 10-6. SSL 3.0 PRF PDB, format of the options byte**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | Reserved | IEKT | OEKT | IEOV | OEOV |

**Table 10-7. SSL 3.0 PRF PDB, description of the options byte**

| Field | Description |
|---|---|
| 7-4 | Reserved |
| 3<br>IEKT | Input Encryption Key Type. Note this field applies only to encrypted inputs.<br>0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |
| 2<br>OEKT | Output Encryption Key Type. Note this field applies only to encrypted outputs.<br>0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |

*Table continues on the next page...*

## Table 10-7. SSL 3.0 PRF PDB, description of the options byte (continued)

| Field | Description |
|---|---|
| 1<br>IEOV | Input Encryption Override<br>1 Master-Secret input is not encrypted.<br>0 Master-Secret input is encrypted. |
| 0<br>OEOV | Output Override default. Note this field is ignored if PROTINFO != FFFF or FFFE<br>1 If PROTINFO=FFFF or FFFE, the generated key material is not encrypted.<br>0 If PROTINFO=FFFF or FFFE, the generated key material is encrypted. |

## Table 10-8. SSL 3.0 PRF PDB input and output reference, format

| |
|---|
| Pointer (32-bit or 64-bit) (See Address Pointers) |

## Table 10-9. SSL 3.0 PRF PDB input and output reference, field descriptions

| Field | Description |
|---|---|
| 31-0 or 63-0<br>Pointer | Pointer to the reference. The size of this field is determined by the PS field of the Master Configuration Register.<br>• If 32 bit addresses are selected, 4 bytes are reserved for this field.<br>• If larger addresses are selected, 8 bytes are reserved for this field. |

## Table 10-10. SSL 3.0 PRF PDB input reference control, format

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | Input Secret Length | | | | | | | | | | Reserved | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | Input Seed Part 1 Length | | | | | | | Input Seed Part 2 Length | | | | | | |

## Table 10-11. SSL 3.0 PRF PDB input reference control, field descriptions

| Field | Description |
|---|---|
| 31 | Reserved |
| 30-21<br>Input Secret Length | Length of the input secret in bytes.<br>**NOTE:** If the PRF material is split into keys, (if PROTINFO != FFFF or FFFE) the master_secret must be 48 bytes, and this field is ignored. |
| 20-14 | Reserved |
| 13-7<br>Input Seed Part 1 Length | Length of the Input Seed Part 1 in bytes. Per protocol definitions, legitimate values are 16 or 32. |
| 6-0<br>Input Seed Part 2 Length | Length of the Input Seed Part 2 in bytes. Per protocol definitions, legitimate values are 20 or 32. |

Table 10-12. SSL 3.0 PRF PDB output reference control, format

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SGT | Reserved | | | | | | | LENGTH | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | | | | | | | | | | | | |

Table 10-13. SSL 3.0 PRF PDB output reference control, format

| Field | Description |
|---|---|
| 31<br>SGT | Specifies whether the pointer is a direct reference to the data or a pointer to a scatter/gather table.<br>0 Direct reference<br>1 Reference to a scatter/gather table |
| 30-24 | Reserved |
| 23-16<br>LENGTH | Length of the data. Ignored if PROTINFO != FFFF or FFFE. In these cases the derived from the cipher suite determined by the PROTINFO field of the OPERATION Command.<br>If PROTINFO = FFFF or FFFE, this field signals the length, in bytes, of the master_secret/verify_data output. |
| 15-0 | Reserved |

## 10.2.4 TLS 1.0/TLS 1.1/DTLS PRF overview

This figure shows a functional diagram of TLS 1.0/TLS 1.1/DTLS PRF.

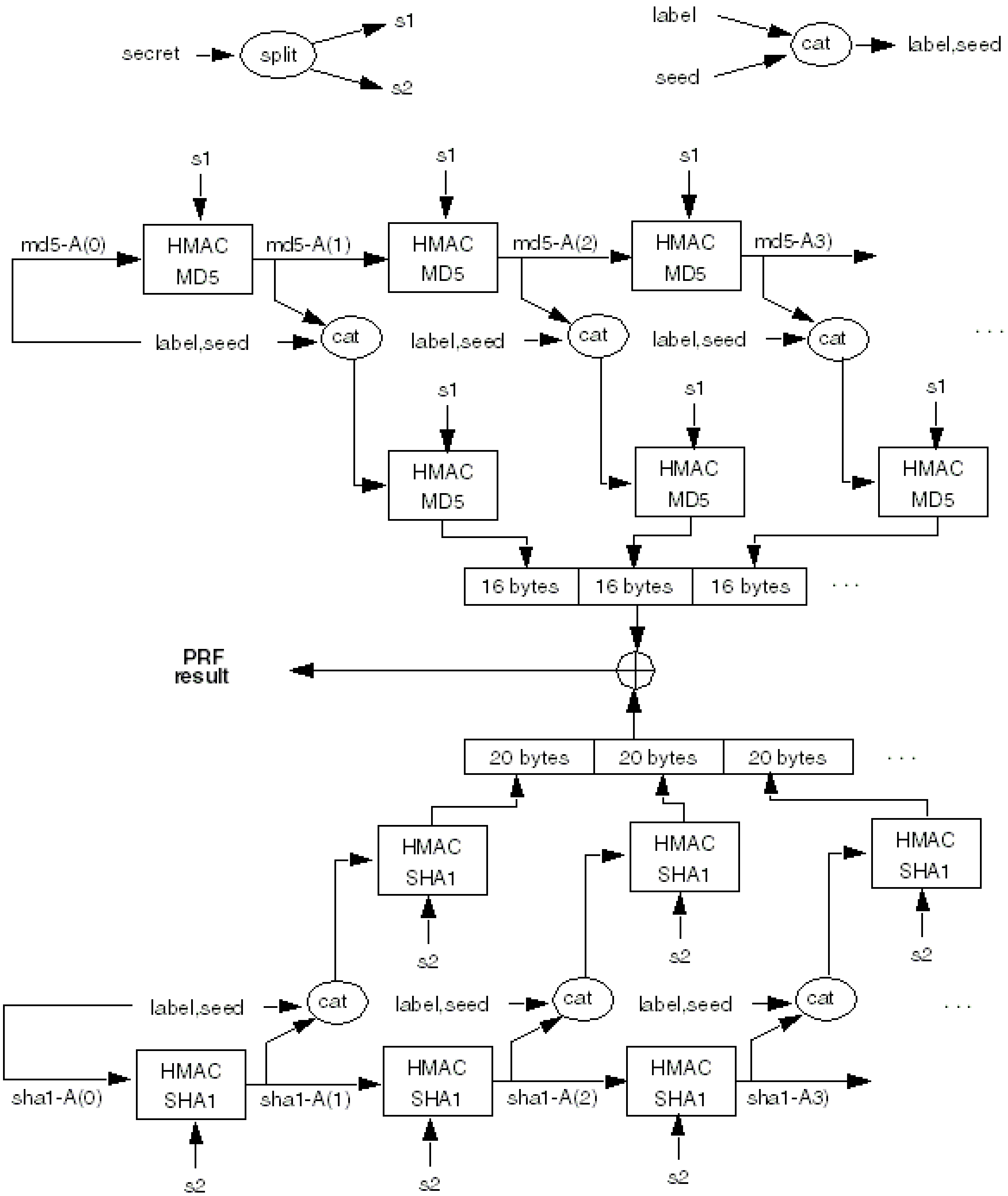


**Figure 10-7. TLS 1.0/TLS 1.1/DTLS PRF**

The TLS 1.0, TLS 1.1, and DTLS PRF uses both HMAC-MD5 and HMAC-SHA-1. SEC implements this PRF with a protocol descriptor that specifies the following inputs: secret, label, and seed. Any of those inputs may be considered sensitive and therefore decrypted using the key-encryption key prior to use.

The seed may be split across multiple memory segments, each individually referenced in the protocol descriptor. Each seed segment referenced in the descriptor is either entirely considered sensitive or not. Each seed segment may be further split in memory using a scatter/gather table. All material referred to by the table must be either completely sensitive or completely not sensitive.

### 10.2.4.1 TLS PRF RFC definitions

RFCs 2246 and 4346 define the TLS PRF as follows:

```
PRF(secret, label, seed) = P_MD5(S1, label + seed) XOR P_SHA-1(S2, label + seed);
P_MD5(secret, seed) = HMAC_MD5(secret, A(1) + seed) + HMAC_MD5(secret, A(2) + seed) + ...
P_SHA-1(secret, seed) = HMAC_SHA-1(secret, A(1) + seed) + HMAC_SHA-1(secret, A(2) + seed)
+ ...

Where for both P_MD5 and P_SHA-1,
A(0) = label + seed
A(i) = HMAC_hash (secret, A(i-1))
```

Note the following:

- The keyword seed is multiply defined, meaning that the value for seed in all subsequent lines takes the meaning that label + seed has in the first line.
- The + operator is defined to mean concatenation (see Figure 10-7 for an illustration). For example:
  - Generating anywhere between 41 and 48 bytes of PRF material requires four iterative executions of P_MD5 and four iterative executions of P_SHA-1, resulting in a total of 14 HMAC computations.
  - Generating between 33 and 40 bytes of PRF material requires four P_MD5 executions, but only three P_SHA-1 executions.
- The secret value is either the master_secret or the premaster_secret.
  - master_secret is typically used and is 48 bytes in length.
  - If the premaster_secret is used, TLS has no explicit bounds on the length, but SEC does not support a premaster_secret greater than 512 bytes in length.
  - If the premaster_secret is longer than 128 bytes, the split secrets s1 and s2 are each larger than the 64-byte maximum permissible HMAC key value, unless preprocessing is performed. SEC performs that preprocessing if required, hashing S1 using MD5 and hashing S2 using SHA-1.
- label is a byte string
  - All valid byte strings identified in standards are less than 16 bytes in length.
  - The descriptor PDB permits the byte string to be either an immediate value in the PDB or to be stored in external memory and be referenced by a pointer.
- The seed is composed of two values. per standards, typical use are the 32-byte client_random and server_random values.

- When generating the master_secret value, seed1 is clientHello.random, and seed2 is serverHello.random.
- When generating key material, that is reversed; seed1 is serverHello.random,and seed2 is clientHello.random.
- For FINISHED message generation, typical use is that seed1 is a 16-byte MD5 hash of all the handshake messages the FINISHED message is covering, and seed2 is the 20-byte SHA-1 hash of all the same handshake messages the FINISHED message is covering. Note that the TLS 1.0/ TLS 1.1/DTLS PRF function does not support the computation of those hashes.
- Whatever the type of seed value, the descriptor PDB can either include the immediate values or can reference them in external memory.

## 10.2.5 Process for TLS 1.0, TLS 1.1, DTLS PRF

SEC splits the secret into two equal portions, entitled s1 and s2.

- s1 is used as the HMAC key for all HMAC-MD5 computations.
- s2 is used as the HMAC key for all HMAC-SHA-1 computations.

If the secret is not even in length, then s1 and s2 both encompass the middle byte. For example, if the secret is 11 bytes:

- s1 and s2 are each 6 bytes in length.
- The middle byte of secret is the last byte of s1 and the first byte of s2.

SEC divides an arbitrary number of bytes of TLS PRF material into up to six distinct memory locations, meaning that the TLS PRF descriptor can have between one and six output pointers for returning PRF material. Note the following:

- Each of these output pointers may be considered sensitive and thus require encryption of that memory location's PRF material using the key encryption key.
- The PRF material generated for any of those memory locations may be prepared for use as an HMAC key.

### 10.2.5.1 How TLS uses PRF material

Typical TLS use of PRF material is to split it the following ways:

- Client-write MAC secret
- Server-write MAC secret
- Client-write key
- Server-write key

- Client-write IV
- Server-write IV

The MAC secrets and keys are sensitive, but the IVs are not.

The MAC secrets are encrypted after they have been expanded into HMAC IPAD and OPAD key material.

If the PRF is being used in this way, the PROTINFO field described in Table 7-55 indicates two things:

- How the material is to be split-for example, HMAC-MD5 keys are 16 bytes long
- How the material is to be treated after it is split-for example, HMAC keys are turned into IPAD/OPAD split keys to maximize performance through the use of these keys

Per the PROTINFO field, keys are encrypted using the key encryption key before being stored in memory.

The PRF can also be used to generate a fixed length value. Per standards, that is done either when generating:

- The master_secret from the premaster_secret
- A FINISHED message

The result of generating the master_secret is a 48-byte value that is encrypted with the key encryption key. The result of generating a FINISHED message is a 12-byte value that is not encrypted with the key encryption key.

### 10.2.5.2 Concatenating input material into one input string (TLS 1.0/1.1/DTLS)

SEC concatenates all input material entitled label and seed into one input string that is persistent throughout the PRF computation.

Using the labels from Figure 10-7 :

- md5-A(0) is the result of an HMAC-MD5 computation over the persistent input string
- sha1-A(0) is the result of an HMAC-SHA-1 computation over the persistent input string.
- md5-A(i) is the result of an HMAC-MD5 computation over md5-A(i-1)
- sha1-A(i) is the result of an HMAC-MD5 computation over sha1-A(i-1).

For every md5-A(i) computed (except for i = 0), md5-A(i) is also prepended to the persistent input string. An HMAC-MD5 computation is performed over the resultant string. This is performed for enough iterations of i to produce the required number of bytes of output.

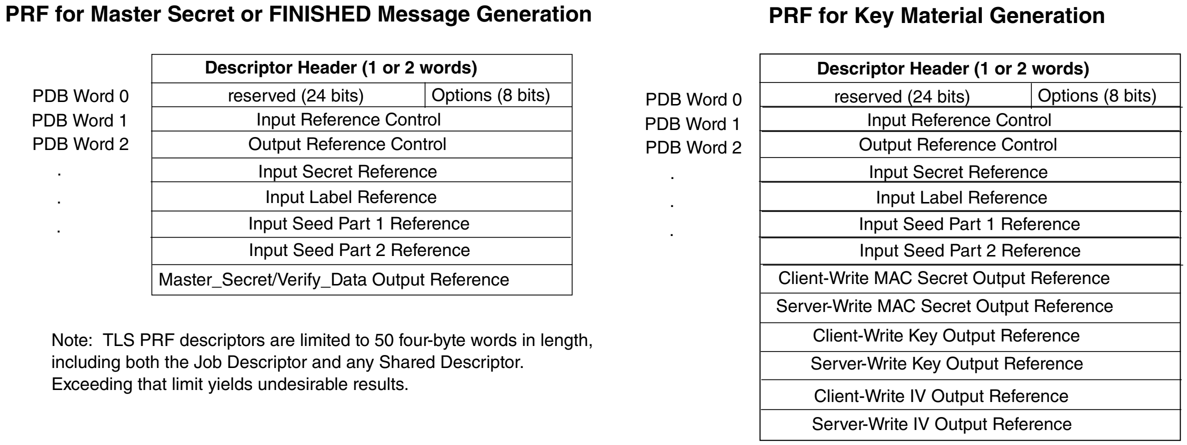
A similar prepending of sha1-A(i) to the persistent input string occurs. An HMAC-SHA-1 computation is performed, and the HMAC-SHA-1 procedure is repeated for enough iterations to produce the required number of bytes of output.

The iterative HMAC-SHA-1 results are concatenated together as are the iterative HMAC-MD5 results. Note that because SHA-1 produces 20-byte results and MD5 produces 16-byte results, there are likely to be more iterations on the MD5 side than on the SHA-1 side.

The PRF material results from performing an XOR of the concatenated MD5 output string with the concatenated SHA-1 output string, truncated to the required length.

### 10.2.6 TLS 1.0, TLS 1.1, DTLS PRF PDB format descriptions

The figure below illustrates the PDB format for the Master Secret, FINISHED and Key Material Generation forms for the TLS 1.0, TLS 1.1 and DTLS PRF protocols.



PRF for Master Secret or FINISHED Message Generation

| | | |
|---|---|---|
| | **Descriptor Header (1 or 2 words)** | |
| PDB Word 0 | reserved (24 bits) | Options (8 bits) |
| PDB Word 1 | Input Reference Control | |
| PDB Word 2 | Output Reference Control | |
| . | Input Secret Reference | |
| . | Input Label Reference | |
| . | Input Seed Part 1 Reference | |
| | Input Seed Part 2 Reference | |
| | Master_Secret/Verify_Data Output Reference | |

Note: TLS PRF descriptors are limited to 50 four-byte words in length, including both the Job Descriptor and any Shared Descriptor. Exceeding that limit yields undesirable results.

PRF for Key Material Generation

| | | |
|---|---|---|
| | **Descriptor Header (1 or 2 words)** | |
| PDB Word 0 | reserved (24 bits) | Options (8 bits) |
| PDB Word 1 | Input Reference Control | |
| PDB Word 2 | Output Reference Control | |
| . | Input Secret Reference | |
| . | Input Label Reference | |
| . | Input Seed Part 1 Reference | |
| | Input Seed Part 2 Reference | |
| | Client-Write MAC Secret Output Reference | |
| | Server-Write MAC Secret Output Reference | |
| | Client-Write Key Output Reference | |
| | Server-Write Key Output Reference | |
| | Client-Write IV Output Reference | |
| | Server-Write IV Output Reference | |

See the tables below for a description of the fields in these PDBs.

Figure 10-8. TLS 1.0/TLS 1.1/DTLS PRF PDB

### Table 10-14. TLS 1.0/TLS 1.1/DTLS PRF PDB, format of the options byte

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | Reserved | IEKT | OEKT | IEOV | OEOV |

### Table 10-15. TLS 1.0/TLS 1.1/DTLS PRF PDB, description of the options byte

| Field | Description |
|---|---|
| 7-4 | Reserved |
| 3<br>IEKT | Input Encryption Key Type. Note this field applies only to encrypted inputs.<br>0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |
| 2<br>OEKT | Output Encryption Key Type. Note this field applies only to encrypted outputs.<br>0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |
| 1<br>IEOV | Input Encryption Override.<br>1 Master-Secret input is not encrypted<br>0 Master-Secret input is encrypted |
| 0<br>OEOV | Output Encryption Override default. Note this field is ignored if PROTINFO != FFFF or FFFE<br>1 If PROTINFO = FFFF or FFFE, the generated key material is not encrypted.<br>0 If PROTINFO = FFFF or FFFE, the generated key material is encrypted. |

### Table 10-16. TLS 1.0 and 1.1/DTLS PRF PDB input and output reference, format

| |
|---|
| Pointer (32-bit or 64-bit) (see Address Pointers) |

### Table 10-17. TLS 1.0 and 1.1/DTLS PRF PDB input and output reference, field descriptions

| Field | Description |
|---|---|
| Pointer | Pointer to the reference<br>The size of this field is determined by the PS field of the Master Configuration Register. If 32 bit addresses are selected, 4 bytes are reserved for this field. If larger addresses are selected, 8 bytes are reserved for this field. |

### Table 10-18. TLS 1.0 and 1.1/DTLS PRF PDB input and output reference, format

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | Input Secret Length | | | | | | | | | | Input Label Length | | | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input Label Length (cont) | | Input Seed Part 1 Length | | | | | | | Input Seed Part 2 Length | | | | | | |

Table 10-19. TLS 1.0 and 1.1/DTLS PRF PDB input and output reference, field descriptions

| Field | Description |
|---|---|
| 31 | Reserved |
| 30-21<br>Input Secret Length | Length of the Input Secret in bytes.<br>**NOTE:** if the PRF material is split into keys, (if PROTINFO != FFFF or FFFE), the master_secret must be 48 bytes, and this field is ignored. |
| 20-14<br>Input Label Length | Length of the Input Label in bytes.<br>Per protocol definitions, legitimate values are between 11 and 15, inclusive. |
| 13-7<br>Input Seed Part 1 Length | Length of the Input Seed Part 1 in bytes.<br>Per protocol definitions, legitimate values are 16 or 32. |
| 6-0<br>Input Seed Part 2 Length | Length of the Input Seed Part 2 in bytes.<br>Per protocol definitions, legitimate values are 20 or 32. |

Table 10-20. TLS 1.0 & 1.1/DTLS PRF PDB output reference control, format

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SGT | Reserved | | | | | | | LENGTH | | | | | | | |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Reserved | | | | | | | | Reserved | | | | | | | |

Table 10-21. TLS 1.0 & 1.1/DTLS PRF PDB output reference control, field descriptions

| Field | Description |
|---|---|
| 31<br>SGT | Specifies whether the pointer is a direct reference to the data or a pointer to a scatter/gather table.<br>0 Direct reference<br>1 Reference to a scatter/gather table |
| 30-24 | Reserved |
| 23-16<br>LENGTH | Length of the data<br>• If PROTINFO = FFFF or FFFE, this field signals the length in bytes of the master_secret/verify_data output.<br>• This field is ignored if PROTINFO != FFFF or FFFE. In these cases the length is derived from the cipher suite determined by the PROTINFO field of the OPERATION Command. |
| 15-0 | Reserved |

## 10.2.7 TLS 1.2 PRF overview

This figure is a functional diagram of TLS 1.2 PRF.

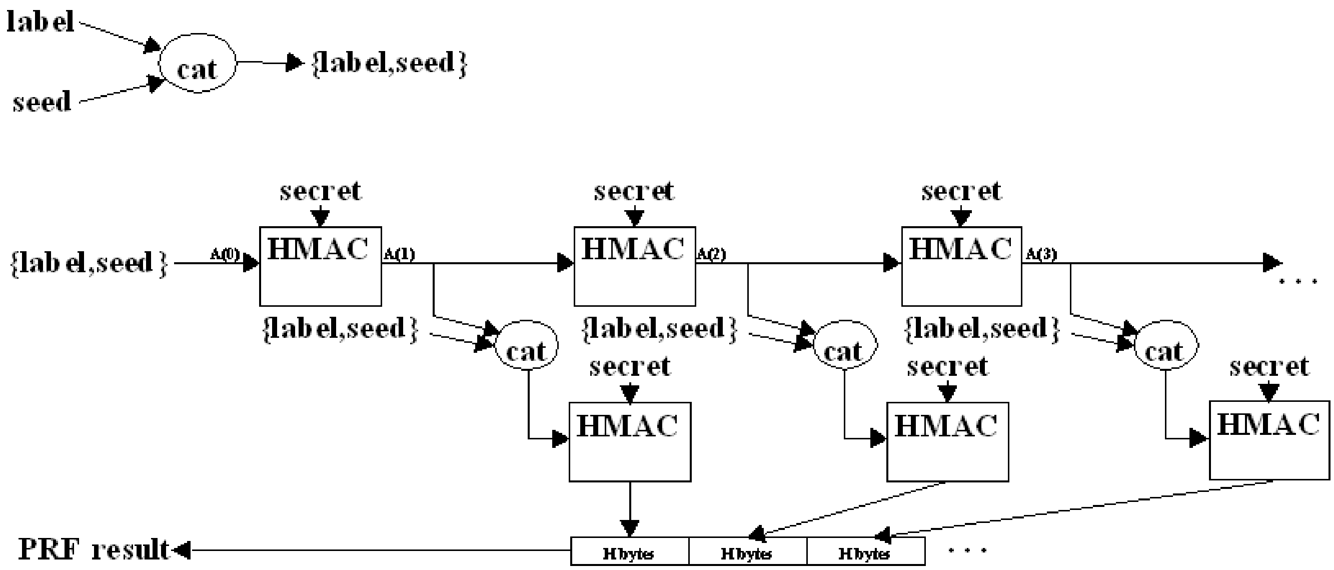


Figure 10-9. TLS 1.2 PRF

The TLS 1.2 PRF uses the same computational structure as the PRF for TLS 1.0, TLS 1.1, and DTLS, but it only uses one underlying hash function for performing the HMAC. The PRF is defined in RFC 5246.

- For all cipher suites supported by TLS 1.1, HMAC-SHA-256 is the underlying hash function used.
- Cipher suites defined under TLS 1.2 are required to explicitly specify a PRF hash function.

Only cipher suites employing HMAC-SHA-256 and HMAC-SHA-384 for PRF are supported for encapsulation and decapsulation.

## 10.2.8 Process for TLS 1.2 PRF

SEC implements this PRF with a protocol descriptor that specifies the following inputs: secret, label, and seed. Any of those inputs may be considered sensitive and so be input in the form of Black Keys. If so, they are decrypted using the key encryption key prior to use.

The seed may be split across multiple memory segments, each referenced by pointers in the protocol descriptor. Each seed segment referenced in the descriptor is either entirely considered sensitive or not. Each seed segment may be further split in memory using a scatter/gather table. Individual segments in a scatter/gather table may not be considered sensitive: either all material referred to by the table is sensitive or it is all not sensitive.

The secret value can be either:

- The master_secret, which is 48 bytes in length and is the value typically used
- The premaster_secret

Although TLS has no explicit bounds on the length of the premaster_secret, SEC does not support a premaster_secret greater than 512 bytes in length. The maximum permissible HMAC key value is:

- 64 bytes for most HMAC algorithms
- 128 bytes for HMAC-SHA-384 and HMAC-SHA-512

If the premaster_secret is longer than this, preprocessing is performed. SEC performs the preprocessing if required, using the specified algorithim to hash the premaster_secret.

### 10.2.8.1 Concantenating input material into one input string (TLS 1.2)

SEC concatenates all input material entitled label and seed into one input string that is persistent throughout the PRF computation.

Referring to the labels in Figure 8-1, there are two lines (or rows) of HMAC computations. The top row keeps producing an HMAC of the previous HMAC result, using secret as the key. That is, A(i) is the result of performing an HMAC of A(i-1). For each of these A(i) values produced (except A(0)), the next D bytes of PRF is the result of performing an HMAC of the persistent input string with the appropriate A(i) value prepended (where D is the size of the digest).

### 10.2.8.2 How TLS uses PRF material (TLS 1.2)

The SEC PRF functions are designed to generate an arbitrary number of bytes of PRF material into an arbitrary number of destinations. Each destination may be designated as secure key material, in which case the result is encrypted using the key encryption key.

Typical TLS use of PRF material is to split it the followingways:

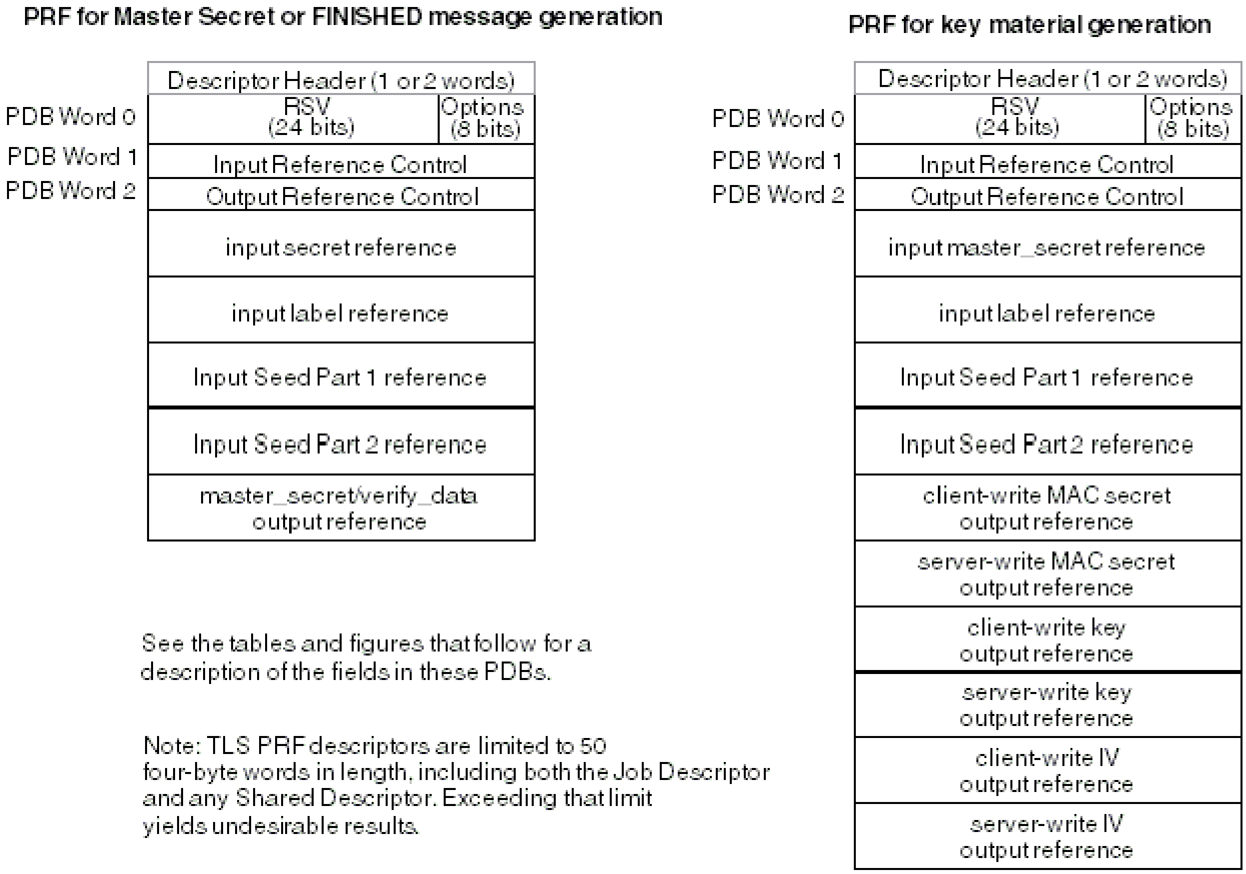
- Client-write MAC secret
- Server-write MAC secret
- Client-write key
- Server-write key
- Client-write IV
- Server-write IV

The MAC secrets and keys are sensitive, but the IVs are not. Typically, the MAC secrets are encrypted after they have been expanded into HMAC IPAD and OPAD key material.

Note that AES-GCM does not make use of a MAC secret. As a result, if an AES-GCM cipher suite is selected, SEC skips the client-write and server-write MAC secrets, regardless of the length specified for each in their output references in the PDB.

Because the PRF is used for other purposes, including generation of a MAC across a series of messages, SEC supports splitting PRF material across between 1 and 8 unique destinations, each of which can enable or disable use of the key encryption key and each of which can enable or disable pre-preparation of HMAC material prior to encryption.

## 10.2.9 TLS 1.2 PRF PDB format descriptions


PRF for Master Secret or FINISHED message generation
Descriptor Header (1 or 2 words)
PDB Word 0
RSV (24 bits)
Options (8 bits)
PDB Word 1
Input Reference Control
PDB Word 2
Output Reference Control
input secret reference
input label reference
Input Seed Part 1 reference
Input Seed Part 2 reference
master_secret/verify_data output reference
PRF for key material generation
Descriptor Header (1 or 2 words)
PDB Word 0
RSV (24 bits)
Options (8 bits)
PDB Word 1
Input Reference Control
PDB Word 2
Output Reference Control
input master_secret reference
input label reference
Input Seed Part 1 reference
Input Seed Part 2 reference
client-write MAC secret output reference
server-write MAC secret output reference
client-write key output reference
server-write key output reference
client-write IV output reference
server-write IV output reference

See the tables and figures that follow for a description of the fields in these PDBs.

Note: TLS PRF descriptors are limited to 50 four-byte words in length, including both the Job Descriptor and any Shared Descriptor. Exceeding that limit yields undesirable results.

**Figure 10-10. TLS 1.2 PRF PDB**

**Table 10-22. TLS 1.2 PRF PDB, format of the options byte**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | Reserved | IEKT | OEKT | IEOV | OEOV |

### Table 10-23. TLS 1.2 PRF PDB, description of the options byte

| Bits | Description |
|---|---|
| 7-4 | Reserved |
| 3<br>IEKT | Input Encryption Key Type. Note that this field applies only to encrypted inputs.<br>0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |
| 2<br>OEKT | Output Encryption Key Type. Note that this field applies only to encrypted outputs.<br>0 AES-ECB-256 encryption<br>1 AES-CCM-256 encryption |
| 1<br>IEOV | Input Encryption Override<br>1 Master-Secret input is not encrypted<br>0 Master-Secret input is encrypted |
| 0<br>OEOV | Output Encryption Override Default. Note this field is ignored if PROTINFO != FFFF or FFFE<br>1 If PROTINFO=FFFF or FFFE, the generated key material is not encrypted.<br>0 If PROTINFO=FFFF or FFFE, the generated key material is encrypted. |

### Table 10-24. TLS 1.2 PRF PDB input and output reference, format

| |
|---|
| Pointer (32-bit or 64-bit) (see Address Pointers) |

### Table 10-25. TLS 1.2 PRF PDB input and output reference, field descriptions

| Field | Description |
|---|---|
| Pointer | Pointer to the reference.<br>The size of this field is determined by the PS field of the Master Configuration Register.<br>• If 32 bit addresses are selected, 4 bytes are reserved for this field.<br>• If addresses larger than 32 bits are selected, 8 bytes are reserved for this field. |

### Table 10-26. TLS 1.2 PRF PDB input reference control, format

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | Input Secret Length | | | | | | | | | | Input Label Length | | | | |
| | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input Label Length (cont) | | Input Seed Part 1 Length | | | | | | | Input Seed Part 2 Length | | | | | | |

### Table 10-27. TLS 1.2 PRF PDB input reference control, field descriptions

| Field | Description |
|---|---|
| 31 | Reserved |

*Table continues on the next page...*

**Table 10-27. TLS 1.2 PRF PDB input reference control, field descriptions (continued)**

| Field | Description |
|---|---|
| 30-21<br>Input Secret Length | Length of the input secret in bytes.<br>**NOTE:** if the PRF material is split into keys, (if PROTINFO != FFFF or FFFE) then the master_secret must be 48 bytes, and this field is ignored. |
| 20-14<br>Input Label Length | Length of the input label in bytes.<br>Per protocol definitions, legitimate values are between 11 and 15, inclusive. |
| 13-7<br>Input Seed Part 1 Length | Length of the Input Seed Part 1 in bytes.<br>Per protocol definitions, legitimate values are 16 or 32. |
| 6-0<br>Input Seed Part 2 Length | Length of the Input Seed Part 2 in bytes.<br>Per protocol definitions, legitimate values are 20 or 32. |

**Table 10-28. TLS 1.2 PRF PDB output reference control, format**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SGT | Reserved | | | | | | | LENGTH | | | | | | | |
| | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | | | | | Reserved | | | | | | | |

**Table 10-29. TLS 1.2 PRF PDB output reference control, field descriptions**

| Field | Description |
|---|---|
| 31<br>SGT | Specifies whether the pointer is a direct reference to the data or a pointer to a scatter/gather table.<br>0: Direct reference<br>1: Reference to a Scatter/Gather Table |
| 30-24 | Reserved |
| 23-16<br>LENGTH | Length of the data.<br>• If PROTINFO = FFFF or FFFE, this field signals the length, in bytes, of the master_secret/verify_data output.<br>• If PROTINFO != FFFF or FFFE, this field is ignored. The length is derived from the cipher suite determined by the PROTINFO field of the OPERATION Command. |
| 15-0 | Reserved |

## 10.3 Implementation of the derived key protocol

This protocol is available to assist with replacing a negotiated key with a derived form of that key. In particular, this protocol can be used for these tasks:

- Compute the IPAD/OPAD split key form of a HMAC key. [see Using the MDHA Key Register with IPAD/OPAD "split keys" for more information]

The use of the derived form of the key is mandatory for bulk-data protocols such as IPsec, where the use of the derived form provides a significant speed advantage.

The derived key protocol (DKP) is designed to allow a negotiated key to be replaced with the derived form in-place in a shared descriptor. For example, an IPsec descriptor can be written to supply an immediate HMAC key in negotiated form as a parameter to the DKP operation command. The DKP computes the IPAD/OPAD "split key" form, leaving the derived key in the Class 2 Key register, available for the subsequent IPsec command. Further, the DKP updates the descriptor, replacing the DKP operation command with the appropriate KEY command, and replacing the negotiated form of the key with the derived form of the key. It is the responsibility of the descriptor author to ensure the resulting derived key will not overwrite any descriptor commands that need to be kept.

### 10.3.1 Using DKP with HMAC keys

When used to generate HMAC keys, DKP receives an unprotected negotiated key and generates an unprotected derived key. If an encrypted split key is desired, or if an encrypted negotiated key is provided, see the FIFO STORE command and Output Data Types 16, 17, 26, 27 in Table 7-31.

When generating derived HMAC keys (also known as "Split Keys"), the four-bit I/O control subfield of the PROTINFO field in the DKP Operation command is split in half; the upper 2 bits define the Input Source, and the lower two bits define the Output Destination. Not all combinations are valid.

Input Source - bits 16-17

**Table 10-30. DKP input destination field**

| Setting | Description |
|---|---|
| 00 | IMM - negotiated key is in words immediately following the DKP Operation Command.<br>This option can only be used with an Immediate Output Destination (OD=00). |
| 01 | SEQ - negotiated key is found in the input frame as defined by the SEQ IN PTR command. This must be the choice when DKP is used in a trusted descriptor. |
| 10 | PTR - the input key is referenced by the address found immediately following the DKP Operation Command. |
| 11 | SGF - the input key is distributed amongst different memory locations as indicated by the Scatter/Gather Table address found immediately following the DKP Operation Command. |

Output Destination - bits 18-19

Table 10-31. DKP output destination field

| Setting | Description |
|---|---|
| 00 | IMM - resulting derived HMAC "split key" will be written back to the descriptor, immediately after the KEY command written to the descriptor, consuming as many words as required. The contents of those words will be overwritten and will not be preserved. The length of the resulting derived HMAC key is twice the underlying hash context length. See Table 10-32<br>Note that IMM is not restricted when used as an Output Destination as it is when used as an Input Source. |
| 01 | SEQ - the resulting derived HMAC "split key" will be written to the output frame as defined by the SEQ OUT PTR command. Note that SEQ is a valid Output Destination only when SEQ is provided as an Input Source. This must be the choice when DKP is used in a trusted descriptor. |
| 10 | PTR - the resulting derived HMAC "split key" will be written back to the memory location specified by the address found immediately after the DKP Operation Command. This option is not valid with Input Source options IMM or SGF. |
| 11 | SGF - the resulting derived HMAC "split key" will be written back to memory per the scatter/gather table found at the address immediately following the DKP operation command. This option is not valid with Input Source options IMM or PTR. |

The twelve-bit length field designates the number of bytes the negotiated key takes. The length of the derived "split" key is determined by the underlying hash function chosen, as shown.

Table 10-32. HMAC derived key lengths

| Hashing algorithm | Length of derived "split" key |
|---|---|
| MD5 | 32 bytes / 8 words |
| SHA-1 | 40 bytes / 10 words |
| SHA-224<br>SHA-256 | 64 bytes / 16 words |
| SHA-384<br>SHA-512 | 128 bytes / 32 words |

## 10.3.2 Implementation of the Blob Protocol

The blob protocol provides a method for cryptographically protecting the confidentiality and integrity of user data across SoC power cycles. The data to be protected is encrypted so that it can be safely placed into non-volatile storage before the SoC is powered down. The key used to encrypt the blob is derived from a non-volatile master secret key so the blob can be decrypted when the SoC powers up again. More details on the Blob protocol can be found in section Blobs

# Chapter 11
# Cryptographic hardware accelerators (CHAs)

This section describes the functionality of each individual CHA used by the DECOs.

### Table 11-1. Summary of cryptographic hardware accelerators (CHAs)

| Definition | Abbreviation | What it implements | Cross-reference |
|---|---|---|---|
| Public-key hardware accelerator | PKHA | RSA, Diffie-Hellman, DSA, Elliptic-Curve Diffie-Hellman, Elliptic-Curve DSA | Public-key hardware accelerator (PKHA) functionality |
| Kasumi f8 and f9 hardware accelerator | KFHA | The Kasumi f8 encryption and Kasumi f9 authentication algorithms | Kasumi f8 and f9 hardware accelerator(KFHA) functionality |
| Data encryption standard accelerator | DESA | The DES and Triple-DES encryption algorithms | Data encryption standard accelerator (DES) functionality |
| Cyclic-redundancy check accelerator | CRCA | The coudble-CRC authentication algorithm | Cyclic-redundancy check accelerator (CRCA) functionality |
| Random number generator | RNG | A true hardware random number generator and a pseudo-random number generator | Random-number generator (RNG) functionality |
| SNOW 3G f8 accelerator | SNOWf8 | The SNOW f8 encryption algorithm | SNOW 3G f8 accelerator functionality |
| SNOW 3G f9 accelerator | SNOWf9 | The SNOW f9 authentication algorithm | SNOW 3G f9 accelerator functionality |
| Message-digest hardware accelerator | MDHA | The MD-5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512SHA-512/224, SHA-512/256 authentication algorithms | Message digest hardware accelerator (MDHA) functionality |
| AES accelerator | AESA | The AES encryption algorithm | AES accelerator (AESA) functionality |

*Table continues on the next page...*

Table 11-1. Summary of cryptographic hardware accelerators (CHAs) (continued)

| Definition | Abbreviation | What it implements | Cross-reference |
|---|---|---|---|
| ZUC encryption accelerator | ZUCE | The ZUC encryption algorithm | ZUC encryption accelerator (ZUCE) functionality |
| ZUC authentication accelerator | ZUCA | The ZUC authentication algorithm | ZUC authentication accelerator (ZUCA) functionality |

## 11.1 Public-key hardware accelerator (PKHA) functionality

The PKHA module is capable of performing a number of different operations used in public-key cryptography, including modular arithmetic functions such as addition, subtraction, multiplication, exponentiation, reduction, squaring, cubing, simultaneous exponentiation, and inversion. All of these functions are provided in both integer and polynomial-binary field versions, except modular subtraction, which is the same as addition for binary polynomials. There are also elliptic-curve functions for point addition, point doubling, point validation, and point multiplication the standard prime and binary curves. Most of these functions can be performed timing-equalized to thwart timing-related side-channel attacks. PKHA also includes a Miller-Rabin primality test function for detecting prime numbers.

The PKHA internally performs modular multiply operations using "Montgomery multiplication". For efficiency, many of these functions have a variant which allows either inputs or outputs in Montgomery form. Some have variants to supply the Montgomery conversion factor. These save time over the variations without. Internally, the PKHA operates on digits of these values. Different versions of the PKHA may have a different digit size. This PKHA has a digit size of 32 bits. This has implications for the inputs and outputs of certain functions. See the discussion on Montgomery arithmetic.

Because the numbers used in public-key cryptography are typically quite large and often referenced many times during a function, the inputs to PKHA are loaded into registers. PKHA has four of these labeled A, B, E, and N. A and B are for operands and results. E is for "keys", and N holds the modulus. For ECC functions, A and B are divided up into equal-size quadrants to accommodate the greater number of inputs required.

PKHA also has two other types of functions for manipulating the data in the registers. These are the Clear Memory and Copy Memory functions. The Clear Memory function allows all or any combination of the registers to be overwritten with zeros. The Copy Memory functions can be used to copy data from any of the A, B or N registers or register quadrant to any register A, B, E or N.

PKHA requires that all data for a given function be loaded before the Mode Register is written to invoke a function. This convention indicates to PKHA that all needed data has been loaded, and the function can now be launched. The typical procedure for executing a PKHA function is to use KEY and FIFO LOAD Commands to load the registers (usually N first), followed by an OPERATION Command to write the Mode Register, followed by one or more FIFO STORE Commands to store the result. PKHA functions may also be cascaded, so that the output of one function stays in a PKHA memory to become an input for the next function.

When loading or storing a value, it is important that its associated size register not change during the operation. To help avoid this issue when loading ECC parameters, make sure that all quadrants of a given register have the same size values by left-filling "short" values with zero. If a size register for a FIFOLOAD command mayl change before it is complete, it is necessary to cause the Descriptor to stall until safe to proceed: insert a JUMP Command before offending command: JUMP jsl = 1 type = 0 cond = nifp offset = 1 (instruction 0xA1000401). In the other case, where a FIFOSTORE may still be in progress when a subsequent command will change the value in its size register, insert a SEQ FIFOSTORE Command before the offending command: SEQ FIFOSTORE length=0 (Instruction 0x68000000).

### 11.1.1 Modular math

Almost all math operations require with a modulus value in the N Memory. Math operations involving multiplication (multiplication, exponentiation, prime test, and ECC functions) are performed internally using Montgomery values.

### 11.1.2 About Montgomery values

The PKHA contains a Modular Arithmetic Unit. Multiplication is always modular multiplication:

$A * B \bmod N$.

The PKHA performs this computation with a Montgomery multiplier. A Montgomery multiplier can be more efficient than a multiply-then-reduce calculation because the modular reduction is done as part of the multiplication and the working product never gets larger than the modulus. In a normal multiplication, the product, before reduction, would be the size of the sum of the factors, so usually twice the size of the modulus. The factors in a Montgomery multiplication each have an R factor, and, as part of the multiplication and modular reduction, one R is removed. Thus, the computation performed is:

$(AR * BR) / R$ mod $N$.

The equivalent of "division by $R$" occurs even if one of the inputs does not have an $R$ factor.

A number of PKHA functions accept inputs in Montgomery form instead of normal values. Some instead take $R^2$ mod $N$ as an input. These functions can be faster than their normal-value alternatives if several operations are performed in a row or if these values are known in advance. This is because, before being used, ($R^2$ mod $N$ needs to be computed and) normal values need to be converted internally to Montgomery form.

The Montgomery form of a value is *value*R* mod $N$, referred to here as $\underline{\textit{value}}$. The term R = $2^{SD}$ is the Montgomery factor, where D is the digit size (of a digit in the PKHA arithmetic unit), in bits, and S is the minimum number of digits needed to hold the value in N. R is therefore dependent on N and D.

To use the PKHA to convert a normal value to a Montgomery value, one must first compute (or know) $R^2$ mod $N$, the Montgomery Conversion Factor. The following steps can be used to convert a value from a normal value into its Montgomery form (A and B inputs may be reversed):

$R^2$ = MOD_R2($N$)

$\underline{A}$ = MOD_MUL_IM_OM($A$, B=$R^2$, $N$)

The equivalent F2M function can be used for binary polynomial values.

Eventually, the value needs to be converted out of Montgomery form. This can be done by performing another multiply ($R^2$ is not needed for this).

```
A = MOD_MUL_IM_OM(A=A, B=1, N)
```

Another method is to cause the PKHA to perform a multiplication and conversion to normal form. Internally, there are two multiplications: first the two inputs, then the product by one.

```
AB mod N = MOD_MUL_IM(A=A, B=B, N)
```

A third method is to have just one factor (either one) in Montgomery form:

```
AB mod N = MOD_MUL_IM_OM(A=A, B=B, N)
```

The following operations can be used to convert a value from a normal value into its Montgomery form (A and B inputs may be reversed):

The equivalent F2M functions can be used for binary polynomial values.

It is possible to add and subtract Montgomery values, if *R* mod *N* is the same. Do not mix and match Montgomery and normal values for addition or subtraction. 5 + 3R => *5/R + 3R* or *5/R + 3*; neither is likely the desired result.

### 11.1.3 Non-modular Math

Although addition, subtraction, and multiplication functions require a modulus, it is possible to perform these calculations without any reduction: the modulus must be larger than the expected result.

For addition and subtraction, this is easily done. For multiplication, the MOD_MUL function may be used, but it is not the most efficient, as internally first $R^2$ mod N will be computed, then two multiplications will be performed (first to convert one factor into Montgomery, then to compute the product, not in Montgomery).

For non-modular multiplication, MUL_IM_OM is much more efficient, as only one multiplication will be performed. This can be used if the factors are not in Montgomery form, i.e., if the product to be calculated is *A*B* instead of *A*B mod N*. Since the multiplier always "divides by *R*', a special modulus value in Nram is required which will make *R* have the value 1. This is done by creating a modulus *N = R-1* so that *R mod N* will have the value one. This way, normal values are the same as Montgomery values; no conversion is necessary and the multiplier will quietly "divide by one" to no effect.

As an example, on a PKHA with a digit size of 32 bits and a product which will be no more than six bytes long, $R = 2^{SD} = 2^{2*32} = 2^{64}$. Therefore the modulus must be 0xFFFFFFFFFFFFFFFF.

For computation with binary polynomials, the equivalent F2M functions may be used.

### 11.1.4 Elliptic-Curve Math

The PKHA provides point math operations on different types of elliptic curves. These include the ability to add two points (+ operator), double a point, and multiply a point by an integer (scalar) value (x operator).

The input points are assumed to be valid points on the curve. If non-point coordinates or invalid curve parameters are used an input, then a non-point set of coordinates are likely to be returned as output. The "ECC Point Check" functions may be used to verify that a point's (x,y) values constitute a point which satisfies the equation for the curve.

The minimum modulus is 1 byte. The maximum modulus is 1024 bits, 128 in length, or one quadrant.

If xx is the Point at Infinity, *P*and *Q* are points on the curve, and *j* and *k* are integers, then the following identities, as well as others easily derived by taking advantage of associative and commutive properites, hold:

- $P + Q = Q + P$
- $P = xx + P$
- $xx = 0 \, x \, P$
- $(j + k)P = (j \, x \, P) + (k \, x \, P)$

There may be times when the negative of a point is necessary:

- When subtracting points $P_A$ - $P_B$
- When multiplying by a negative integer: -abs(k) x $P_A$

To subtract, one can negate the second term and perform an addition, i.e.

$P_C = P_A - P_B = P_A + (-P_B)$

When multiplying by a negative value, one can either negate the starting point or the ending point. The multiplication value is the absolute value of the scalar, i.e., when *k* is negative

$P_C = k \text{ x } P_A = -\text{abs}(k) \text{ x } P_A = \text{abs}(k) \text{ x } (-1P_A) = -(\text{abs}(k) \text{ x } P_A)$

### 11.1.4.1 Point math over a prime field ($F_p$)

The ECC_MOD family of functions perform Add, Double, and scalar Multiply operations on points on a curve defined by the equation:

$E\!: y^2 = x^3 + ax + b \text{ mod } p$

where p is the a prime integer > 3. These operations are available in Affine Coordinates ($x$,$y$).

The modulus (value in N memory) for these operations is *p*, also referred to as *q*.

The equality for the negative of a point *P*, in affine coordinates, is $-P = -(x,y) = (x, -y)$

The operations will not provide useful outputs if the inputs are not valid points on the curve, i.e., if they are not solutions to the curve equation E.

The point at infinity is a possible result for point math operations. The PIZ bit in Operation Status Register can be used to determine when the result of an operation is the point at infinity.

The representation of the point at infinity, in affine coordinates, depends upon the type of curve and the value of the *b* term of the curve's equation:

- Where $b$ is equal to 0: (0, 1)
- Where $b$ is not equal to 0: (0, 0)

The (X,Y,Z) projective coordinates are Jacobian projective coordinates, as defined in IEEE Std 1363-2000, § A.9.6. Some variants of elliptic curve operations which allow for their input and output.

The conversions between the affine and projective versions of a point are:

- Affine to Jacobian projective: $(x, y) => (x, y, 1)$
- Jacobian projective to Affine: $(X, Y, Z) >= (X/Z^2, Y/Z^3)$ for $Z$ not equal to 0. $Z$ equal to 0 is interpreted as the point at infinity.

The curve equation is $E: Y^2 = X^3 + aXZ^4 + bZ^6$

- the equality for the negative of a point is $(X, -Y, Z) = (X, Y, Z)$

### 11.1.4.2 Point math over a binary field ($F_{2^m}$)

The ECC_F2M family of functions perform add, double, and scalar multiply operations on points on a curve defined by the equation:

E: $y^2 + xy = x^3 + ax^2 + b$

These operations are available in Affine Coordinates $(x,y)$. All inputs and output values of polynomial values are in polynomial basis. For example, $x^5+x+1$ is represented as 23h

The modulus (value in N memory) for these functions is $q$, the field-defining irreducible polynomial for the curve. Other documents use other symbols, including $p(t)$, $f(t)$, and $f$.

The equality for the negative of a point, in affine coordinates, is $-P = -(x,y) = (x, x+y)$.

The operations will not provide useful outputs if the inputs are not valid points on the curve, i.e., if they are not solutions to the curve equation E.

Because of the way the point operations are performed over a binary field, these functions require as an input the value $c$ rather than $b$. The relationship between these two values is:

$b = c^4 \bmod q$

and

$c = b^{2^{m-2}} \bmod q$, where $m$ is the degree (the power of its highest-power term) of $q$.

This *c* value is referred to as *b'* in the ECC Public-Key protocols for ECDSA sign, and so on. The calculation of *c* is expensive, so it is obviously an advantage to calculate it only once or have it precomputed. See Special values for common ECC domains for these values for common ECC domains.

The point at infinity is a possible result for point math operations. The PIZ bit in Operation Status Register can be used to determine when the result of an operation is the point at infinity.

The representation of the point at infinity, in affine coordinates, is:

- (0, 0)

The (X,Y,Z) projective coordinates are Jacobian projective coordinates, as defined in IEEE Std 1363-2000, § A.9.6. Some variants of elliptic curve operations which allow for their input and output.

The conversions between the affine and projective versions of a point are:

- Affine to Jacobian projective: $(x, y) => (x, y, 1)$
- Jacobian projective to Affine: $(X, Y, Z) >= (X/Z^2, Y/Z^3)$ for $Z$ not equal to 0. $Z$ equal to 0 is nterpreted as the point at infinity.

For $F_{2^m}$ curves,

- the curve equation is $E: Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6$
- the equality for the negative of a point is $(X, X + Y, Z) = (X, Y, Z)$

All computations must be performed with the proper MOD or F2M operation, with the curve's modulus as the modulus of the function.

### 11.1.4.3 About Jacobian projective coordinates

Jacobian projective coordinates are used as in IEEE Std 1363-2000, § A.9.6. There are variants of elliptic curve operations which allow for their input and output.They are used internally for computation.

The conversions between the affine and projective versions of a point are:

- Affine to Jacobian projective: $(x, y) => (x, y, 1)$
- Jacobian projective to Affine: $(X, Y, Z) >= (X/Z^2, Y/Z^3)$ for $Z$ not equal to 0. $Z$ equal to 0 is interpreted as the point at infinity.

For $F_p$ curves,

- the curve equation is $E: Y^2 = X^3 + aXZ^4 + bZ^6$

- the equality for the negative of a point is *(X, -Y, Z) = (X, Y, Z)*

For $F_{2^m}$ curves,

- the curve equation is *E:* $Y^2 + XYZ = X^3 + aX2Z^2 + bZ^6$
- the equality for the negative of a point is *(X, X + Y, Z) = (X, Y, Z)*

All computations must be performed with the proper MOD or F2M operation, with the curve's modulus as the modulus of the function.

### 11.1.4.4 About the Point at Infinity

The point at infinity is a possible result for point math operations. Knowing its representation is important for programming, though the PIZ bit in Operation Status Register can be used to determine when the result of an operation is the point at infinity.

The representation of the point at infinity, in affine coordinates, depends upon the type of curve and the value of the *b* term of the curve's equation:

- For $F_p$ curves, where *b* is equal to 0: (0, 1)
- For $F_p$ and $F_{2^m}$ curves, where *b* is not equal to 0: (0, 0)

## 11.1.5 PKHA Mode Register

The formats of the PKHA Mode Register are described in detail in PKHA OPERATION command.

The following tables list the valid PKHA_MODE values for all PKHA functions:

PKHA Clear Memory Functions: Table 11-3

PKHA Modular Arithmetic functions: Table 7-75

PKHA Elliptic Curve functions: PKHA OPERATION: Elliptic Curve Functions

PKHA Copy Memory functions: Table 7-80

**NOTE**

Use of any PKHA_MODE value not listed in these tables results in an invalid mode error.

### 11.1.6 PKHA functions

The various PKHA functions are described in the following subsections. The following information applies to all PKHA functions.

- Mode Register bits that may be either 1's or 0's for the given function are represented with *x*.
- For convenience, in all the descriptions below the output is shown as the default B, although the actual output destination can be specified for most functions via the Class 1 Mode Register[OutSel] field to be either the B RAM or the A RAM.
- For each PKHA function, the specified mode bits are in the Class 1 Mode Register[PKHA_MODE_LS] field.
- For all of the PKHA functions, the Class 1 Mode Register[PKHA_MODE_MS] field is set to 8h.
- The descriptions specify the output register(s) and any other registers that might be modified. Note that the default output register is still modified but the output is placed into the specified destination register(s).
- Note that any parameter underlined is in Montgomery form (for example, $\underline{A}$ = AR mod N).
- Errors reported by PKHA are written to the Job Ring Output Status Register and termination status word (Job termination status/error codes). They are encoded in the ERRID field.
- Three flags in the CCB Status Register may be set by PKHA: PIZ, PIO, and PRM. These flags can be tested by the JUMP (HALT) command. is set to indicate that PKHA generated a result equal to zero, or, in the case of ECC functions, the point at infinity.
- PIO is set whenever a GCD routine finds that the Greatest Common Denominator of two numbers is the number 1. For other general non-ECC functions, it means that the result is equal to one. This may also be referred to as the GCD flag.
- PRM is set by the PRIME_TEST routine if it finds that a candidate integer is probably prime (that is, passes the Miller-Rabin primality test).
- It is important to note that the PKHA mathematical functions work in terms of "digits"; that is, the arithmetic unit is pipelined to work on a digit of data at a time. For PKHA-32 a digit = 32 bits (4 bytes) of data, for PKHA-64 a digit = 64 bits (8 bytes), and for PKHA-128 a digit = 128 bits (16 bytes). Therefore, the term 'digit' refers to 32, 64, or 128 bits of data in the input and/or output values used by the PKHA arithmetic unit.

### 11.1.6.1 Clear Memory (CLEAR_MEMORY) function

This function clears the specified registers or quadrants of registers in the PKHA. This includes the A, B, N and E. All registers or quadrants of registers are written with zeros.

A detailed description may be found in PKHA OPERATION: clear memory function.

**Table 11-2. CLEAR_MEMORY function properties**

| Property | Notes |
|---|---|
| Mode value | *ABEN*_0000_00 *QQ_QQ* 00_0001, with the following restrictions on the combinations of ABEN and QQQQ: At least one of ABEN must be on. If E is on, all Q must be zero. Some example encodings are in the table below. |
| Input | None |
| Output | A = 0, B = 0, E = 0, N = 0, or some quadrant(s) thereof, as specified by ABEN and QQQQ. Each *Q* specifies a quadrant, in order from 3 through 0. |
| Requirements | The Mode Register specifies which registers or quadrants of registers to clear.<br>If no quadrants are selected, then all quadrants of the specified register(s) are cleared. |
| Side effects | None |
| Errors reported | Invalid Mode, if no registers are selected, or E with one or more quadrants is selected |
| Flags set | None |

**Table 11-3. Example mode values for PKHA clear memory functions**

| Function name | Register selects | | | | Quadrant selects | | | | Brief description | Bits 19-0, including PKHA_MODE and reserved bits[1] (Hex) |
|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | E | N | 3 | 2 | 1 | 0 | | |
| CLEAR_MEMORY | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | Clear registers A, B, E, N | F0001 |
| | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | Clear registers A, B, E | E0001 |
| | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | Clear registers A, B, N | D0001 |
| | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | Clear registers A, E | A0001 |
| | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | Clear registers A, N | 90001 |
| | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | Clear registers B, N | 50001 |
| | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | Clear register B | 40001 |
| | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | Clear register E | 20001 |
| | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | Clear register N | 10001 |
| | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | Clear quadrants 2 and 3 of register A | 80301 |
| | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | Clear quadrant 0 of registers B, N | 50041 |
| | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | Clear quadrant 3 of registers A, B | C0201 |

[1] PKHA_MODE_MS concatenated with 0h concatenated with PKHA_MODE_LS

### 11.1.6.2 Integer Modular Addition (MOD_ADD) function

**Table 11-4. MOD_ADD function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_0010 ( output placed in B )<br>0000_0000_000***1***_0000_0010 ( output placed in A ) |
| Input | • N = modulus and data size, any integer<br>• A = first addend, any integer less than N<br>• B = second addend, any integer less than N |
| Output | B (or A, if selected) = (A + B) mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and B are each < N. |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N.<br>If (A + B) >= N, N will be subtracted just once from the sum. That is, if (A + B) >= 2N, then the result will not be mod N. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.3 Integer Modular Subtraction (MOD_SUB_1) function

Modular subtraction can be described as follows. If A ≥ B or A = B = 0, then B = A - B. Otherwise, if A < B, then B = A + N - B. The result is always positive and less than N.

**Table 11-5. MOD_SUB_1 function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_0011 ( output placed in B )<br>0000_0000_000***1***_0000_0011 ( output placed in A ) |
| Input | • N = modulus, any integer<br>• A = minuend, any integer less than N<br>• B = subtrahend, any integer less than N |
| Output | B (or A, if selected) = (A - B) mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and B are less than N. |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N. |

*Table continues on the next page...*

**Table 11-5. MOD_SUB_1 function properties (continued)**

| Property | Notes |
|---|---|
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.4 Integer Modular Subtraction (MOD_SUB_2) function

**Table 11-6. MOD_SUB_2 function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_0100 ( output placed in B )<br>0000_0000_000***1***_0000_0100 ( output placed in A ) |
| Input | • N = modulus, any integer<br>• B = minuend, any integer less than or equal to N<br>• A = subtrahend, any integer less than or equal to N |
| Output | B (or A, if selected) = (B - A) mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and B are < N |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.5 Integer Modular Multiplication (MOD_MUL)

The (AB) mod N computation is provided to assist in algorithms and protocols where a single modular multiplication is required and not as a chaining of multiplications. In the latter case, Montgomery form multiplication routines (that is, MOD_MUL_IM or MOD_MUL_IM_OM) are more efficient. This function first computes $R^2$ mod N, then multiples one factor to produce AR, then multiplies AR*B to produce AB.

**Table 11-7. MOD_MUL function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_0101 ( output placed in B )<br>0000_0000_000***1***_0000_0101 ( output placed in A ) |
| Input | • N = modulus, any odd integer |

*Table continues on the next page...*

Table 11-7. MOD_MUL function properties (continued)

| Property | Notes |
|---|---|
| | • A = multiplicand, any integer less than N<br>• B = multiplier, any integer less than N |
| Output | B (or A, if selected) = (AxB) mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and B are < N. |
| Side effects | A and E are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N.<br>• Divide-By-Zero Error is set if the most significant digit of the modulus is all zeros. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.6 Integer Modular Multiplication with Montgomery Inputs (MOD_MUL_IM)

This function takes its inputs, integers, in Montgomery form, multiplies them modulo the value in the N register and returns the result as a field value. To do this, it performs two multiplications: AR*BR => ABR and ABR*1 => AB.

Table 11-8. MOD_MUL_IM function properties

| Property | Notes |
|---|---|
| Mode value | 1000_0000_000***0***_0000_0101 ( output placed in B )<br>1000_0000_000***1***_0000_0101 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A = multiplicand, a value in Montgomery form<br>• B = multiplier, a value in Montgomery form |
| Output | B (or A if selected) = A x B mod N, the non-Montgomery product of the inputs |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and B are less than modulus N |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

## 11.1.6.7 Integer Modular Multiplication with Montgomery Inputs and Outputs (MOD_MUL_IM_OM) Function

This function performs the calculation A*B/R mod N, where R is the Montgomery factor for N. This can be used in several ways:

- If one value is a normal value, and the other is R2 mod N, then the result is the normal value converted to Montgomery.
- If A and B are both Montgomery values, then the result is the product of A and B as a Montgomery value.
- If only one of (A,B) is a Montgomery value, then the result is the product as a normal value.
- If one of (A,B) is a Montgomery value and the other is the value one, then the result is Montgomery value converted to a normal value.

**Table 11-9. MOD_MUL_IM_OM function properties**

| Property | Notes |
|---|---|
| Mode value | 1100_0000_000***0***_0000_0101 ( output placed in B )<br>1100_0000_000***1***_0000_0101 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A = multiplicand, a value in Mongomery format 0 ≤ A < N<br>• B = multiplier, a value in Mongomery format 0 ≤ B < N |
| Output | B (or A, if selected) = (AxB) mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N. |
| Flags set | PIZ is set if the result is zero. |

## 11.1.6.8 Integer Modular Exponentiation (MOD_EXP and MOD_EXP_TEQ)

This function is commonly used to perform a single-step RSA operation. It computes $R^2$ mod N and converts A to Montgomery form before beginning the exponentiation. MOD_EXP_TEQ performs the same operation as MOD_EXP but with an added timing equalization security feature. The exponentiation run-time of MOD_EXP_TEQ, for a given modulus and size of exponent, is constant. In general MOD_EXP will run faster than MOD_EXP_TEQ, but will never run slower.

**Table 11-10. MOD_EXP and MOD_EXP_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for MOD_EXP | 0000_0000_0**0**0***0***_0000_0110 ( output placed in B )<br>0000_0000_0**0**0***1***_0000_0110 ( output placed in A ) |
| Mode value for MOD_EXP_TEQ | 0000_0000_0**1**0***0***_0000_0110 ( output placed in B )<br>0000_0000_0**1**0***1***_0000_0110 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A = an integer 0 ≤ A < N<br>• E = exponent, any integer |
| Output | B (or A, if selected) = $(A^E)$ mod N, a an integer 0 ≤ A < N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• Maximum key (exponent) size = 512> bytes<br>• A < N |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero. |

#### 11.1.6.9 Integer Modular Exponentiation, Montgomery Input (MOD_EXP_IM and MOD_EXP_IM_TEQ) Function

This function is commonly used to perform a single-step RSA operation. It computes $\underline{A}$ = AR (mod N), where R is the Montgomery constant). The input data (base) to be exponentiated must be provided in the Montgomery form. The result will be returned in normal integer (non-Montgomery) representation. MOD_EXP_IM_TEQ performs the same operation as MOD_EXP_IM but with an added timing equalization security feature. The exponentiation run-time of MOD_EXP_IM_TEQ is constant for a given modulus and size of exponent. In general MOD_EXP_IM will run faster than MOD_EXP_IM_TEQ, but will never run slower.

**Table 11-11. MOD_EXP_IM and MOD_EXP_IM_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for MOD_EXP_IM | 1000_0000_0**0**0***0***_0000_0110 ( output placed in B )<br>1000_0000_0**0**0***1***_0000_0110 ( output placed in A ) |
| Mode value for MOD_EXP_IM_TEQ | 1000_0000_0**1**0***0***_0000_0110 ( output placed in B )<br>1000_0000_0**1**0***1***_0000_0110 ( output placed in A ) |
| Input | • N = modulus, any odd integer |

*Table continues on the next page...*

**Table 11-11. MOD_EXP_IM and MOD_EXP_IM_TEQ function properties (continued)**

| Property | Notes |
|---|---|
| | • A = a value $0 \le A < N$, in Montgomery form<br>• E = exponent, any integer (normal integer representation) |
| Output | B (or A, if selected) = $(A^E)$ mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• Maximum key (exponent) size = 512 bytes<br>• A < N |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.10 Integer Simultaneous Modular Exponentiation (MOD_SML_EXP)

MOD_SML_EXP performs two modular exponentiations and multiplies the results. This is faster than doing them separately. It is useful for DSA Verification.

**Table 11-12. MOD_SML_EXP function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0001_0110 ( output placed in B )<br>0000_0000_000***1***_0001_0110 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A0 = an integer < N, first base<br>• E = an integer, first exponent<br>• A2 = an integer < N, second base<br>• B = an integer, second exponent |
| Output | B (or A if selected) = $A0^E * A2^B$ mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 1/2 RAM size<br>• Maximum key (exponent) size and B size = 1/4 RAM size<br>• A0 < N<br>• A2 < N<br>• The values in A0 and A2 must be the same number of bytes, matching the A SIZE register, and should be the same size as N |
| Side effects | A, B, and E are modified. |
| Errors reported | • N size error is set if N size > half RAM<br>• A size error is set if A size > half RAM |

*Table continues on the next page...*

Table 11-12. MOD_SML_EXP function properties (continued)

| Property | Notes |
|---|---|
| | • B size error is set if B size > quarter-RAM<br>• E size error is set if E size > quarter-RAM |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.11 Integer Modular Square (MOD_SQR and MOD_SQR_TEQ)

This function may be used to square an integer value. MOD_SQR_TEQ, the timing equalized version, will take the same time to complete for a given modulus. In general the MOD_SQR version will run faster than MOD_SQR_TEQ, but will never run slower.

Table 11-13. MOD_SQR and MOD_SQR_EXP function properties

| Property | Notes |
|---|---|
| Mode value for MOD_SQR | 0000_0000_0**0**0***0***_0001_1010 ( output placed in B )<br>0000_0000_0**0**0***1***_0001_1010 ( output placed in A ) |
| Mode value for MOD_SQR_TEQ | 0000_0000_0**1**0***0***_0001_1010 ( output placed in B )<br>0000_0000_0**1**0***1***_0001_1010 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A = a field element |
| Output | B (or A, if selected) = (A*A) mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

### 11.1.6.12 Integer Modular Square, Montgomery inputs (MOD_SQR_IM and MOD_SQR_IM_TEQ)

This function may be used to square an integer value. For a given modulus, MOD_SQR_IM_TEQ will take the same time to complete for any value of A. In general MOD_SQR_IM will run faster than MOD_SQR_IM_TEQ, and will never run slower.

**Table 11-14. MOD_SQR_IM and MOD_SQR_IM_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for MOD_SQR_IM | 1000_0000_0**0*0***_0001_1010 ( output placed in B )<br>1000_0000_0**0*1***_0001_1010 ( output placed in A ) |
| Mode value for MOD_SQR_IM_TEQ | 1000_0000_0**1*0***_0001_1010 ( output placed in B )<br>1000_0000_0**1*1***_0001_1010 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A = an integer < N in Montgomery form |
| Output | B (or A, if selected) = (A*A) mod N, an integer |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

### 11.1.6.13 Integer Modular Square, Montgomery inputs and outputs (MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ)

This function may be used to square an integer value. For a given modulus, MOD_SQR_IM_OM_TEQ will take the same time to complete for any value of A. In general MOD_SQR_IM_OM will run faster than MOD_SQR_IM_OM_TEQ, and will never run slower.

**Table 11-15. MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for MOD_SQR_IM_OM | 1100_0000_0**0*0***_0001_1010 ( output placed in B )<br>1100_0000_0**0*1***_0001_1010 ( output placed in A ) |
| Mode value for MOD_SQR_IM_OM_TEQ | 1100_0000_0**1*0***_0001_1010 ( output placed in B )<br>1100_0000_0**1*1***_0001_1010 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A = a Montgomery value < N |
| Output | B (or A, if selected) = (A*A) mod N, in Montgomery form |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = >Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |

*Table continues on the next page...*

### Table 11-15. MOD_SQR_IM_OM and MOD_SQR_IM_OM_TEQ function properties (continued)

| Property | Notes |
|---|---|
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

## 11.1.6.14 Integer Modular Cube (MOD_CUBE and MOD_CUBE_TEQ)

This function may be used to cube an integer value. These functions first compute the Montgomery conversion factor, $R^2 \mod N$ and then convert the value to cube. For a given modulus, MOD_CUBE_TEQ will take the same time to complete for any value of A. In general MOD_CUBE will run faster than MOD_CUBE_TEQ, and will never run slower.

### Table 11-16. MOD_CUBE and MOD_CUBE_TEQ function properties

| Property | Notes |
|---|---|
| Mode value for MOD_CUBE | 0000_0000_0**0**0***0***_0001_1011 ( output placed in B )<br>0000_0000_0**0**0***1***_0001_1011 ( output placed in A ) |
| Mode value for MOD_CUBE_TEQ | 0000_0000_0**1**0***0***_0001_1011 ( output placed in B )<br>0000_0000_0**1**0***1***_0001_1011 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A = a value < N |
| Output | B (or A, if selected) = (A*A*A) mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

#### 11.1.6.15 Integer Modular Cube, Montgomery input (MOD_CUBE_IM and MOD_CUBE_IM_TEQ)

MOD_CUBE_IM is used to cube an integer value. The timing equalized version, MOD_CUBE_IM_TEQ, also cubes an integer value but will take the same time to complete for a given modulus. In general MOD_CUBE_IM will run faster than MOD_CUBE_IM_TEQ, but will never run slower.

**Table 11-17. MOD_CUBE_IM and MOD_CUBE_IM_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for MOD_CUBE_IM | 1000_0000_0**0**0***0***_0001_1011 ( output placed in B )<br>1000_0000_0**0**0***1***_0001_1011 ( output placed in A ) |
| Mode value for MOD_CUBE_IM_TEQ | 1000_0000_0**1**0***0***_0001_1011 ( output placed in B )<br>1000_0000_0**1**0***1***_0001_1011 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A = a Montgomery value |
| Output | B (or A, if selected) = (A*A*A) mod N, a normal value |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

#### 11.1.6.16 Integer Modular Cube, Montgomery input and output (MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ)

MOD_CUBE_IM_OM is used to cube an integer value. The timing equalized version, MOD_CUBE_IM_OM_TEQ, also cubes an integer value but will take the same time to complete for a given modulus. In general MOD_CUBE_IM_OM will run faster than MOD_CUBE_IM_OM_TEQ, but will never run slower.

**Table 11-18. MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for MOD_CUBE_IM_OM | 1100_0000_0**0**0***0***_0001_1011 ( output placed in B )<br>1100_0000_0**0**0***1***_0001_1011 ( output placed in A ) |

*Table continues on the next page...*

### Table 11-18. MOD_CUBE_IM_OM and MOD_CUBE_IM_OM_TEQ function properties (continued)

| Property | Notes |
|---|---|
| Mode value for MOD_CUBE_IM_OM_TEQ | 1100_0000_0**10*0***_0001_1011 ( output placed in B )<br>1100_0000_0**10*1***_0001_1011 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A = a Montgomery value |
| Output | B (or A, if selected) = (A*A*A) mod N, a Montgomery value |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

## 11.1.6.17 Integer Modular Square Root (MOD_SQRT)

The modular square root function computes output result B, such that ( B x B ) mod N = input A. If no such B result exists, the result will be set to 0 and the PKHA "prime" flag will be set. Input values A and B are limited to a maximum size of 128 bytes. Note that two such square root values may exist. This algorithm will find either one of them, if any exist. The second possible square root (B') can be found by calculating B' = N - B.

### Table 11-19. MOD_SQRT function properties

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0001_0111 ( output placed in B )<br>0000_0000_000***1***_0001_0111 ( output placed in A ) |
| Input | • N0 (N RAM, 1st quadrant) = modulus, any odd integer<br>• A0 (A RAM, 1st quadrant) = input value, for which a square root is to be calculated |
| Output | B0 (or A0, if selected) is calculated such that ( B0 x B0 ) mod N = A mod N. If no such B exists, then B is set to 0. |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and B are < N. |
| Side effects | All of the following PKHA RAM quadrants are modified: E2, E3, N1, N2, N3, B1, B2, B3, A1, A2, A3. |
| Errors reported | N Size Error is set if the size of N is greater than 128. |
| Flags set | PRM is set if no square root solution can be found. |

### 11.1.6.18 Integer Modulo Reduction (MOD_AMODN)

MOD_AMODN computes the remainder of A divided by N. A and N can be of any size and it is not required that A > N, but N must be non-zero.

**Table 11-20. MOD_AMODN function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_0111 ( output placed in B )<br>0000_0000_000***1***_0000_0111 ( output placed in A ) |
| Input | • N = modulus, any non-zero integer<br>• A = any integer |
| Output | B (or A, if selected) = A mod N, A reduced modulo N |
| Requirements | • N = non-zero value<br>• Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Divide By Zero Error is set if N = 0. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.19 Integer Modular Inversion (MOD_INV)

MOD_INV computes the inverse of A, if an inverse exists. If the modulus, N, is prime, then all values of A, 1 ≤ A < N, are guaranteed to have an inverse mod N. If N is not prime, A may or may not have an inverse. It will have one only if GCD(A, N) == 1.

**Table 11-21. MOD_INV function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_1000 ( output placed in B )<br>0000_0000_000***1***_0000_1000 ( output placed in A ) |
| Input | • N = modulus, any non-zero integer<br>• A = any non-zero integer lass than N |
| Output | B (or A, if selected) = $A^{-1}$ mod N, an integer, the multiplicative inverse of A |
| Requirements | • Neither A or N can be zero.<br>• A must be less than N.<br>• Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | A and E are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512. |

*Table continues on the next page...*

**Table 11-21. MOD_INV function properties (continued)**

| Property | Notes |
|---|---|
| | • A Size Error is set if the size of A is greater than size of N.<br>• Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0, or if there is no inverse. |
| Flags set | None |

### 11.1.6.20 Integer Montgomery Factor Computation (MOD_R2)

This function is used to compute a constant to assist in converting operands into the Montgomery residue system representation. The constant $R^2$(mod N) is dependent upon the digit size of the PKHA and the value in N.

MUL, EXP, and ECC functions that do not have "IM" (Montgomery inputs) or an R2 input will internally invoke this routine to determine the constant and do the conversions before other operations.

If the modulus N is a protocol- or system-wide parameter that does not change frequently, such as in ECC operations for a specific curve, save this computed constant, because this routine takes a not-insignificant amount of time to complete.

**Table 11-22. MOD_R2 function properties**

| Property | Notes |
|---|---|
| Mode value | 1000_0000_000***0***_0000_1100 ( output placed in B )<br>1000_0000_000***1***_0000_1100 ( output placed in A ) |
| Input | N = modulus, any odd integer |
| Output | B (or A, if selected) = R2 mod N, where R = $2^{SD}$ where S is size of a digit in bits and D is the number of digits of N; in other words, D = ceiling [sizeof(N) in bits / S] |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even. |
| Flags set | None |

### 11.1.6.21 Integer Greatest Common Divisor (MOD_GCD)

MOD_GCD finds the greatest common divisor of two integers.

**Table 11-23. MOD_GCD function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_1110 ( output placed in B )<br>0000_0000_000***1***_0000_1110 ( output placed in A ) |
| Input | • N = any integer. The most-significant digit of N must be non-zero.<br>• A = any integer less than or equal to N |
| Output | B (or A, if selected) = GCD(A,N), an integer less than or equal to A that divides both A and N<br>If the output is placed in B, the MOD_INV result is available in A. |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and N may not both be zero |
| Side effects | A is modified |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• A Size Error is set if the size of A is greater than size of N.<br>• Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0. |
| Flags set | PIO is set if the result is 1. |

## 11.1.6.22 Miller_Rabin Primality Test (PRIME_TEST)

**Table 11-24. PRIME_TEST function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_1111 ( output placed in B )<br>0000_0000_000***1***_0000_1111 ( output placed in A ) |
| Input | • N[1] = Candidate prime integer<br>• A = An initial random seed for the base value of exponentiation; can be any integer 2 < A < N - 2<br>• B = "t" parameter, which is the number of trial runs. By default, it is set at 1 or B[7:0], whichever is greater. Only the lowest byte of the supplied value is used. |
| Output | B (or A, if selected) = 1 if candidate is believed to be prime, otherwise 0 |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and N may not both be zero |
| Side effects | N and A are modified |
| Errors reported | • N Size Error is set if size of N = 0 or size of N > 512.<br>• B Size Error is set if N size > 256. and the least-significant byte of B > 31.<br>• Divide-By-Zero Error is set if no seed can be found that is in the legal range of 2 < A < N-2. This occurs if N = 1 or N = 3. |
| Flags set | PRM is set if the candidate is believed to be prime |

[1] If the most significant digit of N is zero, the result is always composite, the output is the value zero, and the PRM flag is not set, regardless of the primality of the value of N.

### 11.1.6.23 Binary Polynomial ($F_{2^m}$) Addition (F2M_ADD) function

This function performs binary polynomial modular addition without any modulo reduction, as the value in the N register is ignored. Only its size is used, to determine the size of the result.

This type of addition is the equivalent of a bitwise XOR and this function may be used for that purpose.

This function could as easily be labeled F2M_SUB, as it is mathematically equivalent.

**Table 11-25. F2M_ADD function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_0010 ( output placed in B )<br>0000_0000_000***1***_0000_0010 ( output placed in A ) |
| Input | • Size of N (modulus size)<br>• A = first addend, a binary polynomial<br>• B = second addend, a binary polynomial |
| Output | B (or A, if selected) = A xor B, a binary polynomial |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• The N need not be written, because its contents are ignored, but the size of N must be written. This size is needed because inputs A and B are considered binary polynomials modulo some irreducible polynomial N. |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.24 Binary Polynomial ($F_{2^m}$) Modular Multiplication (F2M_MUL)

The (AB) mod N computation is provided to assist in algorithms and protocols where a single modular multiplication is required and not as a chaining of multiplications. In the latter case, Montgomery form multiplication routines (that is, F2M_MUL_IM or F2M_MUL_IM_OM) are more efficient. This function first computes $R^2$ mod N, then multiples $A*R^2$ to produce AR, then multiplies AR*B to produce AB.

**Table 11-26. F2M_MUL function properties**

| Property | Notes |
|---|---|
| Mode value | 0010_0000_000***0***_0000_0101 ( output placed in B )<br>0010_0000_000***1***_0000_0101 ( output placed in A ) |

*Table continues on the next page...*

**Table 11-26. F2M_MUL function properties (continued)**

| Property | Notes |
|---|---|
| Input | • N = modulus, an irreducible polynomial<br>• A = multiplicand, a field element<br>• B = multiplier, a field element |
| Output | B (or A, if selected) = (AB) mod N, a field element |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and B are field elements modulo N. |
| Side effects | A and E are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.25 Binary Polynomial ($F_{2^m}$) Modular Multiplication with Montgomery Inputs (F2M_MUL_IM) Function

This function takes its inputs, binary polynomials, in Montgomery form, multiplies them modulo the value in the N register, used as a reduction polynomial, and returns the result as a field value. To do this, it performs two multiplications: AR*BR => ABR and ABR*1 => AB.

**Table 11-27. F2M_MUL_IM function properties**

| Property | Notes |
|---|---|
| Mode value | 1010_0000_000***0***_0000_0101 ( output placed in B )<br>1010_0000_000***1***_0000_0101 ( output placed in A ) |
| Input | • N = modulus, an "odd" binary polynomial of order m<br>• A = multiplicand, a binary polynomial < $2^m$, in Montgomery form<br>• B = multiplicand, a binary polynomial < $2^m$, in Montgomery form |
| Output | B (or A, if selected) = (AxB) mod N, a a binary polynomial < $2^m$, non-Montgomery form |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and B are field elements in Montgomery form and must be modulo reduced by irreducible polynomial N. |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.26 Binary Polynomial ($F_{2^m}$) Modular Multiplication with Montgomery Inputs and Outputs (F2M_MUL_IM_OM) Function

This function performs the calculation A*B/R mod N, where R is the Montgomery factor for N. This can be used in several ways:

- If one value is a normal value, and the other is R2 mod N, then the result is the normal value converted to Montgomery.
- If A and B are both Montgomery values, then the result is the product of A and B as a Montgomery value.
- If only one of (A,B) is a Montgomery value, then the result is the product as a normal value.
- If one of (A,B) is a Montgomery value and the other is the value one, then the result is Montgomery value converted to a normal value.

**Table 11-28. F2M_MUL_IM_OM function properties**

| Property | Notes |
|---|---|
| Mode value | 1110_0000_000***0***_0000_0101 ( output placed in B )<br>1110_0000_000***1***_0000_0101 ( output placed in A ) |
| Input | • N = modulus, an "odd" binary polynomial of order m<br>• A = a binary polynomial 0 < $2^m$, in Montgomery form.<br>• B = a binary polynomial 0 < $2^m$, in Montgomery form. |
| Output | B (or A, if selected) = (AxB) mod N, in Montgomery form |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and B are field elements in Montgomery form. |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error is set if the size of B is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.27 Binary Polynomial ($F_{2^m}$) Modular Exponentiation (F2M_EXP and F2M_EXP_TEQ)

This function is similar to MOD_EXP but works on binary polynomials. It is provided mainly to assist in the computation of elliptic curve parameter "c", where c = $b^{2^{m-2}}$ mod $n$) given an elliptic curve parameter "b" and the field-defining polynomial in N. It computes $R^2$ mod N and converts A to Montgomery form before beginning the exponentiation. F2M_EXP_TEQ performs the same operation as F2M_EXP but with an added timing equalization security feature. Its exponentiation run-time, for a given modulus and size of exponent, is constant. In general F2M_EXP will run faster than F2M_EXP_TEQ, but will never run slower.

**Table 11-29. F2M_EXP function properties**

| Property | Notes |
|---|---|
| Mode value for F2M_EXP | 0010_0000_0**0**0***0***_0000_0110 ( output placed in B )<br>0010_0000_0**0**0***1***_0000_0110 ( output placed in A ) |
| Mode value for F2M_EXP_TEQ | 0010_0000_0**1**0***0***_0000_0110 ( output placed in B )<br>0010_0000_0**1**0***1***_0000_0110 ( output placed in A ) |
| Input | • N = modulus, an "odd" binary polynomial of order m<br>• A = a binary polynomial < $2^m$<br>• E = exponent, any integer |
| Output | B (or A, if selected) = ($A^E$) mod N, a binary polynomial |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• Maximum key (exponent) size = 512 bytes |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.28 Binary Polynomial ($F_{2^m}$) Simultaneous Modular Exponentiation (F2M_SML_EXP)

F2M_SML_EXP performs two modular exponentiations on binary polynomials, and multiplies the results. This is faster than doing them separately. It is useful for DSA Verification.

Table 11-30. F2M_SML_EXP function properties

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0001_0110 ( output placed in B )<br>0000_0000_000***1***_0001_0110 ( output placed in A ) |
| Input | • N = modulus, any odd integer<br>• A0 = a field element, first base<br>• E = an integer, first exponent<br>• A2 = a field element, second base<br>• B = an integer, second exponent |
| Output | B (or A if selected) = $A0^E$ * $A2^B$ mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 1/2 RAM<br>• Maximum key (exponent) size and B size = 1/4 RAM size<br>• A0 < N<br>• A2 < N<br>• The values in A0 and A2 must be the same number of bytes, matching the A SIZE register, and should be the same size as N |
| Side effects | A, B, and E are modified. |
| Errors reported | • N size error is set if N size > half RAM<br>• A size error is set if A size > half RAM<br>• B size error is set if B size > quarter-RAM<br>• E size error is set if E size > quarter-RAM |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.29 Binary Polynomial ($F_{2^m}$) Modular Square (F2M_SQR and F2M_SQR_TEQ)

F2M_SQR may be used to square an binary polynomial value. The timing equalized version, F2M_SQR_TEQ, will also square a binary polynomial value but for a given modulus will always take the same time to complete. F2M_SQR will usually run faster than F2M_SQR_TEQ, but will never run slower

Table 11-31. F2M_SQR and F2M_SQR_TEQ function properties

| Property | Notes |
|---|---|
| Mode value for F2M_SQR | 0010_0000_0**0**0***0***_0001_1010 ( output placed in B )<br>0010_0000_0**0**0***1***_0001_1010 ( output placed in A ) |
| Mode value for F2M_SQR_TEQ | 0010_0000_0**1**0***0***_0001_1010 ( output placed in B )<br>0010_0000_0**1**0***1***_0001_1010 ( output placed in A ) |
| Input | • N = modulus, an "odd" binary polynomial of order m<br>• A = a binary polynomial of order < m |
| Output | B (or A, if selected) = (A*A) mod N |
| Requirements | • Minimum modulus size = 1 byte |

*Table continues on the next page...*

**Table 11-31. F2M_SQR and F2M_SQR_TEQ function properties (continued)**

| Property | Notes |
|---|---|
| | • Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

### 11.1.6.30 Binary Polynomial ($F_{2^m}$) Modular Square, Montgomery Input (F2M_SQR_IM and F2M_SQR_IM_TEQ)

F2M_SQR_IM may be used to square a binary polynomial in Montgomery form. F2M_SQR_IM_TEQ, the timing equalized version, will take the same time to complete for a given modulus. F2M_SQR_IM will generally run faster than F2M_SQR_IM_TEQ, but will never run slower.

**Table 11-32. F2M_SQR_IM and F2M_SQR_IM_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for F2M_SQR_IM | 1010_0000_0**00*0***_0001_1010 ( output placed in B )<br>1010_0000_0**00*1***_0001_1010 ( output placed in A ) |
| Mode value for F2M_SQR_IM_TEQ | 1010_0000_0**10*0***_0001_1010 ( output placed in B )<br>1010_0000_0**10*1***_0001_1010 ( output placed in A ) |
| Input | • N = modulus, an "odd" binary polynomial of order m<br>• A = a binary polynomial of order < m in Montgomery form |
| Output | B (or A, if selected) = (A*A) mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

### 11.1.6.31 Binary Polynomial ($F_{2^m}$) Modular Square, Montgomery Input and Output (F2M_SQR_IM_OM and F2M_SQR_IM_OM_TEQ)

F2M_SQR_IM_OM may be used to square a binary polynomial in Montgomery form, and will output the result in Montgomery form. F2M_SQR_IM_OM_TEQ, the timing equalized version, will take the same time to complete for a given modulus. F2M_SQR_IM_OM will generally run faster than F2M_SQR_IM_OM_TEQ, but will never run slower.

**Table 11-33. F2M_SQR_IM_OM and F2M_SQR_IM_OM_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for F2M_SQR_IM_OM | 1110_0000_0**0**0***0***_0001_1010 ( output placed in B )<br>1110_0000_0**0**0***1***_0001_1010 ( output placed in A ) |
| Mode value for F2M_SQR_IM_OM _TEQ | 1110_0000_0**1**0***0***_0001_1010 ( output placed in B )<br>1110_0000_0**1**0***1***_0001_1010 ( output placed in A ) |
| Input | • N = modulus, an "odd" binary polynomial of order m<br>• A = a binary polynomial of order < m, possibly in Montgomery form |
| Output | B (or A, if selected) = (A*A) mod N, in Montgomery form |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

### 11.1.6.32 Binary Polynomial ($F_{2^m}$) Modular Cube (F2M_CUBE and F2M_CUBE_TEQ)

F2M_CUBE may be used to cube a binary polynomial value. The function will first compute the Montgomery conversion factor, $R^2 \bmod N$ and convert the value to cube. F2M_CUBE_TEQ, the timing equalized version, performs the same function but will take the same time to complete for a given modulus. F2M_CUBE will generally run faster than F2M_CUBE_TEQ, but will never run slower.

**Table 11-34. F2M_CUBE and F2M_CUBE_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for F2M_CUBE | 0010_0000_0**0**0***0***_0001_1011 ( output placed in B )<br>0010_0000_0**0**0***1***_0001_1011 ( output placed in A ) |

*Table continues on the next page...*

Table 11-34. F2M_CUBE and F2M_CUBE_TEQ function properties (continued)

| Property | Notes |
|---|---|
| Mode value for F2M_CUBE_TEQ | 0010_0000_0**1**0***0***_0001_1011 ( output placed in B )<br>0010_0000_0**1**0***1***_0001_1011 ( output placed in A ) |
| Input | • N = modulus, an "odd" binary polynomial of order *m*<br>• A = a binary polynomial of order <*m* |
| Output | B (or A, if selected) = (A*A*A) mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

### 11.1.6.33 Binary Polynomial ($F_{2^m}$) Modular Cube, Montgomery Input (F2M_CUBE_IM and F2M_CUBE_IM_TEQ)

F2M_CUBE_IM may be used to cube a binary polynomial value that is in Montgomery form. F2M_CUBE_IM_TEQ, the timing equalized version, performs the same function but will take the same time to complete for a given modulus. F2M_CUBE_IM will generally run faster than F2M_CUBE_IM_TEQ, but will never run slower.

Table 11-35. F2M_CUBE_IM and F2M_CUBE_IM_TEQ function properties

| Property | Notes |
|---|---|
| Mode value for F2M_CUBE_IM | 1010_0000_0**0**0***0***_0001_1011 ( output placed in B )<br>1010_0000_0**0**0***1***_0001_1011 ( output placed in A ) |
| Mode value for F2M_CUBE_IM_TEQ | 1010_0000_0**1**0***0***_0001_1011 ( output placed in B )<br>1010_0000_0**1**0***1***_0001_1011 ( output placed in A ) |
| Input | • N = modulus, an "odd" binary polynomial of order m<br>• A = a binary polynomial of order < m in Montgomery form |
| Output | B (or A, if selected) = (A*A*A) mod N, a binary polynomial |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |

*Table continues on the next page...*

Table 11-35. F2M_CUBE_IM and F2M_CUBE_IM_TEQ function properties (continued)

| Property | Notes |
|---|---|
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

### 11.1.6.34 Binary Polynomial ($F_{2^m}$) Modular Cube, Montgomery Input and Output (F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_TEQ)

F2M_CUBE_IM_OM may be used to cube a binary polynomial value that is in Montgomery form, and output the result in Montgomery form. F2M_CUBE_IM_OM_TEQ, the timing equalized version, performs the same function but will take the same time to complete for a given modulus. F2M_CUBE_IM_OM will generally run faster than F2M_CUBE_IM_OM_TEQ, but will never run slower.

Table 11-36. F2M_CUBE_IM_OM and F2M_CUBE_IM_OM_EXP function properties

| Property | Notes |
|---|---|
| Mode value for F2M_CUBE_IM_OM | 1110_0000_0**0**0***0***_0001_1011 ( output placed in B )<br>1110_0000_0**0**0***1***_0001_1011 ( output placed in A ) |
| Mode value for F2M_CUBE_IM_OM_TEQ | 1110_0000_0**1**0***0***_0001_1011 ( output placed in B )<br>1110_0000_0**1**0***1***_0001_1011 ( output placed in A ) |
| Input | • N = modulus, an "odd" binary polynomial of order m<br>• A = a binary polynomial of order < m, possibly in Montgomery form |
| Output | B (or A, if selected) = (A*A*A) mod N, in Montgomery form |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A < N |
| Side effects | |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N. |
| Flags set | PIZ is set if the result is zero.<br>GCD is set if the result is one. |

### 11.1.6.35 Binary Polynomial ($F_{2^m}$) Modulo Reduction (F2M_AMODN)

F2M_AMODN computes the remainder of A divided by N. This is the equivalent of the MOD_AMODN function applied to a binary polynomial. A and N can be of any size and it is not required that A > N, but N must be non-zero.

**Table 11-37. F2M_AMODN function properties**

| Property | Notes |
|---|---|
| Mode value | 0010_0000_000***0***_0000_0111 ( output placed in B )<br>0010_0000_000***1***_0000_0111 ( output placed in A ) |
| Input | • N = modulus, any non-zero polynomial<br>• A = any polynomial |
| Output | B (or A, if selected) = A mod N, a polynomial, binary element modulo N |
| Requirements | • N = non-zero value<br>• Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Divide By Zero Error is set if N = 0. |
| Flags set | PIZ is set if the result is zero.<br>PIO is set if the result is one. |

### 11.1.6.36 Binary Polynomial ($F_{2^m}$) Modular Inversion (F2M_INV)

F2M_INV computes the multiplicative inverse of a binary polynomial.

**Table 11-38. F2M_INV function properties**

| Property | Notes |
|---|---|
| Mode value | 0010_0000_000***0***_0000_1000 ( output placed in B )<br>0010_0000_000***1***_0000_1000 ( output placed in A ) |
| Input | • N = modulus, an irreducible polynomial<br>• A = a field element |
| Output | B (or A, if selected) = $A^{-1}$ mod N, a field element, the multiplicative inverse of A |
| Requirements | • A is an element of the binary polynomial field.<br>• Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | A and E are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0. |
| Flags set | None |

### 11.1.6.37 Binary Polynomial ($F_{2^m}$) $R^2$ Mod N (F2M_R2) Function

This function is used to compute the Montgomery Conversion Factor, which is used to convert operands into the Montgomery residue system representation. The constant $R^2$(mod N) is dependent upon the digit size of the PKHA and the value of N. If this value is not available, then this routine (function) is called to determine the constant before other operations. If N contains a protocol- or system-wide parameter that does not change frequently, such as in ECC operations for a specific curve, save this computed constant, because this routine takes a considerable amount of time to complete.

**Table 11-39. F2M_R2 function properties**

| Property | Notes |
|---|---|
| Mode value | 0010_0000_000*0*_0000_1100 ( output placed in B )<br>0010_0000_000*1*_0000_1100 ( output placed in A ) |
| Input | N = modulus, an irreducible polynomial |
| Output | B (or A, if selected) = R2 mod N, where R = $2^{SD}$ where S is size of a digit in bits and D is the number of digits of an irreducible polynomial, in other words D = ceiling [sizeof(N) in bits / S] |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | None |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even. |
| Flags set | None |

### 11.1.6.38 Binary Polynomial ($F_{2^m}$) Greatest Common Divisor (F2M_GCD) Function

MOD_GCD finds the greatest common divisor of two binary polynomials.

**Table 11-40. F2M_GCD function properties**

| Property | Notes |
|---|---|
| Mode value | 0010_0000_000*0*_0000_1110 ( output placed in B )<br>0010_0000_000*1*_0000_1110 ( output placed in A ) |
| Input | • N = any polynomial. The most-significant digit of N must be non-zero.<br>• A = any polynomial with degree less than or equal to N |
| Output | B (or A, if selected) = BINARY_GCD(A,N), a polynomial with degree less than or equal to polynomial A that divides both A and N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes<br>• A and N may not both be zero |

*Table continues on the next page...*

**Table 11-40. F2M_GCD function properties (continued)**

| Property | Notes |
|---|---|
| Side effects | A is modified |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• A Size Error is set if the size of A is greater than size of N.<br>• Divide-By-Zero Error is set if N or A = 0, or if the most significant digit of N = 0. |
| Flags set | PIO is set if the result is 1. |

### 11.1.6.39 ECC $F_p$ Point Add, Affine Coordinates (ECC_MOD_ADD) Function

ECC_MOD_ADD performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates.

**Table 11-41. ECC_MOD_ADD function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0000_1001 ( output placed in B )<br>0000_0000_000***1***_0000_1001 ( output placed in A ) |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero<br>• [A0, A1] = first addend in affine coordinates<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• [B1, B2] = second addend in affine coordinates<br>• B3 = ignored |
| Output | [B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates. |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field and therefore are less than the modulus N. |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error will be set if size of B is greater than size of N.<br>• Divide-By-Zero Error is set if the most-significant digit of N = 0. |
| Flags set | None |

### 11.1.6.40 ECC $F_p$ Point Add, Affine Coordinates, $R^2$ Mod N Input (ECC_MOD_ADD_R2) Function

ECC_MOD_ADD_R2 performs an addition of two points on an elliptic curve. The addends are input and the sum is output in affine coordinates. Since ECC_MOD_ADD_R2 has $R^2$ mod N as an additional input, this function is more efficient than ECC_MOD_ADD, which first must compute $R^2$ mod N before performing the addition.

Table 11-42. ECC_MOD_ADD_R2 function properties

| Property | Notes |
|---|---|
| Mode value | 0001_0000_000*0*_0000_1001 ( output placed in B )<br>0001_0000_000*1*_0000_1001 ( output placed in A ) |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero<br>• [A0, A1] = first addend point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• [B1, B2] = second addend point in affine coordinates (x,y)<br>• B3 = R2 ($R^2$ mod N) input |
| Output | [B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates. |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field formed by N. |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error will be set if size of B is greater than size of N.<br>• Divide-By-Zero Error is set if the most-significant digit of N = 0. |
| Flags set | None |

### 11.1.6.41 ECC $F_p$ Point Double, Affine Coordinates (ECC_MOD_DBL) Function

ECC_MOD_DBL computes the double (B + B) of a point B on an elliptic curve. The input and output are in affine coordinates.

Table 11-43. ECC_MOD_DBL function properties

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000*0*_0000_1010 ( output placed in B ) |

*Table continues on the next page...*

**Table 11-43. ECC_MOD_DBL function properties (continued)**

| Property | Notes |
|---|---|
| | 0000_0000_000***1***_0000_1010 ( output placed in A ) |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero.<br>• [A0, A1, A2] = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• [B1, B2] = input point in affine coordinates<br>• B3 = ignored |
| Output | [B1, B2] (or [A0, A1], if A output selected) = [B1, B2] + [B1, B2], where "+" represents an elliptic-curve point addition. Output is in affine coordinates (x, y). |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• Point coordinates B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field formed by N. |
| Side effects | A0, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity. |

### 11.1.6.42 ECC $F_p$ Point Multiply, Affine Coordinates (ECC_MOD_MUL and ECC_MOD_MUL_TEQ) Function

ECC_MOD_MUL computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. ECC_MOD_MUL_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECC_MOD_MUL in general will run faster than ECC_MOD_MUL_TEQ, but will never run slower.

**Table 11-44. ECC_MOD_MUL and ECC_MOD_MUL_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for ECC_MOD_MUL | 0000_0000_0**0**0***0***_0000_1011 ( output placed in B )<br>0000_0000_0**0**0***1***_0000_1011 ( output placed in A ) |
| Mode value for ECC_MOD_MUL_TEQ | 0000_0000_0**1**0***0***_0000_1011 ( output placed in B )<br>0000_0000_0**1**0***1***_0000_1011 ( output placed in A ) |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero<br>• E = scalar multiplier (k), any integer<br>• [A0, A1] = multiplicand, an input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter |

*Table continues on the next page...*

**Table 11-44. ECC_MOD_MUL and ECC_MOD_MUL_TEQ function properties (continued)**

| Property | Notes |
|---|---|
| | • B1 = ignored<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four equal-size segments of A and B memory locations. |
| Output | • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y).<br>• B0 = undefined<br>• B3 = undefined |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• The point (A0, A1) must be on the elliptic curve formed by (N, A3, B0). |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity. |

The following special cases should be noted:

- For k = 0, this function returns a point at infinity; that is (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(-k). After the computation is complete, the formula -P= (x,-y) can be used to compute the "y" coordinate of the effective final result, and other coordinates are the same.

### 11.1.6.43 ECC $F_p$ Point Multiply, $R^2$ Mod N Input, Affine Coordinates (ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ) Function

ECC_MOD_MUL_R2 computes a scalar multiplication of a point on an elliptic curve. The input point and the output point are in affine coordinates. Since ECC_MOD_MUL_R2 has $R^2$ mod N as an additional input, this function is more efficient than ECC_MOD_MUL, which first must compute $R^2$ mod N before performing the multiplication. ECC_MOD_MUL_R2_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECC_MOD_MUL_R2 in general will run faster than ECC_MOD_MUL_R2_TEQ, but will never run slower.

**Table 11-45. ECC_MOD_MUL_R2 and ECC_MOD_MUL_R2_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for ECC_MOD_MUL_R2 | 0001_0000_0**0**0***0***_0000_1011 ( output placed in B )<br>0001_0000_0**0**0***1***_0000_1011 ( output placed in A ) |
| Mode value for ECC_MOD_MUL_R2_TEQ | 0001_0000_0**1**0***0***_0000_1011 ( output placed in B )<br>0001_0000_0**1**0***1***_0000_1011 ( output placed in A ) |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero.<br>• E = key, scalar multiplier (k), any integer<br>• [A0, A1] = multiplicand, an input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• B1 = R2 mod N, pre-computed as described in MOD_R2<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four equal-size segments of A and B memory locations. |
| Output | • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y).<br>• B0 = undefined<br>• B3 = undefined |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 are elements of the prime field formed by the modulus N. |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity. |

The following special cases should be noted:

- For k = 0, this function returns a point at infinity; that is, (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(- k). After the computation is complete, the formula -P = (x,-y) can be used to compute the "y" coordinate of the effective final result, and other coordinate is the same.

### 11.1.6.44 ECC $F_p$ Check Point (ECC_MOD_CHECK_POINT) Function

ECC_MOD_CHECK_POINT determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $y^3 = x^3 + ax + b$.

ECC_MOD_CHECK_POINT checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2$ mod N and $x^3 + ax + b$ mod N. If they are equal, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged.

**Table 11-46. ECC_MOD_CHECK_POINT function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_0000_0001_1100 |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero.<br>• [A0, A1] = a possible input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• B1 = ignored<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations. |
| Output | • B2 = $R^2$ mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | Various quadrants are modified, but inputs are unchanged. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity.<br>GCD is set if the point is on the curve (but not point at infinity). |

### 11.1.6.45 ECC $F_p$ Check Point, $R^2$ Mod N Input, Affine Coordinates (ECC_MOD_CHECK_POINT_R2) Function

ECC_MOD_CHECK_POINT_R2 determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $y^2 = x^3 + ax + b$.
ECC_MOD_CHECK_POINT_R2 checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2$ mod N and $x^3 + ax + b$ mod N. If they are equal, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain

unchanged. Since this function takes $R^2$ mod N as an additional input, ECC_MOD_CHECK_POINT_R2 is more efficient than ECC_MOD_CHECK_POINT, which first must compute $R^2$ mod N before performing the point check.

**Table 11-47. ECC_MOD_CHECK_POINT_R2 function properties**

| Property | Notes |
|---|---|
| Mode value | 0001_0000_0000_0001_1100 |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero.<br>• [A0, A1] = a possible input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• B1 = (f2m) R2 mod N, pre-computed as described in F2M_R2MODN (0Eh)<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations. |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | Various quadrants are modified, but inputs are unchanged. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity.<br>GCD is set if the point is on the curve (but not point at infinity). |

ECC_MOD_CHECK_POINT_R2 checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2 = x^3 + ax + b$ mod N. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged.

### 11.1.6.46 ECC $F_{2^m}$ Point Add, Affine Coordinates (ECC_F2M_ADD) Function

ECC_F2M_ADD performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates.

**Table 11-48. ECC_F2M_ADD function properties**

| Property | Notes |
|---|---|
| Mode value | 0010_0000_000***0***_0000_1001 ( output placed in B )<br>0010_0000_000***1***_0000_1001 ( output placed in A ) |

*Table continues on the next page...*

Table 11-48. ECC_F2M_ADD function properties (continued)

| Property | Notes |
|---|---|
| Input | • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero.<br>• [A0, A1] = first addend in affine coordinates<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "c" parameter, where c = $b^{2^{m-2}}$ mod *n*<br>• B1, B2] = second addend in affine coordinates<br>• B3 = ignored |
| Output | P[B1, B2] (or P[A0, A1], if A output selected) = P[A0, A1] + P[B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates. |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• Point coordinates A0, A1, B1, and B2 and elliptic curve parameters A3 and B0 are elements of the binary polynomial field N. |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error will be set if size of B is greater than size of N.<br>• Divide By Zero Error is set if the most significant digit of N = 0.<br>• C is Zero Error if B3 is zero. |
| Flags set | None |

### 11.1.6.47 ECC $F_{2^m}$ Point Add, Affine Coordinates, $R^2$ Mod N Input (ECC_F2M_ADD_R2) Function

ECC_F2M_ADD_R2 performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates. Since this function takes $R^2$ mod N as an additional input, ECC_F2M_ADD_R2 is more efficient than ECC_F2M_ADD, which first must compute $R^2$ mod N before performing the addition.

Table 11-49. ECC_F2M_ADD_R2 function properties

| Property | Notes |
|---|---|
| Mode value | 0011_0000_000***0***_0000_1001 ( output placed in B )<br>0011_0000_000***1***_0000_1001 ( output placed in A ) |
| Input | • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero.<br>• [A0, A1] = first addend in affine coordinates<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "c" parameter, where c = $b^{2^{m-2}}$ mod *n*. Must not be zero.<br>• [B1, B2] = second addend in affine coordinates<br>• B3 = R2 input |
| Output | P[B1, B2] (or P[A0, A1], if A output selected) = P[A0, A1] + P[B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates. |
| Requirements | • Minimum modulus size = 1 byte |

*Table continues on the next page...*

Table 11-49. ECC_F2M_ADD_R2 function properties (continued)

| Property | Notes |
|---|---|
| | • Maximum modulus size = 128 bytes<br>• Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the binary polynomial field. |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error will be set if size of B is greater than size of N.<br>• Divide By Zero Error is set if the most significant digit of N = 0.<br>• C is Zero Error if B3 is zero. |
| Flags set | None |

### 11.1.6.48 ECC F$_{2^m}$ Point Double - Affine Coordinates (ECC_F2M_DBL) Function

ECC_F2M_DBL computes the double (B + B) of a point B on an elliptic curve. The input and output are in affine coordinates.

Table 11-50. ECC_F2M_DBL function properties

| Property | Notes |
|---|---|
| Mode value | 0010_0000_000***0***_0000_1010 ( output placed in B )<br>0010_0000_000***1***_0000_1010 ( output placed in A ) |
| Input | • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero.<br>• A0, A1, A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "c" parameter where c = $b^{2^{m-2}}$ mod *n*<br>• [B1, B2] = input point in affine coordinates<br>• B3 = ignored |
| Output | P[B1, B2] (or P[A0, A1], if A output selected) = P[B1, B2] + P[B1, B2], where "+" represents an elliptic-curve point addition. Output is in affine coordinates. |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• Point coordinates B1 and B2, and elliptic curve parameters A3 and B0 are elements of the binary polynomial field formed by N. |
| Side effects | A0, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0.<br>• C is Zero Error if B3 is zero. |
| Flags set | PIZ is set if the result is the point at infinity. |

### 11.1.6.49 ECC $F_{2^m}$ Point Multiply, Affine Coordinates (ECC_F2M_MUL and ECC_F2M_MUL_TEQ) Function

ECC_F2M_MUL computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. ECC_F2M_MUL_TEQ performs the same operation as ECC_F2M_MUL but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. In general ECC_F2M_MUL will run faster than ECC_F2M_MUL_TEQ, but will never run slower.

**Table 11-51. ECC_F2M_MUL and ECC_F2M_MUL_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for ECC_F2M_MUL | 0010_0000_0**00*0***_0000_1011 ( output placed in B )<br>0010_0000_0**00*1***_0000_1011 ( output placed in A ) |
| Mode value for ECC_F2M_MUL_TEQ | 0010_0000_0**11*0***_0000_1011 ( output placed in B )<br>0010_0000_0**11*1***_0000_1011 ( output placed in A ) |
| Input | • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero.<br>• E = key, scalar multiplier (k), any integer<br>• [A0, A1] = multiplicand, an input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "c" parameter where c = $b^{2^{m-2}}$ mod *n*<br>• B1 = ignored<br>• B2 = ignored<br>• B3 = ignored |
| Output | • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y).<br>• B0 = undefined<br>• B3 = undefined |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes (irreducible polynomial of maximum degree 1023<br>• Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 must be elements of the binary polynomial field. |
| Side effects | A0, A1 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0or size of N > 128.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0.<br>• C is Zero Error if B3 is zero. |
| Flags set | PIZ is set if the result is the point at infinity. |

The following special cases should be noted:

- For E = 0, this function returns a point at infinity (0,0).
- For E < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = -E). After the multiplication is complete, the formula -P= (x, x+y) can be used to compute the *y* coordinate of the effective final result; the *x* coordinate stays the same.

### 11.1.6.50 ECC $F_{2^m}$ Point Multiply, $R^2$ Mod N Input, Affine Coordinates (ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ) Function

ECC_F2M_MUL_R2 computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. Since this function takes $R^2$ mod N as an additional input, ECC_F2M_MUL_R2 is more efficient than ECC_F2M_MUL, which first must compute $R^2$ mod N before performing the multiplication.

ECC_F2M_MUL_R2_TEQ performs the same operation as ECC_F2M_MUL_R2 but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. In general ECC_F2M_MUL_R2 will run faster than ECC_F2M_MUL_R2_TEQ, but will never run slower.

**Table 11-52. ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for ECC_F2M_MUL_R2 | 0011_0000_0**0**0***0***_0000_1011 ( output placed in B )<br>0011_0000_0**0**0***1***_0000_1011 ( output placed in A ) |
| Mode value for ECC_F2M_MUL_R2_TEQ | 0011_0000_0**1**0***0***_0000_1011 ( output placed in B )<br>0011_0000_0**1**0***1***_0000_1011 ( output placed in A ) |
| Input | • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero.<br>• E = key, scalar multiplier (k), any integer<br>• [A0, A1] = multiplicand, an input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "c" parameter where c = $b^{2^{m-2}}$ mod *n* and m = degree of polynomial M<br>• B1 = (f2m) R2 mod N, pre-computed as described in F2M_R2MODN (0Eh)<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four, equally size segments of A and B memory locations. |
| Output | • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y).<br>• B0 = undefined<br>• B3 = undefined |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes (irreducible polynomial of maximum degree 1023<br>• Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 must be elements of the binary polynomial field. |
| Side effects | A0, A1, A2, and B3 are modified. |

*Table continues on the next page...*

Table 11-52. ECC_F2M_MUL_R2 and ECC_F2M_MUL_R2_TEQ function properties (continued)

| Property | Notes |
|---|---|
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0.<br>• C is Zero Error if B3 is zero. |
| Flags set | PIZ is set if the result is the point at infinity. |

The following special cases should be noted:

- For k = 0, this function returns a point at infinity (0,0).
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(- k). After the computation is complete, the formula -P = (x, x+y) can be used to compute the "y" coordinate of the effective final result, and other coordinate is the same.

### 11.1.6.51 ECC $F_{2^m}$ Check Point (ECC_F2M_CHECK_POINT) Function

This function determines whether the point (x,y) is on the elliptic curve, i.e. satisfies the equation $y^3 + xy = x^3 + ax^2 + b$.

Table 11-53. ECC_F2M_CHECK_POINT function properties

| Property | Notes |
|---|---|
| Input | • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero.<br>• [A0, A1] = a possible input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "c" parameter<br>• B1 = ignored<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations. |
| Output | • B2 = $R^2$ mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes |
| Side effects | Various quadrants are modified, but inputs are unchanged. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity. |

Table 11-53. ECC_F2M_CHECK_POINT function properties

| Property | Notes |
|---|---|
| | GCD is set if the point is on the curve (but not point at infinity). |

This function checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ set and the routine exits. It then computes $y^2$ mod N and $x^3 + ax + b$ mod N. If they are equal, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set meanning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged..

### 11.1.6.52 ECC $F_{2^m}$ Check Point, $R^2$ (ECC_F2M_CHECK_POINT_R2) Function

This function determines whether the point (x,y) is on the elliptic curve, i.e. satisfies the equation $y^2 + xy = x^3 + ax^2 + b$.

Since this function has $R^2$ mod N as an additional input, it is more efficient than ECC_F2M_CHECK_POINT, which first must compute $R^2$ mod N before performing the operaton.

Table 11-54. ECC_F2M_CHECK_POINT_R2 function properties

| Property | Notes |
|---|---|
| Input | • N = modulus, an irreducible polynomial. The most significant digit of N must be non-zero.<br>• [A0, A1] = a possible input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "c" parameter<br>• B1 = $R^2$ mod N<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations. |
| Output | • B2 = curve "b" parameter = $c^4$ mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes |
| Side effects | Various quadrants are modified, but inputs are unchanged. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity.<br>GCD is set if the point is on the curve (but not point at infinity). |

This function checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ set and the routine exits. It then computes $y^2 + xy = x^3 + ax^2 + b$ mod N. If the equation is true, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set, meanning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. The output will not be present if PIZ is set.

### 11.1.6.53 ECM Modular Multiplication (ECM_MOD_MUL and ECM_MOD_MUL_TEQ) Function

ECM_MOD_MUL computes the scalar multiplication of a point on an elliptic curve in Montgomery form. The input and output are just the x coordinates of the points. ECM_MOD_MUL_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3), constant for a given size of E. ECM_MOD_MUL in general will run faster than ECM_MOD_MUL_TEQ, but will never run slower.

This function computes a point multiplication on a Montgomery curve, using Montgomery values, by means of a Montgomery ladder. At the end of the ladder, P2 = P3 + P1, where P1 is the input and P3 is the result.

**Table 11-55. ECM_MOD_MUL and ECM_MOD_MUL_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for ECM_MOD_MUL | 0000_0000_0**0**0***0***_0100_1011 ( output placed in B )<br>0000_0000_0**0**0***0***_0100_1011 ( output placed in A ) |
| Mode value for ECM_MOD_MUL_TEQ | 0000_0000_0**1**0***0***_0100_1011 ( output placed in B )<br>0000_0000_0**1**0***1***_0100_1011 ( output placed in A ) |
| Input | • N = modulus, a prime number.<br>• E = scalar multiplier (k), any integer<br>• [A0] = multiplicand, an input point's affine x coordinate<br>• A2 = ignored<br>• A3 = elliptic curve a24 parameter, that is, (A+2)/4<br>• B0 = ignored<br>• B1 = ignored<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four equal-size segments of A and B memory locations. |
| Output | • P[B1] (or P[A0], if A output selected) = E x P[A0], where "x" denotes elliptic curve scalar point multiplication. Output is the resulting point's affine x coordinate.<br>• N1 = R2<br>• A1 = X2R, the X the (X,Z) scalar multiplication, with the Montgomery factor. P2x = X2R/Z2R<br>• A2 = Z2R, the Z of the (X,Z) scalar multiplication, with the Montgomery factor<br>• A3 = a24R, the a24 input, with the Montgomery factor<br>• B0 = X3R, the X result of the (X,Z) scalar multiplication, with the Montgomery factor. P3x = X3R/Z3R |

*Table continues on the next page...*

Table 11-55. ECM_MOD_MUL and ECM_MOD_MUL_TEQ function properties (continued)

| Property | Notes |
|---|---|
| | • B2 = Z3R, the Z result of the (X,Z) scalar multiplication, with the Montgomery factor<br>• B3 = X1R, the x input, with the Montgomery factor |
| Requirements | • Maximum modulus size = 128 bytes<br>• The x in (A0) should be on the elliptic curve formed by (N, A3 and "B"). |
| Side effects | All quadrants of A, B, and N are modified except N0. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N. |
| Flags set | PIZ is set if the result is the point at infinity. |

### 11.1.6.54 ECM $F_p$ Point Multiply, $R^2$ Mod N Input, Affine Coordinates (ECM_MOD_MUL_R2 and ECM_MOD_MUL_R2_TEQ) Function

ECM_MOD_MUL_R2 computes a scalar multiplication of a point on an elliptic curve in Montgomery form. The input and output are just the x coordinates of the points. Since ECM_MOD_MUL_R2 has $R^2$ mod N as an additional input, this function is more efficient than ECM_MOD_MUL, which first must compute $R^2$ mod N before performing the multiplication. ECM_MOD_MUL_R2_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3), constant for a given size of E. ECM_MOD_MUL_R2 in general will run faster than ECM_MOD_MUL_R2_TEQ, but will never run slower.

This function computes a point multiplication on a Montgomery curve, using Montgomery values, by means of a Montgomery ladder. At the end of the ladder, P2 = P3 + P1, where P1 is the input and P3 is the result.

Table 11-56. ECM_MOD_MUL_R2 and ECM_MOD_MUL_R2_TEQ function properties

| Property | Notes |
|---|---|
| Mode value for ECM_MOD_MUL_R2 | 0001_0000_0**00*0***_1000_1011 ( output placed in B )<br>0001_0000_0**00*1***_1000_1011 ( output placed in A ) |
| Mode value for ECM_MOD_MUL_R2_TEQ | 0001_0000_0**10*0***_1000_1011 ( output placed in B )<br>0001_0000_0**10*1***_1000_1011 ( output placed in A ) |
| Input | • N = modulus, a prime number.<br>• E = scalar multiplier (k), any integer<br>• [A0] = multiplicand, an input point's affine x coordinate |

*Table continues on the next page...*

**Table 11-56. ECM_MOD_MUL_R2 and ECM_MOD_MUL_R2_TEQ function properties (continued)**

| Property | Notes |
|---|---|
| | • A2 = ignored<br>• A3 = elliptic curve a24 parameter, that is, (A+2)/4<br>• B0 = ignored<br>• B1 = R2 mod N, pre-computed as described in MOD_R2<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four equal-size segments of A and B memory locations. |
| Output | • P[B1] (or P[A0], if A output selected) = E x P[A0], where "x" denotes elliptic curve scalar point multiplication. Output is the resulting point's affine x coordinate.<br>• N1 = R2<br>• A1 = X2R, the X the (X,Z) scalar multiplication, with the Montgomery factor. P2x = X2R/Z2R<br>• A2 = Z2R, the Z of the (X,Z) scalar multiplication, with the Montgomery factor<br>• A3 = a24R, the a24 input, with the Montgomery factor<br>• B0 = X3R, the X result of the (X,Z) scalar multiplication, with the Montgomery factor. P3x = X3R/Z3R<br>• B2 = Z3R, the Z result of the (X,Z) scalar multiplication, with the Montgomery factor<br>• B3 = X1R, the x input, with the Montgomery factor |
| Requirements | • Maximum modulus size = 128 bytes<br>• The x in (A0) should be on the elliptic curve formed by (N, A3 and "B"). |
| Side effects | All quadrants of A, B, and N are modified except N0. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N. |
| Flags set | PIZ is set if the result is the point at infinity. |

### 11.1.6.55 ECT Modular Multiplication (ECT_MOD_MUL and ECT_MOD_MUL_TEQ) Function

ECT_MOD_MUL computes the scalar multiplication of a point on an elliptic curve. The input and output are in affine coordinates. ECT_MOD_MUL_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECT_MOD_MUL in general will run faster than ECT_MOD_MUL_TEQ, but will never run slower.

**Table 11-57. ECT_MOD_MUL and ECT_MOD_MUL_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for ECT_MOD_MUL | 0000_0000_0**0**0***0***>_1000_1011 ( output placed in B )<br>0000_0000_0**0**0***0***_1000_1011 ( output placed in A ) |
| Mode value for ECT_MOD_MUL_TEQ | 0000_0000_0**1**0***0***_1000_1011 ( output placed in B )<br>0000_0000_0**1**0***1***_1000_1011 ( output placed in A ) |

*Table continues on the next page...*

### Table 11-57. ECT_MOD_MUL and ECT_MOD_MUL_TEQ function properties (continued)

| Property | Notes |
|---|---|
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero<br>• E = scalar multiplier (k), any integer<br>• [A0, A1] = multiplicand, an input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" *(or should this be "d")* parameter<br>• B1 = ignored<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four equal-size segments of A and B memory locations. |
| Output | • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y).<br>• B0 = undefined<br>• B3 = undefined |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• The point (A0, A1) must be on the elliptic curve formed by (N, A3, B0). |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity. |

The following special cases should be noted:

- For k = 0, this function returns a point at infinity; that is (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(-k). After the computation is complete, the formula -P= (x,-y) can be used to compute the "y" coordinate of the effective final result, and other coordinates are the same.

### 11.1.6.56 ECT $F_p$ Point Multiply, $R^2$ Mod N Input, Affine Coordinates (ECT_MOD_MUL_R2 and ECT_MOD_MUL_R2_TEQ) Function

ECT_MOD_MUL_R2 computes a scalar multiplication of a point on an elliptic curve. The input point and the output point are in affine coordinates. Since ECT_MOD_MUL_R2 has $R^2$ mod N as an additional input, this function is more efficient than ECT_MOD_MUL, which first must compute $R^2$ mod N before performing the multiplication. ECT_MOD_MUL_R2_TEQ computes the same function, but with an added timing equalization security feature. Its computation run-time is, for a given curve (N, A3, B0), constant for a given size of E. ECT_MOD_MUL_R2 in general will run faster than ECT_MOD_MUL_R2_TEQ, but will never run slower.

**Table 11-58. ECT_MOD_MUL_R2 and ECT_MOD_MUL_R2_TEQ function properties**

| Property | Notes |
|---|---|
| Mode value for ECT_MOD_MUL_R2 | 0001_0000_0000_1000_1011 ( output placed in B )<br>0001_0000_0001_1000_1011 ( output placed in A ) |
| Mode value for ECT_MOD_MUL_R2_TEQ | 0001_0000_0100_1000_1011 ( output placed in B )<br>0001_0000_0101_1000_1011 ( output placed in A ) |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero.<br>• E = key, scalar multiplier (k), any integer<br>• [A0, A1] = multiplicand, an input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• B1 = R2 mod N, pre-computed as described in MOD_R2<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four equal-size segments of A and B memory locations. |
| Output | • P[B1, B2] (or P[A0, A1], if A output selected) = E x P[A0, A1], where "x" denotes elliptic curve scalar point multiplication. Output is in affine coordinates (x,y).<br>• B0 = undefined<br>• B3 = undefined |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• Point coordinates A0 and A1 and elliptic curve parameters A3 and B0 are elements of the prime field formed by the modulus N. |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Key Size Error is set if size of E = 0 or size of E > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity. |

The following special cases should be noted:

- For k = 0, this function returns a point at infinity; that is, (0,0) if curve parameter "b" is nonzero and (0,1) otherwise.
- For k < 0, (that is, a negative scalar multiplication is required), its absolute value should be provided to the PKHA; that is, k = abs(- k). After the computation is complete, the formula -P = (x,-y) can be used to compute the "y" coordinate of the effective final result, and other coordinate is the same.

### 11.1.6.57 ECT $F_p$ Point Add, Affine Coordinates (ECT_MOD_ADD) Function

ECT_MOD_ADD performs an addition of two points on an elliptic curve. The inputs and output are in affine coordinates.

**Table 11-59. ECT_MOD_ADD function properties**

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_1000_1001 ( output placed in B )<br>0000_0000_000***1***_1000_1001 ( output placed in A ) |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero<br>• [A0, A1] = first addend in affine coordinates<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• [B1, B2] = second addend in affine coordinates<br>• B3 = ignored |
| Output | [B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates. |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field and therefore are less than the modulus N. |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error will be set if size of B is greater than size of N.<br>• Divide-By-Zero Error is set if the most-significant digit of N = 0. |
| Flags set | None |

### 11.1.6.58 ECT $F_p$ Point Add, Affine Coordinates, $R^2$ Mod N Input (ECT_MOD_ADD_R2) Function

ECT_MOD_ADD_R2 performs an addition of two points on an elliptic curve. The addends are input and the sum is output in affine coordinates. Since ECT_MOD_ADD_R2 has $R^2$ mod N as an additional input, this function is more efficient than ECT_MOD_ADD, which first must compute $R^2$ mod N before performing the addition.

Table 11-60. ECT_MOD_ADD_R2 function properties

| Property | Notes |
|---|---|
| Mode value | 0001_0000_000***0***_1000_1001 ( output placed in B )<br>0001_0000_000***1***_1000_1001 ( output placed in A ) |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero The most significant digit of N must be non-zero<br>• [A0, A1] = first addend point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• [B1, B2] = second addend point in affine coordinates (x,y)<br>• B3 = R2 ($R^2$ mod N) input |
| Output | [B1, B2] (or [A0, A1], if A output selected) = [A0, A1] + [B1, B2], where "+" represents an elliptic curve point addition. Output is in affine coordinates. |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 128 bytes<br>• Point coordinates A0, A1, B1 and B2, and elliptic curve parameters A3 and B0 are elements of the prime field formed by N. |
| Side effects | A0, A1, A2, A3 and B3 are modified. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 128.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if the size of A is greater than size of N.<br>• B Size Error will be set if size of B is greater than size of N.<br>• Divide-By-Zero Error is set if the most-significant digit of N = 0. |
| Flags set | None |

### 11.1.6.59 ECT $F_p$ Check Point (ECT_MOD_CHECK_POINT) Function

ECT_MOD_CHECK_POINT determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $y^3 = x^3 + ax + b$.

ECT_MOD_CHECK_POINT checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2$ mod N and $x^3 + ax + b$ mod N. If they are equal, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged.

Table 11-61. ECT_MOD_CHECK_POINT function properties

| Property | Notes |
|---|---|
| Mode value | 0000_0000_0000_1001_1100 |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero.<br>• [A0, A1] = a possible input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter |

*Table continues on the next page...*

Table 11-61. ECT_MOD_CHECK_POINT function properties (continued)

| Property | Notes |
|---|---|
| | • B1 = ignored<br>• B2 = ignored<br>• B3 = ignored<br>• A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations. |
| Output | • B2 = $R^2$ mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | Various quadrants are modified, but inputs are unchanged. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity.<br>GCD is set if the point is on the curve (but not point at infinity). |

## 11.1.6.60 ECT $F_p$ Check Point, $R^2$ (ECT_MOD_CHECK_POINT_R2) Function

ECT_MOD_CHECK_POINT_R2 determines whether the point (x,y) is on the elliptic curve, i.e. whether x and y satisfy the equation $y^3 = x^3 + ax + b$.

ECT_MOD_CHECK_POINT_R2 checks whether x and y are < N. If not, the routine exits with no flags set. If the input is O, the point at infinity, then PIZ is set and the routine exits. It then computes $y^2$ mod N and $x^3 + ax + b$ mod N. If they are equal, then the (x,y) coordinates are on the curve and the GCD flag is set. Otherwise, no flags are set meaning that (x,y) are not part of the curve, so the point is invalid. All inputs remain unchanged. Since ECT_MOD_CHECK_POINT_R2 has $R^2$ mod N as an additional input, this function is more efficient than ECT_MOD_CHECK_POINT, which first must compute $R^2$ mod N before performing the operaton.

Table 11-62. ECT_MOD_CHECK_POINT_R2 function properties

| Property | Notes |
|---|---|
| Mode value | 0001_0000_0000_1001_1100 |
| Input | • N = modulus, a prime number. The most significant digit of N must be non-zero.<br>• [A0, A1] = a possible input point in affine coordinates (x,y)<br>• A2 = ignored<br>• A3 = elliptic curve "a" parameter<br>• B0 = elliptic curve "b" parameter<br>• B1 = ignored<br>• B2 = ignored |

*Table continues on the next page...*

Table 11-62. ECT_MOD_CHECK_POINT_R2 function properties (continued)

| Property | Notes |
|---|---|
| | • B3 = ignored<br>• A0-A3 and B0-B3 are four, equal-size segments of A and B memory locations. |
| Output | • B2 = $R^2$ mod N |
| Requirements | • Minimum modulus size = 1 byte<br>• Maximum modulus size = 512 bytes |
| Side effects | Various quadrants are modified, but inputs are unchanged. |
| Errors reported | • Data Size Error is set if size of N = 0 or size of N > 512.<br>• Modulus Even Error is set if N is even.<br>• A Size Error is set if size of A is greater than size of N.<br>• B Size Error is set if size of B is greater than size of N.<br>• Divide-by-Zero Error is set if the most significant digit of N = 0. |
| Flags set | PIZ is set if the result is the point at infinity.<br>GCD is set if the point is on the curve (but not point at infinity). |

### 11.1.6.61 Copy memory, N-Size and Source-Size (COPY_NSZ and COPY_SSZ)

These functions copy data from a PKHA register (or register quadrant) specified as a source, to another PKHA register (or register quadrant) specified as a destination. COPY NSZ copies the amount of data specified by the N Size register. COPY_SSZ copies the amount of data specified in the source register's size register. The source and destination are specified in the Mode Value. The source can be A, B or N. The destination can be A, B, E or N, but not the same as the source.

In a quadrant copy, when NSZ/SSZ exceeds the length of a quadrant, the copy will carry on into the next (higher-numbered) quadrant(s).

When the copy operation has completed, the destination register's size register will be updated to contain the number of bytes copied.

Table 11-63. COPY_NSZ and COPY_SSZ function properties

<table>
<tr><th>Property</th><th>Notes</th></tr>
<tr><td>Mode value</td><td>
<table>
<tr><th>Bits 19:17</th><th>Bits 16,11,10</th><th>Bits 9:8</th><th>Bits 7:6</th><th>Bits 5:0</th></tr>
<tr><th>Source Register</th><th>Destination Register</th><th>Source Segment</th><th>Destination Segment</th><th>Function Code</th></tr>
<tr><td>000 = A Register</td><td>000 = A Register</td><td>00 = Segment 0</td><td>00 = Segment 0</td><td rowspan="4">01_0000 = Copy_NSZ<br>01_0001 = Copy_SSZ</td></tr>
<tr><td>001 = B Register</td><td>001 = B Register</td><td>01 = Segment 1</td><td>01 = Segment 1</td></tr>
<tr><td>011 = N Register</td><td>011 = N Register</td><td>10 = Segment 2</td><td>10 = Segment 2</td></tr>
<tr><td></td><td>010 = E Register</td><td>11 = Segment 3</td><td>11 = Segment 3</td></tr>
</table>
</td></tr>
</table>

Table continues on the next page...

Table 11-63. COPY_NSZ and COPY_SSZ function properties (continued)

<table>
<tr><th>Property</th><th>Notes</th></tr>
<tr><td></td><td>
<table>
<tr><th>Bits 19:17</th><th>Bits 16,11,10</th><th>Bits 9:8</th><th>Bits 7:6</th><th>Bits 5:0</th></tr>
<tr><th>Source Register</th><th>Destination Register</th><th>Source Segment</th><th>Destination Segment</th><th>Function Code</th></tr>
<tr><td>other values reserved</td><td>other values reserved</td><td></td><td></td><td></td></tr>
</table>
1. If the destination register is E, the source and destination seqments must be 00b.
</td></tr>
<tr><td>Input</td><td>None</td></tr>
<tr><td>Output</td><td>None</td></tr>
<tr><td>Requirements</td><td>For Copy_NSZ, the N-size Register must contain a valid value.<br>For Copy_SSZ, the source register's size register must contain a valid value.</td></tr>
<tr><td>Side effects</td><td>The destination register's size register is updated to the number of bytes copied.</td></tr>
<tr><td>Errors reported</td><td>None</td></tr>
<tr><td>Flags set</td><td>None</td></tr>
</table>

1. If the destination register is E, the source and destination seqments must be 00b.

### 11.1.6.62 Right Shift A (R_SHIFT) function

Table 11-64. R_SHIFT function properties

| Property | Notes |
|---|---|
| Mode value | 0000_0000_000***0***_0001_1101 ( output placed in B )<br>0000_0000_000***1***_0001_1101 ( output placed in A ) |
| Input | • A = Input value to be shifted. Bytes above A Size will be assumed to be zero, regardless of the contents of the PKHA A RAM.<br>• B = Number of bit positions that the A RAM will be shifted (Only the least-significant two bytes are used. The upper bytes are ignored.) |
| Output | B (or A, if selected) = the contents of PKHA A RAM (with zeros substituted for bytes above A Size), right-shifted by the number of bit positions specified in the least-significant two bytes of PKHA B RAM, and zero-filled on the left. |
| Requirements | none |
| Side effects | B (or A) is modified. |
| Errors reported | none |
| Flags set | none |

### 11.1.6.63 Compare A B (COMPARE) function

Table 11-65. COMPARE function properties

| Property | Notes |
|---|---|
| Mode value | 0000_0000_0000_0001_1110 |
| Input | • A = Value to be compared<br>• B = Value to be compared<br>• A Size = the number of least-significant bits that will be compared |
| Output | None (other than flags) |
| Requirements | A Size must be &gt;= B Size. |
| Side effects | none |
| Errors reported | If B Size &gt; A Size a "B Size" error will be generated. |
| Flags set | • PKHA_GCD_ONE is set if B &gt; A<br>• PKHA_ZERO is set if B == A<br>• no flag is set if B &lt; A |

### 11.1.6.64 Evaluate A (EVALUATE) function

Table 11-66. EVALUATE function properties

| Property | Notes |
|---|---|
| Mode value | ***SB***00_0000_000***0***_0001_1111 ( output placed in B )<br>***SB***00_0000_000***1***_0001_1111 ( do not modify B )<br>• If the ***S*** bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_0**sss**, where **sss** is the updated A Size.<br>• If the ***B*** bit is set, PKHA will push to the output FIFO a single DWord with the value 000_000_000_**bbbb**, where **bbbb** is the updated number of bits in A.<br>• If the ***S*** bit is set and the ***B*** bit is set, the A-Size DWord will be pushed before the number-of-bits-in-A Dword. |
| Input | • A = Value to be evaluated |
| Output | • A Size is updated with the number of least-significant non-zero bytes, i.e. the position of the most-significant non-zero byte (least-significant byte is byte-position 0). This evaluation considers only the bytes specified by the incoming value of A Size. This allows the incoming value of A Size to be set so that A0 will be evaluated, ignoring the values in A1, A2 and A3, or the incoming value could be set so that (A1,A0) will be evaluated, ignoring A2 and A3, or (A2,A1,A0) could be evaluated, ignoring A3.<br>• If the mode value specifies that the output is to be placed in B, the updated value of A Size will be copied into the least-significant two bytes of B and B Size will be set to 2. If the "do not modify B" option is selected, the updated value of A Size will not be copied into B. |
| Requirements | none |
| Side effects | • A Size will be modified.<br>• B Size may be modified.<br>• One or two DWords mmay be pushed to the output FIFO. |
| Errors reported | none |
| Flags set | • PKHA_GCD_ONE is set if A == 1<br>• PKHA_ZERO is set if A == 0. A Size (and number of bits in A) will be set to 0. Note that this could cause an A Size error in a subsequent PKHA operation. |

### 11.1.7 Special values for common ECC domains

Software can sometimes use the PKHA more effectively if the Montgomery Conversion Factor ($R^2$ mod N) is either provided or previously used to convert other inputs into Montgomery form. For convenience, the conversion factors for common ECC domains have been computed and published here. Some of the other domain values are provided to aid in definite identification of the domain, in the case that the name is not found or is not an exact match.

The following tables give these values for the *q* and *r* modulus values found in ECC domains. These associated Montgomery values are dependent upon the PKHA digit size (16, 32, 64, 128). These tables are for a PKHA with a 32-bit digit.

ECC $F_{2^m}$ requires a *c* (also called *b'*) parameter for the elliptic curve in place of the *b* value. Table 11-68 provides these values in addition to the Montgomery values. The *b'* values are universal and do not change with PKHA digit size.

The following variable definitions apply to both tables. Variable names (*q, r, b, c*) follow the conventions of IEEE Std 1363.

**Name**

The names in this table are associated with, or named in, various published standards. Neither the names nor the domains are guaranteed to be complete. Two values of the domain parameters are provided for purposes of identification.

- Those beginning with "P-", "K-", and "B-" are in FIPS 186 from NIST, found at www.csrc.nist.gov
- Those beginning with "ansix9" are names from ANS X9.62-2005; those beginning with "prime" or "c2pnb" are from an earlier ANSI document
- Those beginning with "sec" are from SEC 2 from the Standards for Efficient Cryptography group, found at www.secg.org
- Those beginning with "wtls" are taken from Wireless Transport Layer Security / Wireless Access Protocol, Version 06-Apr-2001, WAP-261-WTLS-20010406-a. Not all software libraries agree with the mapping of these names to values; care has been taken to identify the values based upon the source documentation.
- Those beginning with "ECDSA", "ECP", "EC2N", "ecp_group", and "Oakley" are from various RFCs found at www.ietf.org
- Those beginning with "GOST" are from the Russian standard GOST R 3410-2001
- Those beginning with "brainpool" are from ECC Brainpool, found at www.ecc-brainpool.org and republished in RFC 5639

**R**

*R* is the Montgomery factor. Its value is $2^{SD}$, where *D* is the PKHA digit size in bits, and *S* is the minimum number of digits needed to hold the modulus. As an example, for a modulus of nine bytes (72 bits), *R* would be

- $2^{80}$ for a PKHA with digit size of 16 bits
- $2^{96}$ for a PKHA with digit size of 32 bits
- $2^{128}$ for a PKHA with digit size of 64 or 128 bits

**q**

This is the *field-defining* value for the elliptic curve. For $F_p$ curves, it is the prime number used as the modulus for all point arithmetic; it is named *p* in some other publications. For $F_{2^m}$ curves, it is the irreducible binary polynomial used as the modulus for all point arithmetic. It is not, as usually defined, $q = 2^m$, i.e. the size of the field.

**L**

This is the number of bytes needed to hold *q* and each of its associated values: *R2modq*, *a*,*b*,*c*, the point coordinates *x* and *y*, the result of an ECDH key agreement, etc.

**R2modq**

This is $R^2 \bmod q$, the Montgomery Conversion Factor when *q* is the modulus.

**r**

This is the (usually prime) number which is the order of *G*, the generator point. It is also usually used as the modulus for the non-ECC-related arithmetic in an ECC primitive. This variable is named *n* in some other publications.

**N**

This is the number of bytes needed to hold *r* and each of its associated values: *R2modr*, private keys, each of the two components of an ECDSA signature, etc.

**R2modr**

This is $R^2 \bmod r$, the Montgomery Conversion Factor when *r* is the modulus.

**b / c (b')**

*b* is the coefficient for the $x^0$ (ones) term in an $F_{2^m}$ elliptic curve equation. Its relationship with *c* is $b = c^4$. *c* is sometimes referred to as *b'* in NXP documentation.

**A / a24**

*a24* is the special value derived from the A coefficient for the $y^2$term in a Montgomery-form elliptic curve equation. Its relationship with *A* is $a24 = (A+2)/4$.

The domains in the table are ordered by size.

**Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits**

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| secp112r1 | 14 | 14 | |
| wtls6 | q | | 0xDB7C2ABF62E35E668076BEAD208B |

Table continues on the next page...

## Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | | | 4451685225093714772084598273548427 |
| | R2modq | | 0x0000000000000000000000000009 |
| | r | | 0xDB7C2ABF62E35E7628DFAC6561C5<br>4451685225093714776491891542548933 |
| | R2modr | | 0xDA4A43AD7F34245D42B9C948C559 |
| secp112r2 | 14 | 14 | |
| | q | | 0xDB7C2ABF62E35E668076BEAD208B<br>4451685225093714772084598273548427 |
| | R2modq | | 0x0000000000000000000000000009 |
| | r | | 0x36DF0AAFD8B8D7597CA10520D04B<br>1112921306273428674967732714786891 |
| | R2modr | | 0x2049C67E5F79E8C06B7825955374 |
| wtls8 | 14 | 15 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFDE7<br>5192296858534827628530496329219559<br>$2^{112}$ - $2^{9}$ - $2^{4}$ - $2^{3}$ - 1 |
| | R2modq | | 0x0000000000000004667100000000 |
| | r | | 0x0100000000000001ECEA551AD837E9<br>5192296858534827767273836114360297 |
| | R2modr | | 0x00E074FD104C86569DB6C204A52932 |
| secp128r1 | 16 | 16 | |
| | q | | 0xFFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFF<br>340282366762482138434845932244680310783<br>$2^{128}$ - $2^{97}$ - 1 |
| | R2modq | | 0x00000024000000040000000800000011 |
| | r | | 0xFFFFFFFE0000000075A30D1B9038A115<br>340282366762482138443322565580356624661 |
| | R2modr | | 0x71875047CDD8151626BC6448FADE9BED |
| secp128r2 | 16 | 16 | |
| | q | | 0xFFFFFFFDFFFFFFFFFFFFFFFFFFFFFFFF<br>340282366762482138434845932244680310783<br>$2^{128}$ - $2^{97}$ - 1 |
| | R2modq | | 0x00000024000000040000000800000011 |
| | r | | 0x3FFFFFFF7FFFFFFFBE0024720613B5A3<br>85070591690620534603955721926813660579 |
| | R2modr | | 0x0EFCA409C09D126A99CD2E9404A3B434 |
| secp160k1 | 20 | 21 | |

*Table continues on the next page...*

## Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| ansix9p160k1 | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFAC73<br>1461501637330902918203684832716283019651637554291<br>$2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^{9} - 2^{8} - 2^{7} - 2^{3} - 2^{2} - 1$ |
| | R2modq | | 0x00000000000000000000000010000A71A1B44BBA9 |
| | r | | 0x0100000000000000000001B8FA16DFAB9ACA16B6B3<br>1461501637330902918203686915170869725397159163571 |
| | R2modr | | 0x00CDCF2BABDFE35D2F4D8A8AAD0F8494330E687AAF |
| secp160r1<br>ansix9p160r1<br>wtls7 | 20 | 21 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFF<br>1461501637330902918203684832716283019653785059327<br>$2^{160} - 2^{31} - 1$ |
| | R2modq | | 0x0000000000000000000000004000000100000001 |
| | r | | 0x0100000000000000000001F4C8F927AED3CA752257<br>1461501637330902918203687197606826779884643492439 |
| | R2modr | | 0x00A0E626837A981E4B3CDC3854085E335F6744F8A4 |
| secp160r2<br>ansix9p160r2 | 20 | 21 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFAC73<br>1461501637330902918203684832716283019651637554291<br>$2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^{9} - 2^{8} - 2^{7} - 2^{3} - 2^{2} - 1$ |
| | R2modq | | 0x00000000000000000000000010000A71A1B44BBA9 |
| | r | | 0x0100000000000000000000351EE786A818F3A1A16B<br>1461501637330902918203685083571792140653176136043 |
| | R2modr | | 0x0076E5A1814769EF9E8DD4D69E29AEB02AD8C126C7 |
| wtls9 | 20 | 21 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC808F<br>1461501637330902918203684832716283019655932313743<br>$2^{160} - 2^{17} - 2^{16} - 2^{14} - 2^{13} - 2^{12} - 2^{11} - 2^{10} - 2^{9} - 2^{8} - 2^{6} - 2^{5} - 2^{4} - 1$ |
| | R2modq | | 0x0000000000000000000000000000000C3C174FE1 |
| | r | | 0x0100000000000000000001CDC98AE0E2DE574ABF33<br>1461501637330902918203687013445034429194588307251 |
| | R2modr | | 0x00CC3AB9A731EBB2AA87D1BED0AEF6B4CF5840D789 |
| brainpoolP160r1 | 20 | 20 | |
| | q | | 0xE95E4A5F737059DC60DFC7AD95B3D8139515620F<br>1332297598440044874827085558802491743757193798159 |
| | R2modq | | 0x6CF12F81C0CA7EF8FED717E0B333F8D625BC14FF |
| | r | | 0xE95E4A5F737059DC60DF5991D45029409E60FC09<br>1332297598440044874827085038830181364212942568457 |

*Table continues on the next page...*

## Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | R2modr | | 0x2BC73851FC9BE6F69E31FE16FC61D4351FDF90EA |
| brainpoolP160t1 | 20 | 20 | |
| | q | | 0xE95E4A5F737059DC60DFC7AD95B3D8139515620F |
| | | | 1332297598440044874827085558802491743757193798159 |
| | R2modq | | 0x6CF12F81C0CA7EF8FED717E0B333F8D625BC14FF |
| | r | | 0xE95E4A5F737059DC60DF5991D45029409E60FC09 |
| | | | 1332297598440044874827085038830181364212942568457 |
| | R2modr | | 0x2BC73851FC9BE6F69E31FE16FC61D4351FDF90EA |
| P-192 | 24 | 24 | |
| secp192r1 | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF |
| ansix9p192r1 | | | 6277101735386680763835789423207666416083908700390324961279 |
| prime192v1 | | | $2^{192}$ - $2^{64}$ - 1 |
| ECPRGF192Random | R2modq | | 0x000000000000000100000000000000020000000000000001 |
| | r | | 0xFFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831 |
| | | | 6277101735386680763835789423176059013767194773182842284081 |
| | R2modr | | 0x28BE5677EA0581A24696EA5BBB3A6BEECE66BACCDEB35961 |
| secp192k1 | 24 | 24 | |
| ansix9p192k1 | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFEE37 |
| | | | 6277101735386680763835789423207666416102355444459739541047 |
| | | | $2^{192}$ - $2^{32}$ - $2^{12}$ - $2^{8}$ - $2^{7}$ - $2^{6}$ - $2^{3}$ - 1 |
| | R2modq | | 0x000000000000000000000000000000010000239201 3C4FD1 |
| | r | | 0xFFFFFFFFFFFFFFFFFFFFFFFE26F2FC170F69466A74DEFD8D |
| | | | 6277101735386680763835789423061264271957123915200845512077 |
| | R2modr | | 0x6A21191C2EC4B2B1F0F4F172195E97E2461C1989250F0702 |
| prime192v2 | 24 | 24 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF |
| | | | 6277101735386680763835789423207666416083908700390324961279 |
| | | | $2^{192}$ - $2^{64}$ - 1 |
| | R2modq | | 0x000000000000000100000000000000020000000000000001 |
| | r | | 0xFFFFFFFFFFFFFFFFFFFFFFFE5FB1A724DC80418648D8DD31 |
| | | | 6277101735386680763835789423078825936192100537584385056049 |
| | R2modr | | 0xA4FEB8C277C030E139DA8CFB4E35E1F62814A261001BE8FF |
| prime192v3 | 24 | 24 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF |
| | | | 6277101735386680763835789423207666416083908700390324961279 |
| | | | $2^{192}$ - $2^{64}$ - 1 |
| | R2modq | | 0x000000000000000100000000000000020000000000000001 |

*Table continues on the next page...*

## Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | r | | 0xFFFFFFFFFFFFFFFFFFFFFFFF7A62D031C83F4294F640EC13<br>6277101735386680763835789423166314882687165660350679936019 |
| | R2modr | | 0x45BCB42FF1CC05D0194D076B366D09BF0305982367330969 |
| brainpoolP192r1 | 24 | 24 | |
| | q | | 0xC302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297<br>4781668983906166242955001894344923773259119655253013193367 |
| | R2modq | | 0xB6225126EED34F1033BF484602C3FE69E2474C6972C7B21A |
| | r | | 0xC302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1<br>4781668983906166242955001894269038308119863659119834868929 |
| | R2modr | | 0x98769B9CE772102BBF4AFD5DBF53AFF0B4727C80E407E8F8 |
| brainpoolP192t1 | 24 | 24 | |
| | q | | 0xC302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297<br>4781668983906166242955001894344923773259119655253013193367 |
| | R2modq | | 0xB6225126EED34F1033BF484602C3FE69E2474C6972C7B21A |
| | r | | 0xC302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1<br>4781668983906166242955001894269038308119863659119834868929 |
| | R2modr | | 0x98769B9CE772102BBF4AFD5DBF53AFF0B4727C80E407E8F8 |
| P-224<br>secp224r1<br>ansix9p224r1<br>wtls12<br>ECPRGF224Random | 28 | 28 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF000000000000000000000001<br>26959946667150639794667015087019630673557916260026308143510066298881 |
| | R2modq | | 0x00000000FFFFFFFFFFFFFFFFFFFFFFFE000000000000000000000001 |
| | r | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFF16A2E0B8F03E13DD29455C5C2A3D<br>26959946667150639794667015087019625940457807714424391721682722368061 |
| | R2modr | | 0xD4BAA4CF1822BC47B1E979616AD09D9197A545526BDAAE6C3AD01289 |
| secp224k1<br>ansix9p224k1 | 28 | 29 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFE56D<br>26959946667150639794667015087019630673637144422540572481099315275117<br>$2^{224} - 2^{32} - 2^{12} - 2^{11} - 2^{9} - 2^{7} - 2^{4} - 2^{1} - 1$ |
| | R2modq | | 0x00000000000000000000000000000000000000010000352602C23069 |
| | r | | 0x010000000000000000000000000001DCE8D2EC6184CAF0A971769FB1F7<br>26959946667150639794667015087019640346510327083120074548994958668279 |
| | R2modr | | 0x00993FF72BB882BD88BBFF32E48BE0320816F60AF534CE24FBEC9FEAA0 |
| brainpoolP224r1 | 28 | 28 | |
| | q | | 0xD7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF<br>22721622932454352787552537995910928073340732145944992304435472941311 |
| | R2modq | | 0x0578FD592E6A6CE43FE8A2AA96AF774C43C20E727867CA8064DCD04F |
| | r | | 0xD7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F |

*Table continues on the next page...*

## Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | | | 22721622932454352787552537995910923612567546342330757191396560966559 |
| | R2modr | | 0x4A73A6563211A5611E9CAE249F24919B9399652CADDAF8AA486CA401 |
| brainpoolP224t1 | 28 | 28 | |
| | q | | 0xD7C134AA264366862A18302575D1D787B09F075797DA89F57EC8C0FF<br>22721622932454352787552537995910928073340732145944992304435472941311 |
| | R2modq | | 0x0578FD592E6A6CE43FE8A2AA96AF774C43C20E727867CA8064DCD04F |
| | r | | 0xD7C134AA264366862A18302575D0FB98D116BC4B6DDEBCA3A5A7939F<br>22721622932454352787552537995910923612567546342330757191396560966559 |
| | R2modr | | 0x4A73A6563211A5611E9CAE249F24919B9399652CADDAF8AA486CA401 |
| prime239v1 | 30 | 30 | |
| | q | | 0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFFF<br>883423532389192164791648750360308885314476597252960362792450860609699839 |
| | R2modq | | 0x0000000000000005000000000005FFFFFFFFFFFC000000000008000000000 |
| | r | | 0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFF9E5E9A9F5D9071FBD1522688909D0B<br>883423532389192164791648750360308884807550341691627752275345424702807307 |
| | R2modr | | 0x2BE4B1BE15BDEB5DF3096A7BE4944FA0CB87DC9852A129052EC789ED615B |
| prime239v2 | 30 | 30 | |
| | q | | 0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFFF<br>883423532389192164791648750360308885314476597252960362792450860609699839 |
| | R2modq | | 0x0000000000000005000000000005FFFFFFFFFFFC000000000008000000000 |
| | r | | 0x7FFFFFFFFFFFFFFFFFFFFFFF800000CFA7E8594377D414C03821BC582063<br>883423532389192164791648750360308886392687657546993855147765732451295331 |
| | R2modr | | 0x76CF025EBF73DDE8A5D15F0C7C29FF23EED0AE5C096A0D32ABCB4B16B765 |
| prime239v3 | 30 | 30 | |
| | q | | 0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFFFFFFFF8000000000007FFFFFFFFFFF<br>883423532389192164791648750360308885314476597252960362792450860609699839 |
| | R2modq | | 0x0000000000000005000000000005FFFFFFFFFFFC000000000008000000000 |
| | r | | 0x7FFFFFFFFFFFFFFFFFFFFFFF7FFFFF975DEB41B3A6057C3C432146526551<br>883423532389192164791648750360308884771190369765922550517967171058034001 |
| | R2modr | | 0x11500EB94E46F16737BEB7A266592D93C18845A5EB3F814C07B00EA6ACF5 |
| P-256<br>secp256r1<br>ansix9p256r1 | 32 | 32 | |
| | q | | 0xFFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF |

Table continues on the next page...

## Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| prime256v1<br>ECDSA-256<br>ecp_group_19<br>ECPRGF256Random | | | 115792089210356248762697446949407573530086143415290314195533631308867097853951 |
| | R2modq | | 0x00000004FFFFFFFDFFFFFFFFFFFFFFFEFFFFFFFBFFFFFFFF0000000000000003 |
| | r | | 0xFFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551<br>115792089210356248762697446949407573529996955224135760342422259061068512044369 |
| | R2modr | | 0x66E12D94F3D956202845B2392B6BEC594699799C49BD6FA683244C95BE79EEA2 |
| secp256k1 | 32 | 32 | |
| ansix9p256k1 | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F<br>115792089237316195423570985008687907853269984665640564039457584007908834671663<br>$2^{256} - 2^{32} - 2^{9} - 2^{8} - 2^{7} - 2^{6} - 2^{4} - 1$ |
| | R2modq | | 0x000000000000000000000000000000000000000000000001000007A2000E90A1 |
| | r | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141<br>115792089237316195423570985008687907852837564279074904382605163141518161494337 |
| | R2modr | | 0x9D671CD581C69BC5E697F5E45BCD07C6741496C20E7CF878896CF21467D7D140 |
| brainpoolP256r1 | 32 | 32 | |
| | q | | 0xA9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377<br>76884956397045344220809746629001649093037950200943055203735601445031516197751 |
| | R2modq | | 0x4717AA21E5957FA8A1ECDACD6B1AC8075CCE4C26614D4F4D8CFEDF7BA6465B6C |
| | r | | 0xA9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7<br>76884956397045344220809746629001649092737531784414529538755519063063536359079 |
| | R2modr | | 0x0B25F1B9C32367629B7F25E76C815CB0F35D176A1134E4A0E1D8D8DE3312FCA6 |
| brainpoolP256t1 | 32 | 32 | |
| | q | | 0xA9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377<br>76884956397045344220809746629001649093037950200943055203735601445031516197751 |
| | R2modq | | 0x4717AA21E5957FA8A1ECDACD6B1AC8075CCE4C26614D4F4D8CFEDF7BA6465B6C |

*Table continues on the next page...*

# Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | r | | 0xA9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7<br>7688495639704534422080974662900164909273753178441452953875551906306353 6359079 |
| | R2modr | | 0x0B25F1B9C32367629B7F25E76C815CB0F35D176A1134E4A0E1D8D8DE3312FCA6 |
| GOSTR3410-CryptoPro-A | 32 | 32 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD97<br>115792089237316195423570985008687907853269984665640564039457584007913129639319<br>$2^{256} - 2^9 - 2^6 - 2^5 - 2^3 - 1$ |
| | R2modq | | 0x000000000000000000000000000000000000000000000000000000000005CF11 |
| | r | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF6C611070995AD10045841B09B761B893<br>115792089237316195423570985008687907853073762908499243225378155805079068850323 |
| | R2modr | | 0x551FE9CB451179DBF74885D08A3714C6FB07F8222E76DD529AC2D7858E79A469 |
| GOSTR3410-CryptoPro-B | 32 | 32 | |
| | q | | 0x8000000000000000000000000000000000000000000000000000000000000C99<br>5789604461865809771178549250434395392663499233282028201972879200395656 4823193 |
| | R2modq | | 0x00000000000000000000000000000000000000000000000000000000027ACDC4 |
| | r | | 0x800000000000000000000000000000015F700CFFF1A624E5E497161BCC8A198F<br>5789604461865809771178549250434395392710213316025582682006884449608773 2066703 |
| | R2modr | | 0x09D1D2C4E50824664A2E7E2F6882CF102A3104A7EA43E85529B721F4E6CD7823 |
| GOSTR3410-CryptoPro-C | 32 | 32 | |
| | q | | 0x9B9F605F5A858107AB1EC85E6B41C8AACF846E86789051D37998F7B9022D759B<br>7039008535208330519954771801901843784107951663004518047128434684370563 3502619 |
| | R2modq | | 0x807A394EDE097652186304212849C07B1017BB39C2D346C5409973B4C427FCEA |
| | r | | 0x9B9F605F5A858107AB1EC85E6B41C8AA582CA3511EDDFB74F02F3A6598980BB9<br>7039008535208330519954771801901843784092088264716408103532260145835229 8396601 |
| | R2modr | | 0x7AA61B49A49D4759C67E5D0EE96E8ED304FDA8694AFDA24BE94FAAB66ABA180E |

*Table continues on the next page...*

## Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| brainpoolP320r1 | 40 | 40 | |
| | q | | 0xD35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B32E27<br>1763593322239166354161909842446019520889512772719515192772960415288640868802149818095501499903527 |
| | R2modq | | 0xA259BA4A6C2D92525455A964E614D6D21F4C881F30C5B676C2478A8D906978EF994EE88A743B52F9 |
| | r | | 0xD35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E98691555B44C59311<br>1763593322239166354161909842446019520889512772717686063760686124016784784845843468355685258203921 |
| | R2modr | | 0x31EC87C73200B14FE30D35244E6390FE86B330BCAF86C40991C3001BE0E16805679D29DF2513E4CD |
| brainpoolP320t1 | 40 | 40 | |
| | q | | 0xD35E472036BC4FB7E13C785ED201E065F98FCFA6F6F40DEF4F92B9EC7893EC28FCD412B1F1B32E27<br>1763593322239166354161909842446019520889512772719515192772960415288640868802149818095501499903527 |
| | R2modq | | 0xA259BA4A6C2D92525455A964E614D6D21F4C881F30C5B676C2478A8D906978EF994EE88A743B52F9 |
| | r | | 0xD35E472036BC4FB7E13C785ED201E065F98FCFA5B68F12A32D482EC7EE8658E98691555B44C59311<br>1763593322239166354161909842446019520889512772717686063760686124016784784845843468355685258203921 |
| | R2modr | | 0x31EC87C73200B14FE30D35244E6390FE86B330BCAF86C40991C3001BE0E16805679D29DF2513E4CD |
| P-384<br>secp384r1<br>ansix9p384r1<br>ECDSA-384<br>ecp_group_20<br>ECPRGF384Random | 48 | 48 | |
| | q | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFF0000000000000000FFFFFFFF<br>39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319 |
| | R2modq | | 0x00000000000000000000000000000001000000020000000 0FFFFFFFE000000000000000200000000FFFFFFFE00000001 |
| | r | | 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC7634D81F4372DDF581A0DB248B0A77AECEC196ACCC52973<br>39402006196394479212279040100143613805079739270465446667946905279627659399113263569398956308152294913554433653942643 |
| | R2modr | | 0x0C84EE012B39BF213FB05B7A28266895D40D49174AAB1CC5BC3E483AFCB82947FF3D81E5DF1AA4192D319B2419B409A9 |
| brainpoolP384r1 | 48 | 48 | |
| | q | | 0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123ACD3A729901D1A71874700133107EC53 |

*Table continues on the next page...*

### Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | | | 21659270770119316173069236842332604979796116387017648600081618503821089934025961822236561982844534088440708417973331 |
| | R2modq | | 0x36BF6883178DF842D5C6EF3BA57E052C621401919918D5AF8E28F99CC9940899535283343D7FD965087CEFFF40B64BDE |
| | r | | 0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A7CF3AB6AF6B7FC3103B883202E9046565<br>21659270770119316173069236842332604979796116387017648600075645274821611501358515537962695117368903252229601718723941 |
| | R2modr | | 0x0CE8941A614E97C28F886DC965165FDB574A74CB52D748FF2A927E3B9802688A37264E202F2B6B6EAC4ED3A2DE771C8E |
| brainpoolP384t1 | 48 | 48 | |
| | q | | 0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B412B1DA197FB71123ACD3A729901D1A71874700133107EC53<br>21659270770119316173069236842332604979796116387017648600081618503821089934025961822236561982844534088440708417973331 |
| | R2modq | | 0x36BF6883178DF842D5C6EF3BA57E052C621401919918D5AF8E28F99CC9940899535283343D7FD965087CEFFF40B64BDE |
| | r | | 0x8CB91E82A3386D280F5D6F7E50E641DF152F7109ED5456B31F166E6CAC0425A7CF3AB6AF6B7FC3103B883202E9046565<br>21659270770119316173069236842332604979796116387017648600075645274821611501358515537962695117368903252229601718723941 |
| | R2modr | | 0x0CE8941A614E97C28F886DC965165FDB574A74CB52D748FF2A927E3B9802688A37264E202F2B6B6EAC4ED3A2DE771C8E |
| brainpoolP512r1 | 64 | 64 | |
| | q | | 0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F3<br>8948962207650232551656602815159153422162609644098354511344597187200057010413552439917934304191956942765446530386427345937963894309923928536070534607816947 |
| | R2modq | | 0x3C4C9D05A9FF6450202E19402056EECCA16DAA5FD42BFF8319486FD8D5898057E0C19A7783514A2553B7F9BC905AFFD3793FB1302715790549AD144A6158F205 |
| | r | | 0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA70330870553E5C414CA92619418661197FAC10471DB1D381085DDADDB58796829CA90069<br>8948962207650232551656602815159153422162609644098354511344597187200057010413418528378981730643524959857451398370029280583094215613882043973354392115544169 |
| | R2modr | | 0xA794586A718407B095DF1B4C194B2E56723C37A22F16BBDFD7F9CC263B790DE3A6F230C72F0207E83EC64BD033B7627F0886B75895283DDDD2A3681ECDA81671 |
| brainpoolP512t1 | 64 | 64 | |

*Table continues on the next page...*

### Table 11-67. Special Values for common ECC $F_p$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | q | | 0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA703308717D4D9B009BC66842AECDA12AE6A380E62881FF2F2D82C68528AA6056583A48F3<br>8948962207650232551656602815159153422162609644098354511344597187200057010413552439917934304191956942765446530386427345937963894309923928536070534607816947 |
| | R2modq | | 0x3C4C9D05A9FF6450202E19402056EECCA16DAA5FD42BFF8319486FD8D5898057E0C19A7783514A2553B7F9BC905AFFD3793FB1302715790549AD144A6158F205 |
| | r | | 0xAADD9DB8DBE9C48B3FD4E6AE33C9FC07CB308DB3B3C9D20ED6639CCA70330870553E5C414CA92619418661197FAC10471DB1D381085DDADDB58796829CA90069<br>8948962207650232551656602815159153422162609644098354511344597187200057010413418528378981730643524959857451398370029280583094215613882043973354392115544169 |
| | R2modr | | 0xA794586A718407B095DF1B4C194B2E56723C37A22F16BBDFD7F9CC263B790DE3A6F230C72F0207E83EC64BD033B7627F0886B75895283DDDD2A3681ECDA81671 |
| P-521<br>secp521r1<br>ansix9p521r1<br>ECDSA-521<br>ecp_group_21<br>ECPRGF521Random | 66 | 66 | |
| | q | | 0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF<br>6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151<br>$2^{521}$ - 1 |
| | R2modq | | 0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000400000000000 |
| | r | | 0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFA51868783BF2F966B7FCC0148F709A5D03BB5C9B8899C47AEBB6FB71E91386409<br>6864797660130609714981900799081393217269435300143305409394463459185543183397655394245057746333217197532963996371363321113864768612440380340372808892707005449 |
| | R2modr | | 0x019A5B5A3AFE8C44383D2D8E03D1492D0D455BCC6D61A8E567BCCFF3D142B7756E3A4FB35B72D34027055D4DD6D30791D9DC18354A564374A6421163115A61C64CA7 |

### Table 11-68. Special Values for common ECC $F_{2^m}$ domains when PKHA digit size is 32 bits

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| sect113r1 | 15 | 15 | |
| wtls4 | q | | 0x0200000000000000000000000000201 |

*Table continues on the next page...*

### Table 11-68. Special Values for common ECC $F_{2^m}$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | | | $x^{113} + x^9 + 1$ |
| | R2modq | | 0x000000000000000001000040000000 |
| | b<br>c | | 0xE8BEE4D3E2260744188BE0E9C723<br>0x0173E834AF28EC76CB83BD8DFEB2D5 |
| | r | | 0x0100000000000000D9CCEC8A39E56F<br>5192296858534827689835882578830703 |
| | R2modr | | 0x002D02609ABE76F866BDCE5B3F9BCC |
| sect113r2 | 15 | 15 | |
| | q | | 0x020000000000000000000000000201<br>$x^{113} + x^9 + 1$ |
| | R2modq | | 0x000000000000000001000040000000 |
| | b<br>c | | 0x95E9A9EC9B297BD4BF36E059184F<br>0x0054D9F03957174A32329167D7FE71 |
| | r | | 0x010000000000000108789B2496AF93<br>5192296858534827702972497909952403 |
| | R2modr | | 0x00471CB662E29CB41ABC888E16FF49 |
| wtls1 | 15 | 14 | |
| | q | | 0x020000000000000000000000000201<br>$x^{113} + x^9 + 1$ |
| | R2modq | | 0x000000000000000001000040000000 |
| | b<br>c | | 0x01<br>0x000000000000000000000000000001 |
| | r | | 0xFFFFFFFFFFFFFFFDBF91AF6DEA73<br>5192296858534827627896703833467507 |
| | R2modr | | 0x9A1AB7E0A60C212FBD48A8239130 |
| sect131r1 | 17 | 17 | |
| | q | | 0x080000000000000000000000000000010D<br>$x^{131} + x^8 + x^3 + x^2 + 1$ |
| | R2modq | | 0x0000000000000000401440000000000000 |
| | b<br>c | | 0x0217C05610884B63B9C6C7291678F9D341<br>0x03DB89B405E491160E3B2F07B0CE20B37E |
| | r | | 0x0400000000000000023123953A9464B54D<br>1361129467683753853893932755685365560653 |
| | R2modr | | 0x00739BBCD15B208AC45847F42ED438E023 |
| sect131r2 | 17 | 17 | |
| | q | | 0x080000000000000000000000000000010D<br>$x^{131} + x^8 + x^3 + x^2 + 1$ |

*Table continues on the next page...*

## Table 11-68. Special Values for common ECC $F_{2^m}$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | R2modq | | 0x00000000000000040144000000000000000 |
| | b | | 0x04B8266A46C55657AC734CE38F018F2192 |
| | c | | 0x07CBB9920D71A48E099C38D71DA6490EB1 |
| | r | | 0x0400000000000000016954A233049BA98F |
| | | | 1361129467683753853879535043412812867983 |
| | R2modr | | 0x01BB89631FCB716E50598B58B5058E0389 |
| Oakley 3 | 20 | - | |
| | q | | 0x0800000000000000000000004000000000000001 |
| | | | $x^{155} + x^{62} + 1$ |
| | R2modq | | 0x0000004000000000000000000000000000000400 |
| | b | | 0x07338F |
| | c | | 0x0031100000000223A000C4474000088E8000111D1D |
| B-163 | 21 | 21 | |
| ansix9t163r2 | q | | 0x0800000000000000000000000000000000000000C9 |
| sect163r2 | | | $x^{163} + x^7 + x^6 + x^3 + 1$ |
| EC2NGF163Random | R2modq | | 0x000000000000000000000001410400000000000000 |
| | b | | 0x020A601907B8C953CA1481EB10512F78744A3205FD |
| | c | | 0x072C4E1EF7CB2F3A035D33104294159609138BB404 |
| | r | | 0x040000000000000000000292FE77E70C12A4234C33 |
| | | | 5846006549323611672814742442876390689256843201587 |
| | R2modr | | 0x003488BE6C9C552CFE775F73CFB60B416A9AA88652 |
| K-163 | 21 | 21 | |
| ansix9t163k1 | q | | 0x0800000000000000000000000000000000000000C9 |
| sect163k1 | | | $x^{163} + x^7 + x^6 + x^3 + 1$ |
| EC2NGF163Koblitz | R2modq | | 0x000000000000000000000001410400000000000000 |
| wtls3 | b | | 0x01 |
| | c | | 0x000000000000000000000000000000000000000001 |
| | r | | 0x04000000000000000000020108A2E0CC0D99F8A5EF |
| | | | 5846006549323611672814741753598448348329118574063 |
| | R2modr | | 0x01719E20D16A34F5053B1368AE089C83FBAA63410E |
| sect163r1 | 21 | 21 | |
| ansix9t163r1 | q | | 0x0800000000000000000000000000000000000000C9 |
| | | | $x^{163} + x^7 + x^6 + x^3 + 1$ |
| | R2modq | | 0x000000000000000000000001410400000000000000 |
| | b | | 0x0713612DCDDCB40AAB946BDA29CA91F73AF958AFD9 |
| | c | | 0x05ED403ED58EB45B1CCECA0F4F61655549861BE052 |
| | r | | 0x03FFFFFFFFFFFFFFFFFFFF48AAB689C29CA710279B |

*Table continues on the next page...*

## Table 11-68. Special Values for common ECC $F_{2^m}$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | | | 5846006549323611672814738465098798981304420411291 |
| | R2modr | | 0x03F45AD7608E554A90299358F3719D4236333FE8B2 |
| wtls5 | 21 | 21 | |
| | q | | 0x080000000000000000000000000000000000000107<br>$x^{163} + x^8 + x^2 + x^1 + 1$ |
| | R2modq | | 0x000000000000000000000004005400000000000000 |
| | b<br>c | | 0xC9517D06D5240D3CFF38C74B20B6CD4D6F9DD4D9<br>0x0453E1E4B7291F5C2D53CE18483F007081E7EA26EC |
| | r | | 0x0400000000000000000001E60FC8821CC74DAEAFC1<br>5846006549323611672814741626226392056573832638401 |
| | R2modr | | 0x02704CFBABEA28A831BAD35BCAA440A89884D1FA9B |
| Oakley 4 | 24 | - | |
| | q | | 0x020000000000000000000000000002000000000000000001<br>$x^{185} + x^{69} + 1$ |
| | R2modq | | 0x000000000100000000000000000000000000000000004000 |
| | b<br>c | | 0x1EE9<br>0x000000000000300000018000C00C000000638030001C0009 |
| sect193r1 | 25 | 25 | |
| ansix9t193r1 | q | | 0x0200000000000000000000000000000000000000000000008001<br>$x^{193} + x^{15} + 1$ |
| | R2modq | | 0x00000000000000000000000000100000004000000000000000 |
| | b<br>c | | 0xFDFB49BFE6C3A89FACADAA7A1E5BBC7CC1C2E5D831478814<br>0x0167B35EB4313F263D0F7A3D5036F0A0A3C980D40E5A053ED2 |
| | r | | 0x01000000000000000000000000C7F34A778F443ACC920EBA49<br>6277101735386680763835789423269548053691575186051040197193 |
| | R2modr | | 0x009F0A6812CD5A0961578029E866525B193ED6F556637F68CD |
| sect193r2 | 25 | 25 | |
| ansix9t193r2 | q | | 0x0200000000000000000000000000000000000000000000008001<br>$x^{193} + x^{15} + 1$ |
| | R2modq | | 0x00000000000000000000000000100000004000000000000000 |
| | b<br>c | | 0xC9BB9E8927D4D64C377E2AB2856A5B16E3EFB7F61D4316AE<br>0x006989FE6BFE30EDDC3244269F3AAD18D66CF3DB3E3302FAA8 |
| | r | | 0x010000000000000000000000015AAB561B005413CCD4EE99D5<br>6277101735386680763835789423314955362437298222279840143829 |
| | R2modr | | 0x00B24356750478EA3C6D96955F00208DD023F898087B1BF123 |
| B-233 | 30 | 30 | |
| sect233r1 | q | | 0x020000000000000000000000000000000000000004000000000000000001 |

*Table continues on the next page...*

### Table 11-68. Special Values for common ECC $F_{2^m}$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| ansix9t233r1 | | | $x^{233} + x^{74} + 1$ |
| EC2NGF233Random | R2modq | | 0x000000000004000000000000000000000000000000000000400000000000 |
| wtls11 | b | | 0x66647EDE6C332C7F8C0923BB58213B333B20E9CE4281FE115F7D8F90AD |
| | c | | 0x0007D5EF4389DFF11ECDBA39C30970D3CE35CEBBA58473F64B4DC0F2686C |
| | r | | 0x0100000000000000000000000000000013E974E72F8A6922031D2603CFE0D7 |
| | | | 6901746346790563787434755862277025555839812737345013555379383634485463 |
| | R2modr | | 0x006AB044AA57CDD6D0CC9138B004578CD5EFE7E89545CDAA1BA1C26DD4D1 |
| K-233 | 30 | 29 | |
| sect233k1 | q | | 0x020000000000000000000000000000000000000004000000000000000001 |
| ansix9t233k1 | | | $x^{233} + x^{74} + 1$ |
| EC2NGF233Koblitz | R2modq | | 0x000000000004000000000000000000000000000000000000400000000000 |
| wtls10 | b | | 0x01 |
| | c | | 0x000000000000000000000000000000000000000000000000000000000001 |
| | r | | 0x8000000000000000000000000000069D5BB915BCD46EFB1AD5F173ABDF |
| | | | 3450873173395281893717377931138512760570940988862252126328087024741343 |
| | R2modr | | 0x59BEBED80293C813EEB5B58A0AF7E3EB91DB9A5B861710AC1009468BB6 |
| sect239k1 | 30 | 30 | |
| ansix9t239k1 | q | | 0x800000000000000000004000000000000000000000000000000000000001 |
| | | | $x^{239} + x^{158} + 1$ |
| | R2modq | | 0x000000000000100000000000000000008000000000000000000440000000 |
| | b | | 0x01 |
| | c | | 0x000000000000000000000000000000000000000000000000000000000001 |
| | r | | 0x2000000000000000000000000000005A79FEC67CB6E91F1C1DA800E478A5 |
| | | | 220855883097298041197912187592864814948216561321709848887480219215362213 |
| | R2modr | | 0x183E8C975E5EA68E203395FBEC1187B0F40DFFCA2CE64F17F77925590A73 |
| B-283 | 36 | 36 | |
| sect283r1 | q | | 0x08000000000000000000000000000000000000000000000000000000000000000000010A1 |
| ansix9t283r1 | | | $x^{283} + x^{12} + x^7 + x^5 + 1$ |
| EC2NGF283Random | R2modq | | 0x00000000000000000000000000000000000000000000000000000000000000040110040 0 |
| | b | | 0x027B680AC8B8596DA5A4AF8A19A0303FCA97FD7645309FA2A581485AF6263E313B79A2F5 |
| | c | | 0x03D8C93D3B0EA81D9294034D7EE3135D0AC5FC8D9CB0276F7211F880F0D81CA4C6E87B38 |

*Table continues on the next page...*

## Table 11-68. Special Values for common ECC $F_{2^m}$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | r | | 0x03FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEF90399660FC938A90165B042A7<br>CEFADB307<br>7770675568902916283677847627294075626569625924376904889109196526770044277787378692871 |
| | R2modr | | 0x0299ADD3FE013DB2E23755FAA9545A49222D8461643773D41D288BCA1EBB695<br>767D7CA78 |
| K-283<br>sect283k1<br>ansix9t283k1<br>EC2NGF283Koblitz | 36 | 36 | |
| | q | | 0x0800000000000000000000000000000000000000000000000000000000000000000<br>010A1<br>$x^{283} + x^{12} + x^7 + x^5 + 1$ |
| | R2modq | | 0x0000000000000000000000000000000000000000000000000000000000000004011<br>00400 |
| | b<br>c | | 0x01<br>0x0000000000000000000000000000000000000000000000000000000000000000000<br>00001 |
| | r | | 0x01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE9AE2ED07577265DFF7F94451E<br>061E163C61<br>3885337784451458141838923813647037813284811733793061324295874997529815829704422603873 |
| | R2modr | | 0x00868DB4F6424B4B297831F5EBA11AE2B1AC177F33C6C133859892757E41B7C<br>AE8927BDB |
| B-409<br>sect409r1<br>ansix9t409r1<br>EC2NGF409Random | 52 | 52 | |
| | q | | 0x0200000000000000000000000000000000000000000000000000000000000000000<br>0000000000000008000000000000000000001<br>$x^{409} + x^{87} + 1$ |
| | R2modq | | 0x0000000000000000000000000000000000000000000000000000000010000000000<br>0000000000000000000000000000000004000 |
| | b<br>c | | 0x21A5C2C8EE9FEB5C4B9A753B7B476B7FD6422EF1F3DD674761FA99D6AC27C<br>8A9A197B272822F6CD57A55AA4F50AE317B13545F<br>0x0149B8B7BEBD9B63653EF1CD8C6A5DD105A2AAAC36FE2EAE43CF28CE1CB7<br>C830C1ECDBFA413AB07FE35A57811AE4F88D30AC63FB |
| | r | | 0x010000000000000000000000000000000000000000000000000001E2AAD6A612F3<br>3307BE5FA47C3C9E052F838164CD37D9A21173<br>661055968790248598951915308032771039828404682964281219284648798304157774827374805208143723762179110965979867288366567526771 |
| | R2modr | | 0x0007C24B27E70A941D4738F415F186A66A9FF8783C798E99D4A152BF0CE0C0C<br>C273CDA3D70AA82C43A336EFBE1479034DB8EF936 |
| K-409<br>sect409k1<br>ansix9t409k1<br>EC2NGF409Koblitz | 52 | 51 | |
| | q | | 0x0200000000000000000000000000000000000000000000000000000000000000000<br>0000000000000008000000000000000000001<br>$x^{409} + x^{87} + 1$ |

*Table continues on the next page...*

### Table 11-68. Special Values for common ECC $F_{2^m}$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | R2modq | | 0x00000000000000000000000000000000000000000000000000000000100000000000<br>0000000000000000000000000000000004000 |
| | b<br>c | | 0x01<br>0x0000000000000000000000000000000000000000000000000000000000000000000<br>0000000000000000000000000000000000001 |
| | r | | 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE5F83B2D4E<br>A20400EC4557D5ED3E3E7CA5B4B5C83B8E01E5FCF<br>3305279843951242994759576540163855199142023414821406096423243950228807112892491910506732584577774580140963665906177313586 71 |
| | R2modr | | 0x50DC9B805D4BEBA0701EDA0D529DAD74A3ED9914801EFC3F5D0760180600F3<br>725B714A1C6E7B3C68A06DF3709E9354226F8D6C |
| B-571<br>sect571r1<br>ansix9t571r1<br>EC2NGF571Random | 72 | 72 | |
| | q | | 0x080000000000000000000000000000000000000000000000000000000000000000000<br>000000000000000000000000000000000000000000000000000000000000000000000<br>00000425<br>$x^{571} + x^{10} + x^5 + x^2 + 1$ |
| | R2modq | | 0x0000000000000000000000000000000000000000000000000000000000000000000<br>000000000000000000000000000000000000000000000000000000000000000000000<br>40104400 |
| | b<br>c | | 0x02F40E7E2221F295DE297117B7F3D62F5C6A97FFCB8CEFF1CD6BA8CE4A9A18<br>AD84FFABBD8EFA59332BE7AD6756A66E294AFD185A78FF12AA520E4DE739BAC<br>A0C7FFEFF7F2955727A<br>0x06395DB22AB594B1868CED952578B6539FABA69406D9B2986123A185C85832E<br>25FD5B63833D51442ABF1A9C05FF0ECBD88D7F77997F4DC9156AAF1CE0816468<br>6DDFF75116FBC9A7A |
| | r | | 0x03FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF<br>FFFFFFFFFFFE661CE18FF55987308059B186823851EC7DD9CA1161DE93D5174D<br>66E8382E9BB2FE84E47<br>386453752301725834469535189093198734429892732970643499865723525145151<br>914228956042453614399938941577308313388112192694448624687246281681307<br>0234528288303332411393191105285703 |
| | R2modr | | 0x00780C1005944C99C498CDB275BF7CCC389C0853C856C10F3786A7DCF3AA9A<br>E196A3FB16F1DE5AF21B1318667E55C15B9A8ABF1B469BD13D57BB95B60B677D<br>BCAA35B843B87069F9 |
| K-571<br>sect571k1<br>ansix9t571k1<br>EC2NGF571Koblitz | 72 | 72 | |
| | q | | 0x080000000000000000000000000000000000000000000000000000000000000000000<br>000000000000000000000000000000000000000000000000000000000000000000000<br>00000425<br>$x^{571} + x^{10} + x^5 + x^2 + 1$ |
| | R2modq | | 0x0000000000000000000000000000000000000000000000000000000000000000000<br>000000000000000000000000000000000000000000000000000000000000000000000<br>40104400 |
| | b<br>c | | 0x01 |

*Table continues on the next page...*

### Table 11-68. Special Values for common ECC $F_{2^m}$ domains when PKHA digit size is 32 bits (continued)

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| | | | 0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001 |
| | r | | 0x020000000000000000000000000000000000000000000000000000000000000000000000131850E1F19A63E4B391A8DB917F4138B630D84BE5D639381E91DEB45CFE778F637C1001<br>1932268761508629172347675945465993672149463664853217499328617625725759571144780212268133978522706711834706712800825351461273674974066617311929682421617092503555733685276673 |
| | R2modr | | 0x019433720D8C7057F7F3F824CCB3E09071584DD65C1437B2406C8210EEF4949565D35EE4FE01AAEE96D0DC137749D25AA49C07F63F829BF85960C535AA90F11DDEC4B62B18F2E26D |

For these Montgomery domains, some functions require *a24* instead of *A*. The PKHA requires that a24 be (A+2)/4.

### Table 11-69. Special Values for common ECM MOD (Montgomery curves) domains when PKHA digit size is 32 bits

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| Curve25519 | 32 | 32 | |
| | q | | 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFED<br>$2^{255}$-19 |
| | R2modq | | 0x00000000000000000000000000000000000000000000000000000000000005A4 |
| | A<br>A24 | | 0x076D06<br>486662<br>0x01DB42<br>121666 |
| | r | | 0x1000000000000000000000000000000014DEF9DEA2F79CD65812631A5CF5D3ED<br>7237005577332262213973186563042994240857116359379907606001950938285454250989 |
| | R2modr | | 0x0399411B7C309A3DCEEC73D217F5BE65D00E1BA768859347A40611E3449C0F01 |

**Table 11-70. Special Values for common ECT MOD (Edwards curves) domains when PKHA digit size is 32 bits**

| Name | L | N | |
|---|---|---|---|
| | var | | Value (hex, decimal, sums of powers) |
| edwards25519 | 32 | 32 | |
| | q | | 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFED<br>$2^{255}$-19 |
| | R2modq | | 0x00000000000000000000000000000000000000000000000000000000000005A4 |
| | r | | 0x1000000000000000000000000000000014DEF9DEA2F79CD65812631A5CF5D3ED<br>7237005577332262213973186563042994240857116359379907606001950938285454250989 |
| | R2modr | | 0x0399411B7C309A3DCEEC73D217F5BE65D00E1BA768859347A40611E3449C0F01 |

## 11.2 Kasumi f8 and f9 hardware accelerator(KFHA) functionality

Kasumi is a radio-interface, cryptographic algorithm set for LTE (two other sets are SNOW and ZUC). Kasumi 3G f8 is the encryption algorithm, and Kasumi 3G f9 is the authentication/integrity algorithm within this cryptographic set. The KFHA (Kasumi f8/9 hardware accelerator) implements both the f8 and f9 modes of the Kasumi cryptography algorithm. The KFHA CHA is controlled via the class 1 CHA registers. Because it is not possible to own two Class 1 CHAs simultaneously, it is not possible to "snoop" between two KFHAs or between a KFHA and an AESA.

### 11.2.1 KFHA use of the Mode Register

The KFHA uses the Mode Register as follows:

- The Encrypt/Decrypt field of the Mode Register is not used by KFHA, because there is no difference between encrypt and decrypt in Kasumi. However, this bit should be set to indicate whether encryption or decryption is being performed so that the Performance Counter registers properly count events (see the description of the PM registers in the register descriptions section of this document).
- The ICV check field of the Mode Register is used to enable/disable ICV checking.
- The Algorithm (ALG) field of the Mode Register must be set to "Kasumi" to activate the KFHA.

- The Additional Algorithm Information (AAI) field must be set to either "f8" or "f9". In addition, the bits for either GSM or EDGE (but not both) may be OR'ed into the values for f8 or f9 to enable GSM or EDGE.
- The Algorithm State (AS) field of the Mode Register is used to select between the operations described in this table.

**Table 11-71. Mode Register[AS] operation selections in KFHA**

| Mode | Indication |
|---|---|
| INIT | KFHA starts with an initialization process. This puts the KFHA into an initial state needed at the start of an operation |
| INIT/FINALIZE | KFHA starts with an initialization process, and at the end of the message process, any selected padding will occur |
| FINALIZE | KFHA starts with no initialization process, and at the end of the message process, any selected padding will occur |
| UPDATE | KFHA starts with no initialization process |

## 11.2.2 KFHA use of the Context Register

The KFHA uses the Context Register as follows:

- The 512-bit Context Register is used as the working registers for KFHA.
- The count, bearer, direction, ca, ce, FRESH, and ICV_in fields within the Context Register should be loaded by software before INIT or INIT/FINALIZE operations.
- The Context Register is continually updated during processing, so at the end of an operation, it reflects the current state of the KFHA. The bits are assigned as shown in these tables.

**Table 11-72. Context usage in Kasumi 3G modes**

| Mode | DWord number | Initial input definition | Context switching definition | Final result definition |
|---|---|---|---|---|
| Kasumi 3G f8 mode | 0 | count, bearer, direction, ca, ce | count, bearer, direction ca, ce | - |
| | 1 | - | - | - |
| | 2 | - | - | - |
| | 3 | - | - | - |
| | 4 | - | f8 C register | - |
| | 5 | - | f8 B register | - |
| | 6 | - | f8 A register | - |
| | 7 | - | core data out | - |
| Kasumi 3G f9 mode | 0 | count, bearer, direction, ca, ce | count, bearer, direction, ca, ce | count, bearer, direction, ca, ce |
| | 1 | FRESH, ICV_in | FRESH, ICV_in | FRESH, ICV_in |
| | 2 | - | f9 C register | ICV_out |

*Table continues on the next page...*

Table 11-72. Context usage in Kasumi 3G modes (continued)

| Mode | DWord number | Initial input definition | Context switching definition | Final result definition |
|---|---|---|---|---|
| | 3 | - | f9 A register | - |

Table 11-73. Format of context DWord 0 in Kasumi 3G f8 and f9 modes

| Bits 0-31 | Bits 32-36 | Bit 37 | Bits 38-47 | Bits 48-63 |
|---|---|---|---|---|
| count | bearer | direction | ca | ce |

Table 11-74. Format of context DWords 1 and 2 in Kasumi 3G f9 mode

| DWord number | Bits 0-31 | Bits 32-63 |
|---|---|---|
| 1 | FRESH | ICV_in |
| 2 (output only) | ICV_out | - |

## 11.2.3 KFHA use of the Key Register

The KFHA uses the Key Register as follows:

- The Key Register contains the 16-byte key used by the KFHA, which is placed in bits [127:0]. (The first key byte is in Key Register bits [127:120], the second key byte is in key [119:112], and so on).
- The Key Size Register must be programmed to a value of 16. Any other value produces a KFHA key size error.

## 11.2.4 KFHA use of the Data Size Register

The KFHA does not need to know the total size of the message being processed. It only needs to know how many bits will be processed out of the last 64-bit DWord of the message. Therefore, it uses only the 3 LSB of the Data Size Register (PDB 2:0]) and the NUMBITS field (bits 63:61 of the Data Size Register) during message processing.

## 11.2.5 KFHA error conditions

Errors that can occur while operating the KFHA CHA include the following:

**Table 11-75. KFHA error conditions and results**

| Error condition | Result |
|---|---|
| After a FINALIZE or INIT/FINALIZE operation completes and the Mode Register is cleared, if the mode is re-written to UPDATE or FINALIZE (with no DECO reset in between) | Mode error<br>**NOTE:** The assumption is that any "Final" packet requires an initialization or a reset for a following packet. |
| After an INIT or UPDATE operation (that is, operations that are not "Final") completes and the Mode Register is cleared, if the mode is re-written to INIT or INIT/FINALIZE (with no DECO reset in between) | Mode error<br>**NOTE:** The assumption is that any "Initial" packet would only follow a "Final" packet or a reset. |
| If both GSM and EDGE bits are set in the AAI field of the mode register while KFHA is operating | Mode error |
| When operating KFHA, neither F8 or F9 are selected in the mode register AAI field | Mode error |
| If the KFHA is operated with a key size of any value other than 16 bytes | Key size error |
| If any input data is received by KFHA that is not type "Message" | Data sequence error |
| If KFHA is operating in F9 mode with ICV checking enabled in the Mode Register | ICV mismatch produces an ICV check error |

## 11.3 Data encryption standard accelerator (DES) functionality

DES performs encryption and decryption on 64-bit values using the algorithm found in FIPS46-3. The DES module in SEC supports both single- and triple-DES functionality and ECB, CBC, CFB, and OFB modes as well as key parity checking in compliance with the DES specification. DES is controlled from the class 1 CHA registers.

### 11.3.1 DESA use of the Mode Register

The DESA uses the Mode Register as follows:

- The encryption field (ENC) controls whether DESA is encrypting or decrypting data.
- The Algorithm State (AS) field is not used to affect DESA functionality and should be set to zero at all times.
- The Additional Algorithm Information field (AAI) specifies the mode DESA runs. The supported modes are electronic code book (ECB), cipher block chaining (CBC), cipher feedback (CFB-8), and output feedback (OFB), described as follows:
  - ECB (0x20h) mode is a confidentiality mode that features, for a given key, the assignment of a fixed ciphertext block to each plaintext block (analogous to the assignment of code words in a codebook).

  - CBC (0x10h) mode is a confidentiality mode whose encryption process features the combining ("chaining") of the plaintext blocks with the previous ciphertext blocks. CBC mode requires an IV to combine with the first plaintext block. The IV does not need to be secret, but it must be unpredictable.
  - CFB (0x30h) mode is a confidentiality mode that features the feedback of successive ciphertext segments into the input blocks of the forward cipher to generate output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The CFB mode requires an IV as the initial input-block.
  - OFB (0x40h) mode is a confidentiality mode that features the iteration of the forward cipher on an IV to generate a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
- Key parity checking for DESA that checks for odd parity within each byte of the key is enabled with a value of (0x80h) in the AAI field.
- The algorithm field (ALG) must be programmed to DES (0x20h) or 3DES (0x21h).

### 11.3.2 DESA use of the Key Register

The DESA uses the Key Register as follows:

- The Key Register contains the 8-, 16-, or 24-byte key that is used during permutation in all DES modes.
- The DES specification defines the key as having odd parity in each byte.
- Key parity can be verified using the correct mode setting.

### 11.3.3 DESA use of the Key Size Register

DESA uses the Key Size Register as follows:

- Key size can be either 8, 16, or 24 bytes.
- A key size of 8 is valid only in single-DES mode.
- Values of 16 and 24 bytes can be used only in triple-DES mode.
- An illegal key size error is generated when in single-DES mode with a key size other than 8 or when in triple-DES mode with a key size other than 16 or 24.

### 11.3.4 DESA use of the Data Size Register

The DESA uses the Data Size Register as follows:

- The Data Size Register is written with the number of bytes of data to be processed.
- All DES modes except OFB expect to process data that is a multiple of 8 bytes and generates an error if the data size written is not an 8-byte multiple.
- This register must be written to start data processing.
- Because writing to the Data Size Register causes the written value to be added to the previous value in the register, the register may be written multiple times while data is being processed in order to increase the amount of input data that will be processed.

### 11.3.5 DESA Context Register

The DESA uses the Context Register as follows:

- For CBC, OFB, and CFB modes, the initialization vector is written to and read from the DESA Context Register.
- The value of this register changes as a result of the encryption process and reflects the context of DESA.
- DESA uses the first eight bytes of the Context Register to hold the beginning and final IV value for the CBC, OFB, and CFB modes. The bits are assigned as follows: Context DWord0: IV = desa_context[63:0]

### 11.3.6 Save and store operations in DESA context data

DESA is able to process data in chunks by saving the intermediate IV from the Context Register after each chunk of data and restoring the IV and key to the correct registers before processing any subsequent chunks of data.

## 11.4 Cyclic-redundancy check accelerator (CRCA) functionality

CRCA performs 1- to 32-bit cyclic redundancy code computation. Cyclic redundancy checks are a common algorithm for providing additional integrity check bits that are included with data in danger of corruption. A typical usage is to include a CRC for each packet transmitted over a network and then recompute this CRC at the destination to verify that the received packet is uncorrupted. CRC is controlled via the class 2 CHA registers

## 11.4.1 CRCA modes of operation

The CRCA module supports the following primary modes of operation:

- IEEE 802 CRC32 protocol mode
- iSCSI (IETF 3385) CRC32c protocol mode
- Dynamic polynomial custom mode

The protocol-specific modes automatically use the correct polynomial. The custom mode takes the desired polynomial from the Key Register. All modes support either raw (DOS and DOC equal one), default processing, or any combination of the use of DIS, DOS, DOC, and IVZ control bits. Default processing involves bit-swapping, byte-swapping, complementing the CRC result, and setting initial value to FFFFFFFFh for specifications compliance. Raw mode (with DOS and DOC set) does not modify the CRC result in any way and is commonly used for partial CRC calculations. ICV checking can be performed in all modes.

## 11.4.2 CRCA use of the Mode Register

The CRCA uses the Mode Register as follows:

- The Mode Register is used to program the function of the CRCA.
- The ICV field selects whether a comparison between the computed CRC and the provided CRC should be performed. When enabled, ICV check compares a CRC calculated across the message against the received ICV. The comparison is performed after the bit manipulations controlled by the DIS, DOS, DOC, IVZ bits are performed.
- The Algorithm State (AS) field controls two features in the CRC module, namely the ability to load context and inhibit mangling output data. The LSB of the AS field selects whether to load the context value that was written into the class 2 Context Register. The MSB of the AS field is used to inhibit both the output swapping and the 1's complement of data when the current mode of operation does not result in a final computation of the CRC. Programming the initialize or update modes overrides the functionality of the DOS and DOC bits. This table is a synopsis of the 2-bit AS field and its functionality.

**Table 11-76. 2-bit AS functionality and synopsis**

| Value | Phase | Actions |
|---|---|---|
| 00 | Update | • Loads context<br>• Does not mangle output data |

*Table continues on the next page...*

**Table 11-76. 2-bit AS functionality and synopsis (continued)**

| Value | Phase | Actions |
|---|---|---|
| 01 | Initialize | • Does not load context<br>• Does not mangle output data |
| 10 | Finalize | • Loads context<br>• Mangles output data |
| 11 | Init/Finalize | • Does not load context<br>• Mangles output data |

- The Algorithm field (ALG) must be set to CRC.
- The Additional Algorithm Information (AAI) field controls the modes and functionality of the CRC module. The lower 4 bits select one of the three modes that the CRC operates in while the upper 5 bits allow individual control over how the CRC engine manipulates its input and output data. The lower 4-bit definitions for the mode values are described in this table.

**Table 11-77. Lower 4 bits AAI functionality**

| Value | Mode | Actions |
|---|---|---|
| 0001 | IEEE 802 mode | The CRC32 algorithm is performed using the polynomial 04C11DB7h. |
| 0010 | iSCSI mode (also called IETF 3385) | The CRC32c algorithm is performed using the polynomial 1EDC6F41h. |
| 0100 | Dynamic custom mode | • The CRC remainder is computed using the polynomial programmed into the Key Register.<br>• The polynomial can be 1-32 bits.<br>• The polynomial must be left justified. |

- In compliance with the IEEE 802 and iSCSI CRC implementations, the CRCA result is bit-swapped, byte-swapped, and complemented before it is output. The upper 5 bits of the AAI field allow the user to individually control the input bit-swapping (DIS), the output bit-swapping (DOS), complementing of output data (DOC), and initial value zero (IVZ).

**Table 11-78. Upper 5 bits AAI functionality**

| Value | Name | Description | Encodings |
|---|---|---|---|
| 00001 | DIS | Don't Input Swap | • 0: Input data is bit-swapped<br>• 1: Turns off swapping of the input data bits |
| 00010 | DOS | Don't Output Swap | • 0: Output data is bit and byte-swapped<br>• 1: Turns off bit-/byte-swapping of the output data |
| 00100 | DOC | Don't Output Complement | • 0: Output data is complemented<br>• 1: Turns off complementing the CRC output data |
| 01000 | IVZ | Initial Value Zero | • 0: Initial CRC value is FFFFFFFFh<br>• 1: Initial CRC value is 00000000h |

### 11.4.3 CRCA Key Register

The CRCA uses the Key Register as follows:

- The Class 2 Key Register stores the polynomial for custom mode.
- Any size polynomial may be used, up to 32 bits.
- The key must be left-justified in the Key Register.

### 11.4.4 CRCA Key Size Register

The CRCA uses the Key Size Register as follows:

- For IEEE 802 CRC32 protocol mode and iSCSI (IETF 3385) CRC32c protocol mode, it is not necessary to write to this register, because the polynomial size is clearly fixed by the algorithm.
- For dynamic custom polynomial mode, write a 4-byte value into the Key Size register to specify the size of the polynomial.

### 11.4.5 CRCA Data Size Register

The CRCA uses the Data Size Register as follows:

- The Data Size Register is written with the number of bytes of data to be processed.
- This register must be written to start data processing.

### 11.4.6 CRCA Context Register

The CRCA uses the Context Register as follows:

- The 32-bit Context Register is used to load partial CRCs. This register can be written with an intermediate CRC result prior to processing any data. Once processing is complete, the CRC result is available from this register. The reset state of this register is all ones, because this allows the CRC32 algorithm to detect bit errors in the leading zeros of a message.
- CRC uses the first word of the Context Register to load a CRC value for the engine to start from a previous, partial calculation. This register also holds the intermediate or final CRC when calculations are done. The bits are assigned as follows:
- Context DWord0: CRC register = {00000000h, crca_context[0:31]}

### 11.4.7 Save and restore operations in CRCA context data

To run CRC for multiple, related groups of data, it is necessary to program CRC in the correct initial mode (either initialize or update) and then to save the intermediate CRC value from the Context Register to be used for all subsequent data. When it is necessary to continue from the previous point, the intermediate CRC value must be loaded into the Context Register before the Data Size Register is written.

CRC must be programmed in either update or finalize modes (AS field) to continue from the previous partial calculation. For CRC to compute its final value, the module would need to be programmed in the final mode (AS field). This causes CRC to both swap and complement the output and provide the calculated CRC in the Context Register. If a custom polynomial was used for the above calculations, it would need to be loaded into the Key Register each time calculations were continued.

## 11.5 Random-number generator (RNG) functionality

The RNG generates cryptographically-strong, random data. SEC's RNG utilizes a true random-number generator (TRNG) as well as a deterministic random-bit generator (DRBG) to achieve both true randomness and cryptographic strength.

The random numbers generated by the RNG are intended for direct use as secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms. Note that before data can be obtained from the RNG, it must be instantiated in a particular mode by executing the appropriate descriptor. Also, a descriptor must be executed to load the JDKEK, TDKEK and TDSK registers with data from the RNG.

### 11.5.1 RNG features summary

The RNG module includes these distinctive features:

- Complete implementation of DRBG_Hash (SP800-90A) using SHA-256.
- Support for two state handles.
- Built-in entropy source conforming to SP800-90B and BSI AIS/31.
- Integrated entropy source for instantiating and re-seeding the DRBG.
- The RNG may be accessed through a register interface for test purposes.
- IP global interface. The RNG system clock and resets are controlled through the IP global interface.

- Interface to supply SEC with random data as required for descriptors, and to initialize key encryption key and Trusted Descriptor Signing Key Registers.
- Random-number stream obtainable using job descriptor.
- Random numbers automatically supplied as padding when needed by protocols.
- External interface to furnish random data to other modules such as SecMon.

## 11.5.2 RNG functional description

While the RNG consists of several, functional sub-modules, its overall functionality can be easily described from the top level in terms of a few functional operations. These operations are seed generation and random number generation. Each of these operations require coordination of the RNG's true random-number generator (TRNG) and deterministic random-bit generator (DRBG). TRNG creates real entropy (seed generation) and DRBG generates cryptographically strong data using this entropy (random-number generation).

### 11.5.2.1 RNG state handles

The RNG in SEC implements two state handles. Each state handle is:

- A completely independent virtual RNG.
- Instantiated independently in deterministic or nondeterministic mode, and with or without prediction resistance
- Seeded (or reseeded) independently with independent entropy

Thereafter, each state handle maintains an independent context for the RNG's deterministic random bit generator (DRBG).

Note that the JDKEK, TDKEK, TDSK and ZMK (if the ZMK is set for hardware programming) and the random data to seed the AES Differential Power Analysis protection (if implemented in the AESA) are initialized by data drawn from State Handle 0, and that any random padding required by SEC's built-in protocols is also drawn from State Handle 0. Because this data may be confidential, there are special security features to ensure that State Handle 0 is not inadvertently or maliciously instantiated in deterministic mode when it should have been instantiated in nondeterministic mode. See the discussion of the RNGSH0 and RANDDPAR fields in the Security Configuration Register.

### 11.5.2.2 RNG NIST certification

SEC's RNG is designed to be NIST-certifiable. One of the requirements of that certification is the ability to test the RNG prior to normal operation. This requires instantiating the RNG in test (deterministic) mode rather than normal (nondeterministic) operational mode, and then having software run various tests on the RNG. To allow an opportunity for this testing, SEC does not automatically instantiate the RNG in operational mode or automatically load the JDKEK, TDKEK and TDSK registers, and does not automatically respond to a request for random data from the SecMon. After the tests have completed, or if the tests are going to be skipped, the RNG must be instantiated in operational mode and the JDKEK, TDKEK and TDSK registers must be loaded. These steps are accomplished by executing descriptors as described in this table. The execution of these descriptors must be initiated by software, typically via the job ring interface.

**Table 11-79. Examples of Descriptors to initialize, instantiate and uninstantiate the RNG and to initialize the JDKEK, TDKEK and TDSK**

| Descriptor | Value | Execution |
|---|---|---|
| Descriptor to instantiate RNG State Handle 0 in deterministic (test) mode<br>**NOTE:** This descriptor would be executed prior to running tests on the RNG. | B080 0004h | HEADER command indicating a descriptor with a length of four 32-bit words |
| | 1281 0004h | LOAD Command with 4 bytes of immediate data; destination is Class 1 Key Size register |
| | 0000 0000h | 4 bytes of immediate data (entropy input is null) |
| | 8250 0005h | OPERATION command, Class 1, RNG, Instantiate, Test Mode |
| | | |
| Descriptor to uninstantiate RNG State Handle 0<br>**NOTE:** This descriptor would be executed after running tests on the RNG. | B080 0002h | HEADER command, indicating a descriptor with a length of two 32-bit words |
| | 8250 000Dh | OPERATION command, Class 1, RNG, Uninstantiate, Test Mode |
| | | |
| Descriptor to instantiate RNG State Handle 0 in normal mode and load the JDKEK, TDKEK, and TDSK registers<br>**NOTE:** This descriptor would be executed to start normal operation of the RNG and also initialize JDKEK, TDKEK and TDSK. | B080 0006h | HEADER command, indicating a descriptor with a length of six 32-bit words |
| | 8250 0004h | OPERATION command, Class 1, RNG, Instantiate, Non-Test Mode |
| | A200 0001h | JUMP command, wait until Class 1 (RNG) done then local jump to next command |
| | 1088 0004h | LOAD command, 4 bytes of Immediate data, destination Clear Written Register |
| | 0000 0001h | 4 bytes of immediate data (clear the Class 1 Mode Register. This resets the done interrrupt and returns the RNG to idle.) |
| | 8250 1000h | OPERATION command, Class 1, RNG, Secure Key, Generate |
| The Generate command for secure keys allows for an optional "additional_input" of up to 256 bits that would be loaded into the Class 1 Context register prior to executing the OPERATION Generate command. | | |
| | | |

*Table continues on the next page...*

Table 11-79. Examples of Descriptors to initialize, instantiate and uninstantiate the RNG and to initialize the JDKEK, TDKEK and TDSK (continued)

| Descriptor | Value | Execution |
|---|---|---|
| Descriptor to instantiate RNG State Handle 0 in nondeterministic (normal) mode<br>**NOTE:** This descriptor would be executed to start normal operation of the RNG, but without initializing JDKEK, TDKEK and TDSK. | B080 0002h | HEADER command, indicating a descriptor with a length of two 32-bit words |
| | 8250 0004h | OPERATION command, Class 1, RNG, Instantiate, Non-Test Mode |
| The Instantiate command allows for an optional "personalization_string" of up to 256 bits that would be loaded into the Class 1 Context register prior to executing the OPERATION Instantiate command. | | |

## 11.5.3 RNG operations

RNG operations are performed by appropriately setting the Algorithm State (AS) field of the OPERATION command.

Table 11-80. RNG Operations

| Value of AS | Name | Function |
|---|---|---|
| 00 | State-handle generate operation | Causes the RNG to generate random data from the selected state handle and push that data to the output FIFO. The amount of data generated is based on the value in the Class 1 Data Size register. The descriptor can also provide 256 bits of additional input via the Class 2 Key Register, which is used as additional entropy when generating the requested data. The TST bit value must match the deterministic/nondeterministic mode of the selected state handle, else a test error is generated.[1] A test error is also generated if a Generate command is issued to a state handle that is not instantiated. |
| 01 | State-handle instantiation operation | Causes the RNG to set up the initial context for the specified state handle. The state handle remains instantiated in the specified mode (deterministic or nondeterministic) until it is uninstantiated or SEC is reset. A test error is generated if an attempt is made to instantiate a state handle that is already instantiated.<br>• TST bit = 0. Nondeterministic instantiation. When instantiating a state handle in nondeterministic (normal) mode, the state handle is seeded with 512 bits of high-grade random entropy from the TRNG and an optional 256-bit personalization string supplied by the descriptor via the Class 1 Context Register.<br>• TST bit = 1. Deterministic instantiation. When instantiating a state handle in deterministic (test) mode, the state handle is seeded with 256 bits of user-specified entropy supplied via the Class 1 Key register and an additional 256 bits of nonce supplied via the Class 2 Key register. Seeding the state handle with known entropy and nonce values allows for deterministic testing. Note that once the RNGSH0 bit in the Security Configuration register has been set to 1, State Handle 0 can no longer be instantiated in deterministic mode. State Handle 0 |

*Table continues on the next page...*

Table 11-80. RNG Operations (continued)

| Value of AS | Name | Function |
|---|---|---|
| | | produces the random numbers used for nonces and padding within the built-in protocols, so this special protection can be used to prevent accidentally or maliciously substituting a test instantiation in place of a nondeterministic instantiation. |
| 10 | State-handle reseed operation | Causes the RNG to reseed an already instantiated state handle; that is, the current state associated with the selected state handle is replaced with new state information. A test error is generated if an attempt is made to reseed a state handle that is not instantiated.<br>• For a state handle in nondeterministic mode, the DRNG is seeded with 512 bits of entropy from the TRNG and an optional 256-bit additional input from the descriptor via the Class 1 Context Register.<br>• For a state handle in deterministic mode, 256 bits of user-specified entropy is taken from the Class 1 Key Register. Nonce is not used for reseeding. |
| 11 | State-handle uninstantiate operation | Causes the RNG to uninstantiate the specified state handle, which prevents the state handle from being used to generate data. The state handle can later be instantiated again. A test error is generated if an attempt is made to uninstantiate a state handle that is not instantiated. |

1. There is one exception to this rule. A test error is not generated if State Handle 0 is in Test mode but a generate operation requests nondeterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.

## 11.5.4 RNG use of the Key Registers

RNG uses the key registers as follows:

- RNG uses the Class 1 Key Register only when instantiating or reseeding a state handle in deterministic (test) mode.
- RNG uses the Class 2 Key Register only when instantiating a state handle in deterministic (test) mode. In these cases, the descriptor has the TST bit set during the OPERATION command and has loaded known values into the following registers:
  - 256-bit entropy input in the Class 1 Key Register (for instantiate and reseed operations)
  - 256-bit nonce in the Class 2 Key register (only for instantiate operations)
- When instantiating or reseeding a state handle in nondeterministic mode, the key registers are ignored and entropy is instead obtained from the TRNG.

## 11.5.5 RNG use of the Context Register

The Class 1 Context Register is used to supply an optional 256-bit personalization string when instantiating a state handle, or to supply an optional 256 bits of additional input when reseeding a state handle or generating random data.

### 11.5.6 RNG use of the Data Size Register

The RNG uses the Data Size Register as follows:

- When an RNG generate command is executed, the value in the Data Size Register specifies the number of bytes of random data that should be generated and pushed onto the Output FIFO.
- When an RNG instantiate command is executed, the value in the Data Size Register specifies a reseed interval, measured in number of ***generate*** requests.
- The RNG uses a default reseed value of 10,000,000 requests. This means that 10,000,000 ***generate*** requests are processed before an automatic reseed operation occurs. For a system with the clock speed between 133MHz - 400MHz, the reseed happens between 3-20 seconds if RNG operations are being processed at the maximum rate.
- The Data Size Register holds 32 bits so the user can specify a larger or smaller value. If the user does not specify a reseed interval, the default value is used.

## 11.6 SNOW 3G f8 accelerator functionality

SNOW is a radio interface cryptographic algorithm set for LTE (two other sets are Kasumi and ZUC). The f8 mode confidentiality algorithm is defined as a word-oriented stream cipher that generates a sequence of 32-bit words under the control of a 128-bit key and a 128-bit initialization value. It can be used to encrypt or decrypt blocks of data between 1 and 20000 bits in length. Some of the features of the SNOW f8 accelerator include the following:

- Message encryption and decryption in f8 (UEA2) mode
- Throughput of up to 4 bytes per cycle
- Support for multiple session message processing through context switching
- Support for descriptor sharing
- Total message size of up to $2^{32}$ bits (processed in chunks of no more than $2^{17}$-1 bytes per session)
- Support for any number of bits in the last byte of the message
- Automatic zeroization of the invalid bits in the last incomplete byte of the message

### 11.6.1 Differences between SNOW 3G f8 and SNOW 3G f9

Some of the key differences between SNOW 3G f8 and f9 are as follows:

- SNOW 3G f8 is the encryption algorithm, and SNOW 3G f9 is the authentication algorithm within this cryptographic set.
- The SNOWf8 hardware accelerator implements the f8 encryption mode of operation of the SNOW algorithm, whereas the SNOWf9 hardware accelerator implements the f9 integrity authentication mode of the SNOW algorithm.
- The SNOW f8 CHA is programmed using Class 1 CCB registers, whereas the SNOW f9 CHA is programmed using Class 2 CCB registers. Note that it is possible to encrypt or decrypt data using SNOW f8 and also hash the same data using SNOW f9 authentication via "snooping"; that is, passing the same data simultaneously to both CHAs ("in snooping"), or passing the output of one CHA directly to the input of the other CHA ("out snooping"). However, in those versions of SEC that implement more than one DECO but only one SNOW f8 CHA and one SNOW f9 CHA, the descriptor must select the SNOW f9 CHA first. Selecting the SNOW f8 CHA first and then selecting the SNOW f9 CHA within the same descriptor results in an error indication.

## 11.6.2 SNOW 3G f8 use of the Mode Register

The SNOW 3G f8 uses the Mode Register as follows:

- The SNOW 3G f8 accelerator is enabled by setting the Algorithm (ALG) field of the Class 1 Mode Register to 60h.
- The f8 mode is enabled by setting the Additional Algorithm Information (AAI) field to C0h.
- The Algorithm State (AS) field should be set to "Initialize" state when a new message is to be processed. The SNOW 3G f8 accelerator initializes the core engine (keystream generator) based on the key and initialization parameters COUNT-C, BEARER and DIRECTION in a 32-step initialization process. This is a necessary step before keystream generation can begin. It is possible to perform this initialization in advance without the need to provide any input data by writing 0 to the Data Size register. The AS field should be reset (or set to "Update" state) after context switch, assuming that Key/Context Registers are restored, when continuing message processing. In this case, the state of the keystream generator necessary for continuation of message processing is in the Key/Context Registers and initialization is not needed.
- Other fields in the Mode Register have no effect on f8 mode.
- If the AAI field is set to a value that does not correspond to f8 mode, an illegal-mode error is generated. The Mode, Key Size and Data Size Registers can be written in any order. The operation will begin after all of these have been written.

### 11.6.3 SNOW 3G f8 use of the Context Register

The SNOW 3G f8 uses the Context Register as follows:

- SNOW 3G f8 uses the Class 1 Key and Context registers.
- The usage of the Key and Context registers in the f8 mode is described in this table.

**Table 11-81. Key/Context Register usage in SNOW 3G f8 mode**

| Register | DWord number | Initialize input definition | Update input definition[1] |
|---|---|---|---|
| Key Register | 0 | Key[0:63] | s0, s1 |
| | 1 | Key[64:127] | s2, s3 |
| | 2 | - | s4, s5 |
| | 3 | - | s6, s7 |
| Context Register | 0 | Count-C \|\| Bearer \|\| Direction \|\| 0 | s8, s9 |
| | 1 | - | - |
| | 2 | - | - |
| | 3 | - | r1 |
| | 4 | - | r2, r3 |
| | 5 | - | s10, s11 |
| | 6 | - | s12, s13 |
| | 7 | - | s14, s15 |

[1] The symbols in this column represent values written back by SNOW 3G f8. These values comprise the state of the keystream generator that must be restored after context switch for the message processing to continue.

- In the f8 mode, the Context Register is treated as an extension of the Key Register; that is, it is automatically encrypted when saved and decrypted when restored. The IV value must be written to the Context Register when starting a new Job in the f8 mode. This value consists of SNOW 3G f8 initialization parameters in the order shown in this table.

**Table 11-82. IV that must be written to Class 1 Context Register in SNOW 3G f8 mode**

| 0-31 | 32-36 | 37 | 38-63 |
|---|---|---|---|
| Count-C | Bearer | Direction | 0 |

### 11.6.4 SNOW 3G f8 use of the Data Size Register

The SNOW 3G f8 uses the Data Size Register as follows:

- SNOW 3G f8 uses the 17 lsbs of the Class 1 Data Size Register to indicate the number of bytes of input data and uses bits 63-61 to indicate the number of valid bits in the last byte.
- SNOW 3G f8 internally decrements this value as it processes the message. It continue to process data until the value in the Data Size register reaches zero. If 0 is written to the Data Size register and the AS field of the Mode Register is set to "Initialize", SNOW 3G f8 keystream generator is initialized and Key and Context Registers contain this initialized state.

### 11.6.5 SNOW 3G f8 use of the Key Register

The SNOW 3G f8 uses the Key Register as follows:

- A 128-bit key must be written to the Class 1 Key Register with offset of 0 if the AS field of the Mode Register is set to "Initialize".
- The key is necessary for the initialization of the keystream generator but it is not needed when the AS field of the Mode Register is set to "Update"; that is, when a message processing is continued after context switch.
- The Key Register is used to implement internal state of the keystream generator as depicted in Table 11-81.

### 11.6.6 SNOW 3G f8 use of the Key Size Register

Writing to this register is not required by SNOW 3G f8, because the SNOW 3G f8 key is always 16 bytes long. Writing a value of 16 to this register is allowed, but writing a value other than 16 causes a key-size error to be generated.

## 11.7 SNOW 3G f9 accelerator functionality

SNOW 3G f9 is a keyed word-oriented stream integrity/authentication algorithm that generates a 32-bit message digest under the control of a 128-bit key and a 128-bit initialization value. The message may be between 1 and 20000 bits in length. The algorithm is based on the same stream cipher (SNOW 3G f9) as is used by the confidentiality algorithm f8. See Differences between SNOW 3G f8 and SNOW 3G f9 for more on the differences between the SNOW f8 and the SNOW f9 CHAs. Some of the features of the SNOW f9 CHA are as follows:

- Message authentication in f9 (UIA2) mode
- Automatic comparison of the received and computed MAC values (ICV check)

- Throughput of up to 2 bytes per cycle
- Support for multiple session message processing through context switching
- Support for descriptor sharing
- Total message size of up to $2^{32}$ bits (processed in chunks of no more than $2^{17}$-1 bytes per session)
- Supports any number of bits in the last byte of the message
- Automatic zeroization of the invalid bits in the last incomplete byte of the message

## 11.7.1 SNOW 3G f9 use of the Mode Register

The SNOW 3G f9 uses the Mode Register as follows:

- The SNOW 3G f9 accelerator is enabled by setting the Algorithm (ALG) field of the class 2 Mode Register to A0h.
- The f9 mode is enabled by setting the Additional Algorithm Information (AAI) field to C8.
- The Algorithm State (AS) field must be set to "Initialize" state when the first session of message processing is to be performed. This assumes that message processing is split into multiple sessions; that is, that the first one is not also the final session. The SNOW 3G f9 accelerator initializes the core engine (keystream generator) based on the key and an IV built from initialization parameters COUNT-I, DIRECTION, BEARER and FRESH in a 32-step initialization process. This is a necessary step before keystream generation can begin. It is possible to perform this initialization in advance without the need to provide any input data by writing 0 to the Data Size register.
- If the AS mode field is set to "Initialize/Finalize" (11b), but there is no message to be processed (0 written to the Data Size Register), the computed MAC will be identical to the keystream word Z5 as defined in the SNOW specification. If the CICV mode bit is also set, the ICV/MAC expected on the input-data FIFO is checked against Z5.
- If the data size is 0, and CICV is 1, AS set to "Update" (00b) means that Check ICV job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the input-data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session.
- The AS field must be set to "Finalize" state when the last session of message processing is to be performed. This enables computation of the MAC.
- The AS field must be set to "Initialize/Finalize" state when the whole message is processed in one session.
- The ICV bit of the Mode Register must be set for the f9 mode to compare computed MAC/ICV with the received ICV. The received ICV must be provided through the input data FIFO following the message data at which time the FIFO data type must

be set to ICV. If this bit is reset, the f9 mode does not expect ICV to be put on Input Data FIFO.

- The illegal-mode error is generated if ICV bit is set, but the AS field is not set to either "Initialize/Finalize" or "Finalize" state, except for CICV-only jobs; that is, the data size is non-zero and valid (there are no more writes to the Data Size Register).
- If the AAI field is set to a value that does not correspond to f9 mode, the illegal-mode error is generated. The Mode, Key Size and Data Size Registers can be written in any order. The operation will begin after all of these have been written.
- When SEC descriptor sharing mode is used with SNOW 3G f9, clear mode, followed by clear-done interrupt command, must be issued between SNOW 3G f9 jobs.

## 11.7.2 SNOW 3G f9 use of the Context Register

The SNOW 3G f9 uses the Context Register as follows:

- This table shows context usage in the f9 mode that is relevant for its programming.

**Table 11-83. Context usage in SNOW 3G f9 mode**

| Register | DWord number | Initialization input definition | Update state (for context switching) | Finalize output definition |
|---|---|---|---|---|
| Key Register | 0 | Key[0:63] | - | - |
| | 1 | Key[64:127] | - | - |
| | 2 | - | - | - |
| | 3 | - | - | - |
| | 4 | - | IV | - |
| | 5 | - | z1, z2 | - |
| | 6 | - | z3, z4 | - |
| | 7 | - | z5, bit length | - |
| Context Register | 0 | Count-C ‖ 0 ‖ Direction ‖ 0 | - | {MAC,32'h0} |
| | 1 | {FRESH, 32'h0} (3G) {Bearer, 59'h0} (LTE) | - | - |

- For 3G, the IV value is built as shown in this table.

**Table 11-84. IV in Class 2 Context for 3G in SNOW 3G f9 mode**

| 0-31 | 32-36 | 37 | 38-63 | 64-95 |
|---|---|---|---|---|
| Count-C | 0 | Direction | 0 | FRESH |

- LTE systems do not include a FRESH value in the f9 IV value. It is instead built as shown in this table.

Table 11-85. IV in Class 2 Context for LTE in SNOW 3G f9 mode

| 0-31 | 32-36 | 37 | 38-63 | 64-68 | 69-95 |
|---|---|---|---|---|---|
| Count-C | 0 | Direction | 0 | Bearer | 0 |

- At the end of processing, SNOW 3G f9 overwrites IV in the context word 0 with the MAC/ICV. Because the MAC is a 32-bit value, it is written left-justified and the remaining bits are cleared.
- Values z1-z5 are the keystream words computed during initialization of the f9 mode by the keystream generator. After initialization stage is complete, keystream generator is not active any more in the f9 mode. The processing is based on the Galois Field (GF) multiplier implemented as part of the f9 mode logic. The bit length is a value copied from the data size register to be used to compute the final MAC. In case of multi-session message processing, this value represents the total message length as each session's data size is accumulated.
- To read only the final MAC value, the "finalize" option must be present in the AS mode setting. When saving context, the starting address must be the address of the first double word of the Key Registers.

## 11.7.3 SNOW 3G f9 use of the Data Size Register

The SNOW 3G f9 uses the Data Size Register as follows:

- SNOW 3G f9 uses the 17 lsbs of the Class 2 Data Size register to indicate the number of bytes of input data, and the NUMBITS field to indicate the number of valid bits in the last byte.
- SNOW 3G f9 internally decrements this value as it processes the message. It continues to process data until the value in the Data Size register reaches zero. If 0 is written to the Data Size register and the AS field of the Mode Register is set to "Initialize", SNOW 3G f9 keystream generator is initialized and the context contains this initialized state.
- In the f9 mode, the data size must be divisible by 64 except when the AS field of the Mode Register is set to "Finalize" or "Initialize/Finalize". In other words, the message can be split for multi-session processing only on a 64-bit boundary. If this rule is violated, the illegal data size error is generated.

### 11.7.4 SNOW 3G f9 use of the Key Register

A 128-bit key must be written to the Class 2 Key Register with offset of 0 if the AS field of the Mode Register is set to "Initialize" or "Initialize/Finalize". The key is necessary for the initialization of the keystream generator.

### 11.7.5 SNOW 3G f9 use of the Key Size Register

The SNOW 3G f9 uses the Key Size Register as follows:

- Writing to this register is not required by SNOW 3G f9, because the SNOW 3G f9 key is always 16 bytes long.
- Writing a value of 16 to this register is allowed, but writing a value other than 16 causes a key-size error to be generated.

### 11.7.6 SNOW 3G f9 use of ICV check

The SNOW 3G f9 uses ICV check as follows:

- The f9 mode can automatically compare received ICV with the computed ICV at the end of processing if the ICV bit of the Mode Register is set and the AS field is set to "Finalize" or "Initialize/Finalize".
- The received ICV must be supplied after message data through the Input Data FIFO.
- The FIFO data type for it must be set to ICV.
- The SNOW 3G f9 mode ICV/MAC is always a 32-bit value.
- If the ICV mode bit is set but the AS field is set to "Initialize" or "Update", an illegal-mode error is generated, except for CICV-only jobs where no processing is done and only ICV check is performed as indicated by data size being 0.
- SNOW 3G f9 generates ICV error if received and computed ICVs do not match.
- It is allowed to create jobs where there is no data to be processed, and only ICV is being checked. For this, the AS mode field should be reset.

## 11.8 Message digest hardware accelerator (MDHA) functionality

The MDHA performs hashing and authentication operations using the hashing algorithms defined in FIPS 180-3 (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512) and MD5. The MDHA also supports SMAC with MD5 and SHA-1 that is used with SSL 3.0 and HMAC that is used by TLS and other protocols. MDHA is controlled by the Class 2 registers.

### 11.8.1 MDHA use of the Mode Register

The MDHA uses the Mode Register as follows:

- The Encryption field (ENC) is not used by the MDHA, and the Authenticate/Protect (AP) field is used only for selecting the appropriate Performance Counter register.
- The Algorithm field (ALG) must be programmed to MD5, SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.
- The ICV field enables ICV checking for MDHA. Starting at the MSB, MDHA verifies the number of bytes in the digest that are defined in the Class 2 ICV Size Register.
- The Algorithm State (AS) field is defined as follows:

**Table 11-86. Mode Register[AS] operation selections in MDHA**

| Operation | Description |
|---|---|
| INIT | The hashing algorithm is initialized with the chaining variables and then hashing begins. Input data must be a non-zero multiple of 64-byte blocks for MD5, SHA-1, SHA-224, SHA-256, and 128-byte blocks for SHA-384 and SHA-512. |
| INIT/FINALIZE | The hashing algorithm is initialized with the chaining variables, and padding is automatically put on the final block of data. Any size of data is supported. |
| UPDATE | The hashing algorithm begins hashing with an intermediate context and running message length. Input data must be a multiple of 64-byte blocks for MD5, SHA-1, SHA-224, SHA-256, and 128- byte blocks for SHA-384 and SHA-512. |
| FINALIZE | The hashing algorithms begin hashing with an intermediate context and running message length. Padding is performed on the final block of data. Any size of data is supported. |

- The Additional Algorithm Information field (AAI) field is defined as follows:
  - The Additional Algorithm Information field (AAI) specifies whether Authentication is performed on the data with the specified algorithm. The optional authentication modes are HMAC, SMAC, and HMAC with precomputed IPAD/OPAD.

- The HMAC mode is defined by FIPS 198-1. This can be performed with any of the hashing algorithms.
- The SMAC mode is defined by the SSL 3.0 specification. This can be performed with MD5 or SHA-1 hashing algorithm only.
- The HMAC with precomputed IPAD/OPAD performs the HMAC algorithm but allows the IPAD and OPAD step to be preloaded and started from instead of the KEY.

**NOTE**

For HMAC and SMAC, the MD5 Key cannot be shared between DECOs until the donor MDHA is done. As a result, if using an MD5 key in a shared descriptor, sharing must be set to NEVER, WAIT or SERIAL, and sharing cannot be permitted to proceed until MDHA is done. For more information on sharing, please refer to Table 7-1.

## 11.8.2 MDHA use of the Key Register

The MDHA uses the Key Register as follows:

- The Key Register is only used when one of the AAI field bits are specified.
- These registers either hold the key or the precomputed IPAD/OPAD split key.
- The size of the IPAD and OPAD are each the size of the digest that is defined by the specified algorithm, except for SHA-224, which is 32 bytes, and SHA-384, which is 64 bytes.

### 11.8.2.1 Using the MDHA Key Register with normal keys

When loading the Key Register with the Key Command (See KEY commands), a KDEST value of 0h results in the key source being loaded, with offset zero, into the Key Register. If the ENC bit = 1 in the Key Command, then the key is decrypted into this register.

### 11.8.2.2 Using the MDHA Key Register with IPAD/OPAD "split keys"

The HMAC function uses an HMAC key per the following equation:

HMAC(Key,Message) = Hash[(Key $\oplus$ OPAD) || HASH((Key $\oplus$ IPAD) || Message)],

where "IPAD" and "OPAD" are constants, "Key" is the HMAC Key, and "HASH" is the chosen hashing function (for example, SHA-256).

#### 11.8.2.2.1 Definition and function of IPAD/OPAD split keys

To improve performance, SEC permits the use of pre-computed IPAD/OPAD "split keys". Computing the values Hash(Key ⊕ IPAD) and Hash(Key ⊕ OPAD) each require MDHA to perform one block of hashing computation. By perforrming this computation once, subsequent HMAC computations can save two blocks of a HMAC computation. As a result, the Key Command has an option to signal MDHA that the key being loaded is a precomputed split key.

#### 11.8.2.2.2 Process flow of using the Key Register with split keys

When MDHA runs, it turns a key into an IPAD/OPAD pair. MDHA writes this pair back to the Class 2 Key Register. Because the IPAD/OPAD pair is required every time, it saves time to create it once and then reuse it rather than starting with the key again. However, the IPAD/OPAD pair does not appear in the Key Register contiguously. To make the hardware much simpler, the IPAD appears at the start of the Key Register and the OPAD starts at the midpoint. So, between a 16-byte IPAD and the 16-byte OPAD, there can be 48 bytes of null data. Rather than having to load or store the null data, use the split key type.

#### 11.8.2.2.3 Using padding with the split key type to align with storage

When doing a FIFO STORE of the split key type, the user provides a length equivalent to the sum of the bytes in the IPAD and OPAD. That is, in the example above, a total length of 32 bytes. DECO knows to get 16 bytes from the start and another 16 bytes from the midpoint. This saves in encryption time and bandwidth. When loading 20-byte IPAD and OPAD (size would be 40), there must be "padding" of 4 bytes following each. That is: {20 bytes of IPAD, 4 bytes of PAD, 20 bytes of OPAD, 4 bytes of PAD}. The padding can be anything, because SEC discards it. The reason for this is to make it align with how the encrypted split key is stored, where the extra padding is used to pad each of IPAD and OPAD to an 8-byte boundary so that they can be handled separately.

#### 11.8.2.2.4 Length of a split key

Because the split key consists of two blocks of material processed by the selected hash algorithm, the length of a split key is twice the length of the hash algorithm's running digest (note exception below). Storage of the split Key in the Class 2 Key Register, however, is such that the value Hash(Key ⊕ IPAD) is at offset zero, and the value Hash(Key ⊕ OPAD) is at offset 64.

### 11.8.2.2.5 Loading/storing a split key with a key command

A split key may be loaded either in encrypted or in unencrypted form. The DECO command to load a split key is the key command with the KDEST field set to 3h. A split key loaded in this way must be stored contiguously in external memory, and is twice the length of the hash algorithm's running digest. For example, the running digest for SHA-224 is 32 bytes, so the length of a SHA-224 split key in external memory is 64 bytes. In addition, the length for a key command associated with a SHA-224 split key must be 64. A split key that SEC has generated has also been encrypted, and for SHA-1, has been padded with 8 additional bytes.

### 11.8.2.2.6 Loading/storing a split key with a FIFO STORE command

The DECO command to store a split key is the FIFO STORE command (see FIFO STORE command), with output-data type set to 16 or 26 (17 or 27 to encrypt using the TDKEK). Generating a split key in this fashion results in the key being encrypted with the JDKEK or TDKEK. Note that the length of an encrypted split key is longer if the FIFO STORE command output-data type selects AES-CCM (16 or 17) for the encrypted key type. Even if AES-ECB is selected (26 or 27), a SHA-1 encrypted split key is always longer, because SEC must add 8 bytes of padding before the pre-encrypted 40 bytes of actual split key can be encrypted.

### 11.8.2.2.7 Sizes of split keys

This table describes the different sizes of split keys depending on how they were generated.

**Table 11-87. Sizes of split keys**

| Hash algorithm | Final digest size | Running digest size | Software-generated split-key Size | AES-ECB encrypted split-key size | AES-CCM encrypted split-key size |
|---|---|---|---|---|---|
| MD5 | 16 bytes | 16 bytes | 32 bytes | 32 bytes | 44 bytes |
| SHA-1 | 20 bytes | 20 bytes | 40 bytes | 48 bytes | 52 bytes |
| SHA-224 | 28 bytes | 32 bytes | 64 bytes | 64 bytes | 76 bytes |
| SHA-256 | 32 bytes | 32 bytes | 64 bytes | 64 bytes | 76 bytes |
| SHA-384 | 48 bytes | 64 bytes | 128 bytes | 96 bytes | 140 bytes |
| SHA-512 | 64 bytes | 64 bytes | 128 bytes | 128 bytes | 140 bytes |

### 11.8.2.2.8 Constructing an HMAC-SHA-1 split key in memory

This figure is an example of how software would construct an HMAC-SHA-1 split key in memory.

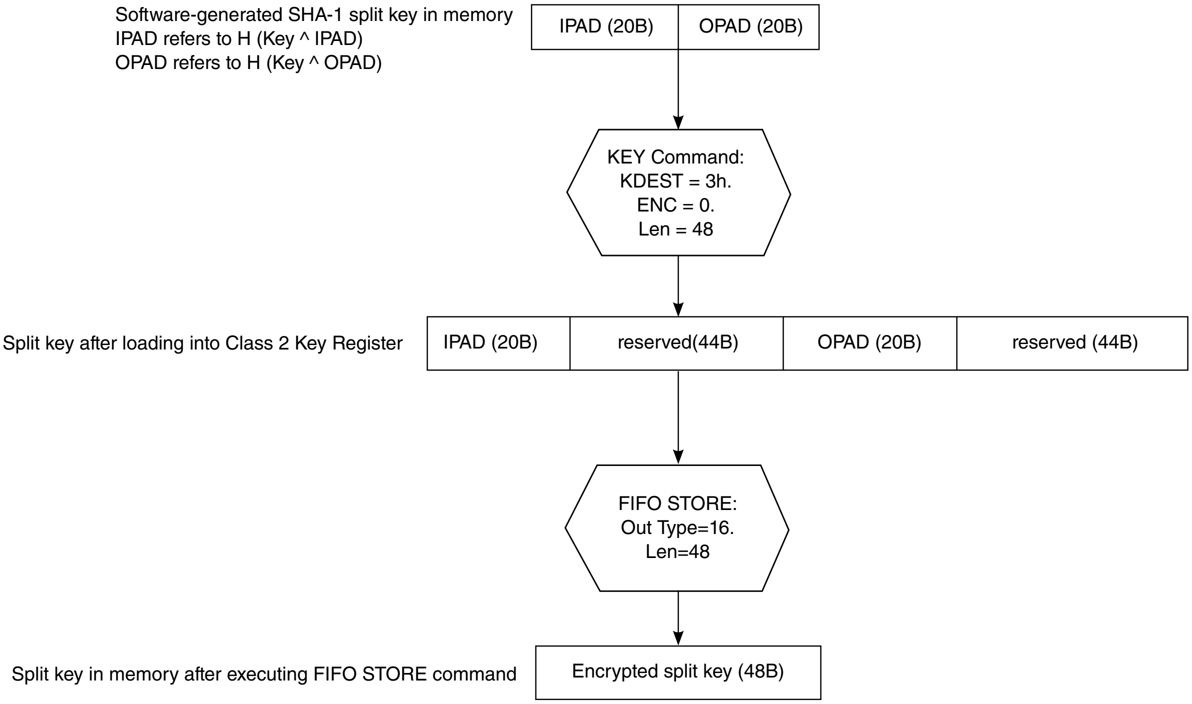

Software-generated SHA-1 split key in memory
IPAD refers to H (Key ^ IPAD)
OPAD refers to H (Key ^ OPAD)
IPAD (20B)
OPAD (20B)
KEY Command:
KDEST = 3h.
ENC = 0.
Len = 48
Split key after loading into Class 2 Key Register
IPAD (20B)
reserved(44B)
OPAD (20B)
reserved (44B)
FIFO STORE:
Out Type=16.
Len=48
Split key in memory after executing FIFO STORE command
Encrypted split key (48B)

**Figure 11-1. Split keys in memory and in the Class 2 Key Register**

Use the KEY Command to load it into the Class 2 Key Register, and then use the FIFO STORE Command to write it back out in encrypted form.

### 11.8.2.3 MDHA use of the Key Size Register

The Key Size Register is defined to be the number of bytes of key that is loaded into the Key Registers. Key Size ranges are defined as followed:

- MD5: 0 → 64 bytes
- SHA-1: 0 → 64 bytes
- SHA-224: 0 → 64 bytes
- SHA-256: 0 → 64 bytes
- SHA-384: 0 → 128 bytes
- SHA-512: 0 → 128 bytes

## 11.8.3 MDHA use of the Data Size Register

The MDHA uses the Data size Register as follows:

- The Data Size Register is written with the number of bytes of data to be processed.

- This register must be written to start data processing.
- This register may be written multiple times while data processing is in progress in order to add the amount written to the register to the previous value in the register.

### 11.8.4 MDHA use of the Context Register

The Context Register stores the current digest and running message length. The running message length will be 8 bytes immediately following the active digest. The digest size is defined as follows:

- MD5: 16 bytes
- SHA-1: 20 bytes
- SHA-224: 28 bytes final digest; 32 bytes running digest
- SHA-256: 32 bytes
- SHA-384: 48 bytes final digest; 64 bytes running digest
- SHA-512: 64 bytes

### 11.8.5 Save and restore operations in MDHA context data

MDHA is able to process data in chunks by saving the intermediate context and running message length from the Context Register after each chunk of data and restoring the context and running message length to the Context Registers before processing any subsequent chunks of data.

## 11.9 AES accelerator (AESA) functionality

The advanced encryption standard accelerator (AESA) module is a hardware co-processor capable of accelerating the advanced encryption standard (AES) cryptographic algorithm.

### 11.9.1 Differences between the AES encrypt and decrypt keys

AES is a block cipher that processes data in 128-bit blocks. It is a symmetric key algorithm, that is, the "same"[1] key is used for both encryption and decryption, although the key appears in a different form for decryption than it does for encryption. The decrypt form of the key is different from the encrypt form of the key because AES successively

[1] The two forms are considered the same key because one can be derived from the other.

modifies the cryptographic key during the steps of the cryptographic operation. The decryption operation yields the correct result only if the modified form of the key (the decrypt key) is used at the beginning of the decryption operation. Unless told otherwise (via the DK bit in the OPERATION command), SEC assumes that a key loaded from memory is the encrypt key, that is, the form appropriate for encryption. If a decryption operation is specified and DK = 0, SEC first goes through the steps required to derive the decrypt key from the encrypt key, and then performs the decryption operation. If a decryption operation is specified and DK = 1 (indicating that a decrypt key has been loaded), the steps required to derive the decrypt key are skipped and the decryption operation is performed immediately, significantly improving performance for small data blocks.

Note that the difference between the encrypt key and the decrypt key must be taken into account when sharing keys between jobs. When an AES decryption job loads a key from memory, it is probably an encrypt key, so the DK bit in the OPERATION command should be set to 0 so that SEC derives the decrypt key from the encrypt key before beginning the decryption operation. But when a subsequent AES decryption job shares the key from a previous decryption job, the key that is shared is a decrypt key. In that case, the DK bit should be set to 1, which tells SEC to skip the key derivation steps. If DK were set to 0 in this case, the decrypt key would be modified as if it were an encrypt key, and consequently, the wrong key value would be used in the decryption operation. Note that a JUMP command with TEST CONDITION set to SHRD (see Table 7-90) can be used to determine whether the OPERATION command should be executed with DK = 0 or DK = 1.

### 11.9.2 AESA as both Class 1 and Class 2 CHA

AESA can be programmed as either a Class 1 or a Class 2 CHA. When used as a Class 1 CHA, all of the modes of operation that it implements are available. Thus, it can perform both confidentiality and authentication tasks. In this case, AESA is programmed via the Class 1 CCB interface and all of the descriptor commands referring to AESA, its data, context or keys must also use Class 1 designator where appropriate.

In order to support processing modes where an AES algorithm is used to perform authentication while another Class 1 CHA is being used for encryption or decryption, AESA can be programmed as a Class 2 CHA, i.e. using the Class 2 CCB interface. In this case only authentication algorithms are available, specifically XCBC-MAC and CMAC. An attempt to program AESA to perform any other mode algorithm will cause an illegal-mode error. When used as a Class 2 CHA, AESA and its input/output data, context data or keys must be referred to as Class 2 in any descriptor command that contains the Class field. The designator for AESA in the ALG field of a mode register remains the same for both Class 1 and 2 mode registers.

The ability to function either as a Class 1 or a Class 2 CHA is further leveraged to allow AESA to operate simultaneously as both a Class 1 and a Class 2 CHA. This allows the user to perform encryption/decryption using any of the supported AES confidentiality modes except the XTS, i.e. ECB, CBC, CTR, OFB, CFB128, while simultaneously performing authentication using either XCBC-MAC or CMAC modes. An attempt to program AESA via Class 1 and 2 mode registers with any other combination of modes will cause an illegal-mode error.

When operating simultaneously as both a Class 1 and a Class 2 CHA, AESA will switch between processing Class 1 or 2 data blocks of 16 bytes after each processed block as long as both Class 1 and 2 blocks are fetched from the corresponding FIFO interface and are ready for processing. It will continue with processing blocks belonging to one Class as long as the data belonging to the other Class is not available. AESA can prefetch data from both Class 1 and Class 2 FIFO interfaces simultaneously. If an error is generated for either Class 1 or Class 2 jobs, the processing is terminated for both classes. The reported error status will contain the appropriate class designator.

AESA can be used either with an in-snooping or an out-snooping data flow. If the Class 2 job needs to process the same data as the Class 1 job, CCB should be programmed to utilize in-snooping. Alternatively, if the Class 2 job performs authentication on the Class 1 job result, out-snooping should be used. If an authentication job using XCBC-MAC or CMAC is to be performed on data in memory, AESA can be programmed either as Class 1 or Class 2.

The descriptor sharing and context switching between different jobs can be utilized with descriptors programming AESA either as Class 1 or Class 2 or both.

When programmed to process Class 1 and Class 2 jobs simultaneously, AESA is considered busy until both jobs complete. For example, after the Class 1 job completes, it is not possible to start another Class 1 job while the initial Class 2 job is still being processed, or vice versa. When clearing an interrupt or issuing a software reset, the corresponding Class Mode Register should also be cleared. The internal registers will not be cleared if AESA is still selected by the other Class Mode Register.

### 11.9.3 AESA modes of operation

The following modes are supported by AESA:

- Electronic codebook (ECB)

- Cipher block chaining (CBC)
- Output feedback (OFB)
- 128-bit cipher feedback (CFB128)
- Counter (CTR)
- XTS tweakable block cipher
- Extended cipher block chaining message authentication code (XCBC-MAC)
- Cipher-based MAC (CMAC)
- CTR and CBC-MAC (CCM)
- Galois/Counter mode (GCM)
- Combined CBC and XCBC (CBC-XCBC)
- Combined CTR and XCBC (CTR-XCBC)
- Combined CBC and CMAC (CBC-CMAC)
- Combined CTR and CMAC (CTR-CMAC)

AES modes can be classified into these categories:

- Confidentiality (ECB, CBC, CTR, OFB, CFB128,XTS)
- Authenticated Confidentiality (CCM, GCM,GCM, CBC-XCBC, CTR-XCBC)
- Authentication (XCBC-MAC, CMAC)

CBC Mode can also be viewed as an authentication mode when used to encrypt data, because it provides CBC-MAC in the context registers.

### 11.9.4 AESA use of registers

Note the following regarding the AESA's use of registers:

- AESA is controlled by either the Class 1 or Class 2 registers.
- For all modes, if AES is selected and the mode code written to the Mode Register does not correspond to any of the implemented AES modes, the illegal-mode error is generated.
- KEY SIZE, MODE and DATA SIZE can be written in any order. The operation will begin after all of these have been written. As a Class 2 CHA, only XBC-MAC and CMAC authentication modes are available. Writing the Class 2 Mode Register to request any other modes will cause an illegal-mode error. Also, for all AES modes, the bit offset in the Data Size Register must be zero when the last write to that register is completed. Failure to comply with these requirements will generate an error in the CCB Status Register.
- When sharing context between consecutive AES jobs, software reset is not issued. To prepare AES for the next job, the Data Size Register and Mode Register must be

cleared, as well as the Done Interrupt. The order of these should be such that the Done Interrupt is not cleared first.
- If ICV-only jobs are created (no data to be processed, only ICV to be checked) in modes that support ICV check, the AS mode field should be reset.

### 11.9.5 AESA use of the parity bit

AESA incorporates fault-detection logic based on parity. The parity bit is computed for every byte of input data and key. These parity bits are then fed to the fault detection logic that computes expected parity of every byte for both key and data based on the AES transformations implemented in the main data-path. The expected parity is compared with the parity of the actual key and data bytes and the hardware error is generated if there is a mismatch.

### 11.9.6 AES ECB mode

The electronic codebook (ECB) mode is a confidentiality mode that features, for a given key, the assignment of a fixed, ciphertext block to each plaintext block, analogous to the assignment of code words in a codebook. In ECB encryption, the forward cipher function is applied directly and independently to each block of the plaintext. The resulting sequence of output blocks is the ciphertext. In ECB decryption, the inverse cipher function is applied directly and independently to each block of the ciphertext. The resulting sequence of output blocks is the plaintext.

#### 11.9.6.1 AES ECB mode use of the Mode Register

AES ECB mode uses the Mode Register as follows:

- The Encrypt (ENC) field should be 1 for ECB encryption and 0 for ECB decryption.
- The ICV/TEST bit is used in ECB mode to activate the fault detection test logic. This logic verifies that the fault detection logic is operational by injecting bit-level errors into input data and key bytes. Because ECB mode does not normally use the Context Registers, the first 128 bits of the context are used in the ECB TEST mode to define which byte of the input data and the key has a bit error injected.
- The Algorithm State (AS) field is not used in ECB mode.
- The Additional Algorithm Information (AAI) field must be set with value 20h that activates ECB mode. Setting the MSB in the AAI field (interpreted as the Decrypt Key or DK bit for AES operations) specifies that the key loaded to the Class 1 Key Register is the decryption form of the key, rather than the encryption form of the key.

If DK = 0, when a decryption operation is requested SEC processes the content of the Class 1 Key Register to yield the decryption form of the key. If DK = 1, SEC skips this processing. The illegal-mode error is generated if DK = 1 and ENC=1.

- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

### 11.9.6.2 AES ECB mode use of the Context Register

ECB does not use Context Registers except when fault-detection test is activated. In this case, the first 128 bits of the context are reserved for the error code. The error code:

- Defines which byte of the input data and the key will have a bit error injected.
- Can have 32, 40, or 48 active bits depending on the key size (16, 24 or 32 bytes in ECB mode).
- Is right justified within first 128 bits of the context such that bit 0 of Context DWord 1 injects error into the MSB of the input data, while bit 16 of Context DWord 1 injects error into the MSB of the key.

If all bits of the error code are 0, no error is injected and fault detection logic does not activate the hardware error. If the ICV/TEST bit of the Class 1 Mode Register is 0 in the ECB mode, the Context Registers have no effect on ECB processing.

**Table 11-88. Context usage in ECB mode**

| Context DWord | Definition | |
|---|---|---|
| | ECB | ECB with ICV/TEST = 1 |
| 0 | - | ERROR CODE |
| 1 | - | |

### 11.9.6.3 AES ECB Mode use of the Data Size Register

The length of the message to be processed in bytes must be written to the Data Size register. If this value is not divisible by 16, the Data Size error is generated.

### 11.9.6.4 AES ECB Mode use of the Key Register

ECB keys must be written to the Class 1 Key Register and can have 16, 24, or 32 bytes.

#### 11.9.6.5 AES ECB Mode use of the Key Size Register

The number of bytes in the ECB key must be written to the Key Size register. The KEY SIZE, MODE and DATA SIZE can be written in any order. Processing starts after all of them have been written. Any value other than 16, 24, or 32 causes the key-size error to be generated.

### 11.9.7 AES CBC, OFB, CFB128 modes

The CBC, OFB, CFB128 modes are considered together because of their similarities and are described in this table.

**Table 11-89. AES CBC, OFB, CFB128 modes**

| Name | Abbreviation | Function |
|---|---|---|
| Cipher-block chaining mode | CBC | Confidentiality mode whose encryption process features the combining ("chaining") of the plaintext blocks with the previous ciphertext blocks. The CBC mode requires an IV (Initialization Vector) to combine with the first plaintext block<br>**NOTE:** CBC mode uses both forward and inverse AES cipher. OFB and CFB use only forward AES cipher. |
| Cipher feedback mode | CFB | Confidentiality mode that features the feedback of successive ciphertext segments into the input blocks of the forward cipher to generate output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The CFB mode requires an IV as the initial input block. AESA implements 128-bit CFB mode where every ciphertext/ plaintext block must have 128 bits |
| Output feedback mode | OFB | Confidentiality mode that features the iteration of the forward cipher on an IV to generate a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. The OFB mode requires IV. The last block of OFB input data can have fewer than 16 bytes |

#### 11.9.7.1 AES CBC, OFB, and CFB128 modes use of the Mode Register

The AES CBC, OFB, and CFB128 modes use the Mode Register as follows:

- The Encrypt (ENC) field should be 1 for encryption and 0 for decryption, except for OFB mode in which this bit is not used.
- The ICV/TEST bit is not used in these modes.
- The Algorithm State (AS) field is used only in CBC mode to prevent IV update in the context for the last data block when set to "Finalize" (2h).
- The Additional Algorithm Information (AAI) field defines which mode is used for processing. For CBC, OFB, and CFB, these values are 10h, 40h, and 30h,

respectively. The Decrypt Key [DK] (AAI field MSB) bit affects CBC mode and specifies that the key loaded to the Class 1 Key Register is the decrypt key. The illegal mode error is generated if DK=1 and ENC=1. If the DK bit is set in OFB or CFB128 modes the illegal-mode error is also generated, because these modes do not use inverse AES cipher.

- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

### 11.9.7.2 AES CBC, OFB, and CFB128 modes use of the Context Register

The AES CBC, OFB, and CFB128 modes use the Context Register as follows:

- AES CBC, OFB, and CFB all use the Context Registers to provide IV, which is updated with every processed block of a message. When a message is split into chunks and processed in multiple sessions, the IV must be saved and later restored for the next chunk to be processed correctly. At the end of CBC processing, IV is also the MAC of the message.
- If the AS field of the Mode Register is set to "Finalize" (2h) in the CBC mode, the last IV update is not written to the context. This enables CBC encryption to effectively perform ECB encryption transformation of a single-block message located in the context in place of IV, and with an all-zero block provided as input data through the FIFO without overwriting the context.

**Table 11-90. Context usage in CBC, OFB, CFB modes**

| Context DWord | Definition |
|---|---|
| 0 | IV [127:64] |
| 1 | IV [63:0] |

### 11.9.7.3 AES CBC, OFB, and CFB128 modes use of the Data Size Register

The AES CBC, OFB, and CFB128 modes use the Data Size Register as follows:

- The byte length of the message to be processed must be written to the Data Size Register.
- The first write to this register initiates processing. This register can also be written during processing, in which case the value written is accumulated to the current state of the register.

- After the Data Size Register is written for the last time, its value must be divisible by 16 in CBC and CFB modes, otherwise the data-size error is generated.
- Only OFB decrements the value in this register with every processed block.

#### 11.9.7.4 AES CBC, OFB, and CFB128 modes use of the Key Register

The AES CBC, OFB, and CFB128 modes uses the Key Register as follows:

- A CBC, OFB, or CFB key must be written to the Class 1 Key Register.
- Keys can have 16, 24, or 32 bytes.

#### 11.9.7.5 AES CBC, OFB, and CFB128 modes use of the Key Size Register

The AES CBC, OFB, and CFB128 modes use the Key Size Register as follows:

- The number of bytes in a key must be written to the Class 1 Key Size register by the time that MODE and DATA SIZE have been written.
- Any value other than 16, 24, or 32 causes a key-size error to be generated.

### 11.9.8 AES CTR mode

The counter (CTR) mode is a confidentiality mode that features the application of the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. Note that the counter value must be unique for each data block that is encrypted with the same key. SEC uses a 128-bit counter to ensure that the counter value will not overflow and wrap around.

**NOTE**

It is the user's responsibility to ensure that the same key value is not used again following a reset.

#### 11.9.8.1 AES CTR mode use of the Mode Register

The AES CTR mode uses the Mode Register as follows:

- The Additional Algorithm Information (AAI) field should be set to 00h to activate CTR mode. If the Decrypt Key [DK] (AAI field MSB) bit is set, the illegal-mode

error is generated, because CTR uses only forward AES cipher requiring encryption rather than decryption keys.
- The Algorithm State (AS) field when set to "Finalize" (2h) prevents counter update in the context for the last data block.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

### 11.9.8.2 AES CTR mode use of the Context Register

The AES CTR mode uses the Context Register as follows:

- CTR uses context dwords 2 and 3 to provide initial counter value (CTR0). This value is incremented with every processed block of a message. When a message is split into chunks and processed in multiple sessions, the CTR0 field of context has to be saved and later restored for the next chunk to be processed correctly.
- If the AS field of the Mode Register is set to Finalize (2h) in the CTR mode, the last counter update is not written to the context. This enables CTR encryption to effectively perform ECB encryption transformation of a single-block message located in the context dwords 2 and 3 in place of CTR0 and with all-zero block provided as input data through the FIFO without overwriting the context.

**Table 11-91. Context usage in CTR mode**

| Context dword | Initial-input definition | Context-switching definition |
|---|---|---|
| 0 | - | - |
| 1 | - | - |
| 2 | CTR0 [127:64] | CTRi [127:64] |
| 3 | CTR0 [63:0] | CTRi [63:0] |

### 11.9.8.3 AES CTR mode use of the Data Size Register

The byte-length of the message to be processed must be written to the Data Size register. The first write to this register initiates processing. It can also be written during processing in which case the value written will be accumulated to the current state of the register. After the Data Size register is written for the last time, the value of this register may not be divisible by 16. CTR decrements the value in this register with every processed block.

### 11.9.8.4 AES CTR mode use of the Key Register

- CTR key must be written to the Class 1 Key Register.
- The Key Register can have 16, 24 or 32 bytes.

#### 11.9.8.5 AES CTR mode use of the Key Size Register

The number of bytes in a key must be written to the Class 1 Key Size register. Any value other than 16, 24, or 32 will cause Key Size error to be generated.

### 11.9.9 AES XTS mode

XTS is a tweakable block-cipher that acts on data units (sectors) of 128 bits or more and uses the AES block-cipher as a subroutine. The key material for XTS-AES consists of a data encryption key (used by the AES block cipher) as well as a "tweak key" that is used to incorporate the logical position of the data block into the encryption.

#### 11.9.9.1 AES XTS mode use of the Mode Register

AES XTS uses the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption.
- The ICV/TEST bit is ignored in this mode.
- The Algorithm State (AS) field is ignored in this mode.
- The Additional Algorithm Information (AAI) field's lower 8 bits must be set to 50h for XTS to be activated.
- The Decrypt Key [DK] (AAI field MSB) bit should be set to 1 only if the AES key, written to the Class 1 Key Register with offset 0, is the decryption form of the key. Otherwise, SEC assumes that the key that was loaded is the encryption form of the key, and before beginning a decryption operation SEC first processes the content of the Class 1 Key Register to derive the decryption form of the key.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

#### 11.9.9.2 AES XTS mode use of the Context Register

AES XTS uses the Context Register as follows:

- Because XTS uses two keys (see AES XTS mode use of the Key Register), Key1 (AES key) and Key2 ("tweak" key), and each can be 32 bytes long, both keys cannot always fit in the Key Register. In that case, Key2 spills into first 32 bytes of the Context Register.
- When these keys are 16 bytes long each, XTS does not use the first 32 bytes of the Context Register.

- Context Register dword 4 is used to provide the sector index (I).
- The sector size in bytes must be provided in the low 16 bits of Context Register dword 5.
- When a message is processed in chunks, all of the key and context data (in the first 6 context dwords) except the Block Index (j) must be saved at the end of the last session and restored before starting a new XTS session on the next chunk of the message. The message split must be done on a sector boundary and sectors have integral number of 16-byte blocks, except the last one, whose size can be any number of bytes higher or equal to 16.

**Table 11-92. Context usage in XTS mode**

| Context dword | Initial-input definition | Context-switching definition |
|---|---|---|
| 0 | Key2 [0:63] (only for 64-byte keys) | Key2 [0:63] (only for 64-byte keys) |
| 1 | Key2 [64:127] (only for 64-byte keys) | Key2 [64:127] (only for 64-byte keys) |
| 2 | Key2 [128:191] (only for 64-byte keys) | Key2 [128:191] (only for 64-byte keys) |
| 3 | Key2 [192:255] (only for 64-byte keys) | Key2 [192:255] (only for 64-byte keys) |
| 4 | Sector Index (I) | Sector Index (I) |
| 5 | Sector Size | Block Index (j), Sector Size |

### 11.9.9.3 AES XTS mode use of the Data Size Register

AES XTS uses the Data Size Register as follows:

- The byte-length of the message to be processed must be written to the Data Size register.
- Processing starts when mode, key size, and data size are all written in any order. This register can also be written during processing, in which case the value written will be accumulated to the current state of the register. XTS decrements the value in this register with every processed block.
- The message size does not have to be a multiple of sector size. However, the size of data in the last sector must be at least 16 bytes-otherwise, cipher text stealing method, employed for processing messages whose last block has fewer than 16 bytes, would be done across sector boundary, which would produce incorrect result. When detected, this situation generates the Data Size error. This error is also generated if sector size is 0 or is not a multiple of 16 bytes, or if the total message size is less than 16 bytes.

### 11.9.9.4 AES XTS mode use of the Key Register

AES XTS uses the Key Register as follows:

- The IEEE 1619-2007 standard defining XTS mode refers to a single XTS-AES key of either 256 or 512 bits, but the key is parsed as a concatenation of two fields of equal size called Key1 and Key2 such that: Key = Key1 | Key2.
  - For a 256-bit key, Key1 must be written to the Class 1 Key Register with offset 0, and Key2 with offset 16.
  - For a 512-bit key, Key2 is written to the first 32 bytes of the context. The AES key(Key1) can be either an encrypt key or a decrypt key.
- If the decrypt key is written to the Key Register, the DK bit (MSB of the AAI field in the Class 1 Mode Register) must be set to 1.

### 11.9.9.5 AES XTS mode use of the Key Size Register

AES XTS uses the Key Size Register as follows:

- The total number of key bytes must be written to the Class 1 Key Size register. The KEY SIZE, MODE, and DATA SIZE can be written in any order. Processing starts after all of them have been written.
- Any value other than 32, or 64 will cause Key Size error to be generated.

## 11.9.10 AES XCBC-MAC and CMAC modes

The AES XCBC-MAC and CMAC modes are described together because of their similarities. They are extensions of the AES CBC mode that produces a key-dependent, one--way hash (or message authentication code (MAC)) in a secure fashion across messages of varying lengths. They also provide data-integrity and data-origin authentication regarding the original message source.

### 11.9.10.1 AES XCBC-MAC and CMAC modes use of the Mode Register

The AES XCBC-MAC and CMAC modes use the Mode Register as follows:

- The Encrypt (ENC) bit is ignored.
- The ICV_TEST bit must be set for computed MAC to be compared with the received MAC. The received MAC must be written to the Input Data FIFO after message data

and the FIFO data type must be set to ICV. If this bit is not set, XCBC-MAC and CMAC do not expect received ICV to be supplied after message data.

- The Algorithm State (AS) field is defined for XCBC-MAC as shown in this table.

**Table 11-93. Mode Register[AS] operation selections in AES XCBC-MAC**

| Operation | Description |
|---|---|
| INITIALIZE | Message is processed in multiple sessions and the current session is the first one. During initialization, derived keys K3 and K2 that are XOR-ed with the last message block are computed and stored in the context to be used in the last processing session. The derived key K1 used as an AES key is computed and written back to the Key Register over the original key |
| INITIALIZE/FINALIZE | Message is processed in a single XCBC session and the final MAC is computed |
| UPDATE | Message is processed in multiple sessions and the current session is neither the first nor the last. Derived keys K2 and K3 are provided in the context and the derived key K1 is provided in the Key Register. If decryption is requested, and data size is not written or is set to 0, and ICV_TEST bit is 1 - AS = UPDATE means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session. |
| FINALIZE | Message is processed in multiple sessions and the current session is the last one. Derived keys K2 and K3 are provided in the context and the derived key K1 is provided in the Key Register. The final MAC is computed |

- The Algorithm State (AS) field is defined for CMAC as shown in this table.

**Table 11-94. Mode Register[AS] operation selections in CMAC**

| Operation | Function |
|---|---|
| INITIALIZE | Message is processed in multiple sessions and the current session is the first one. During initialization, the constant L = E(K, 0) is computed as encrypted block of zeros using key K and stored in the context to be used in the last processing session for derivation of keys K1 and K2. One of these keys will be XOR-ed with the last message block. |
| INITIALIZE/FINALIZE | Message is processed in a single session and the final MAC is computed |
| UPDATE | Message is processed in multiple sessions and the current session is neither the first nor the last. The constant L used for key derivation is provided in the context. If decryption is requested, and data size is not written or is set to 0, and ICV_TEST:w bit is 1 - AS = UPDATE means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session |
| FINALIZE | Message is processed in multiple sessions and the current session is the last one. The constant L used for key derivation is provided in the context. The final MAC is computed |

- If the AS field is not set to either "Initialize/Finalize" or "Finalize" and the ICV_TEST bit is set to 1, the illegal-mode error is generated, except for CICV-only jobs.
- The Additional Algorithm Information (AAI) field must be set to 70h for XCBC and 60h for CMAC to be activated. Setting the DK bit (AAI field MSB) will cause the Illegal Mode error.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

### 11.9.10.2 AES XCBC-MAC and CMAC Modes use of the Context Register

The AES XCBC-MAC and CMAC modes use the Context Register as follows:

- No data needs to be provided in the context when starting a new XCBC or CMAC session.
- The computed MAC and the derived keys K2 and K3 are written back to the context by XCBC.
- The computed MAC and the constant L = E(K,0), computed as encrypted block of zeros using key K, are written back to the context by CMAC.
- When a message is split into chunks and processed in multiple sessions, these values need to be saved before context switch and restored before the next chunk of a message is to be processed. At the end of message processing the first 2 dwords of the context contain the MAC value.

**Table 11-95. Context usage in XCBC-MAC and CMAC modes**

| Mode | Context dword | Context-switching definition | Final-result definition |
|---|---|---|---|
| XCBC-MAC | 0 | MAC[127:64] | MAC[127:64] |
| | 1 | MAC[63:0] | MAC[63:0] |
| | 2 | K3[127:64] | - |
| | 3 | K3[63:0] | - |
| | 4 | K2[127:64] | - |
| | 5 | K2[63:0] | - |
| CMAC | 0 | MAC[127:64] | MAC[127:64] |
| | 1 | MAC[63:0] | MAC[63:0] |
| | 2 | L[127:64] | - |
| | 3 | L[63:0] | - |

### 11.9.10.3 AES XCBC-MAC and CMAC modes use of the Class 1 ICV Size Register

The AES XCBC-MAC and CMAC modes use the ICV Size Register as follows:

- This register is used to provide received ICV/MAC byte-size when it is other than 16 bytes.
- The computed ICV/MAC written to the context in the XCBC mode is always 16 bytes.

- In CMAC mode, this register determines also the computed MAC size-the remaining bytes are cleared.
- Supported values for ICV size are 4 to 16 bytes. If this register is 0, the size of ICV is 16 bytes.

#### 11.9.10.4 AES XCBC-MAC and CMAC modes use of the Data Size Register

The AES XCBC-MAC and CMAC modes use the Data Size Register as follows:

- The byte-length of the message to be processed must be written to the Data Size register.
- The first write to this register initiates processing. It can also be written during processing in which case the value written is accumulated to the current state of the register.
- XCBC-MAC and CMAC decrement the value in this register with every processed block.

#### 11.9.10.5 AES XCBC-MAC and CMAC modes use of the Key Register

The AES XCBC-MAC and CMAC modes use the Key Register as follows:

- The key must be written to this register.
- For XCBC-MAC, if the AS mode field is set to either "Initialize" or "Initialize/ Finalize", it is the original XCBC key (K) that must be written here. Otherwise, the derived key (K1) must be restored to this register. CMAC only uses original key K as an AES key.

#### 11.9.10.6 AES XCBC-MAC and CMAC modes use of the Key Size Register

The AES XCBC-MAC and CMAC modes use the Key Size Register as follows:

- The total number of key bytes must be written to the Class 1 Key Size register.
- For XCBC-MAC, any value other than 16 causes a key-size error to be generated. For CMAC, this error is generated only if any value other than 16, 24, or 32 is written.

#### 11.9.10.7 ICV checking in AES XCBC-MAC and CMAC modes

Automatic ICV checking is enabled by setting the ICV_TEST bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/ Finalize"; otherwise the illegal-mode error is generated, except for CICV-only (Check-ICV-only) jobs.

The received ICV must be provided on the FIFO after the message data. The FIFO data type must be set to ICV when it is put on the FIFO. The size of the received and computed ICV is provided in the Class 1 ICV Size register.

If the ICV check detects a mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

### 11.9.11 AESA CCM mode

CCM consists of two related processes: generation encryption and decryption verification, which combine two cryptographic primitives: counter mode encryption (CTR) and cipher-block chaining based authentication (CBC-MAC). Only the forward cipher function of the block cipher algorithm is used within these primitives. Note that the counter value must be unique for each data block that is encrypted with the same key. SEC uses a 128-bit counter to ensure that the counter value does not overflow and wrap around.

**NOTE**

It is the user's responsibility to ensure that the same key value is not used again following a reset.

#### 11.9.11.1 Generation encryption

A cipher-block chaining is applied to the payload, the associated data (AAD), and the nonce to generate a message authentication code (MAC); then counter mode encryption is applied to the MAC and the payload to transform them into an unreadable form, called the ciphertext. Thus, CCM generation encryption expands the size of the payload by the size of the MAC.

#### 11.9.11.2 Decryption verification

Counter-mode decryption is applied to the purported ciphertext to recover the MAC and the corresponding payload; then cipher block chaining is applied to the payload, the received associated data, and the received nonce to verify the correctness of the MAC.

In CCM mode, the FIFO data type must be set to message type for message data, while for AAD, either AAD or message type can be used.

### 11.9.11.3 AES CCM mode use of the Mode Register

The AES CCM mode uses the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption.
- The ICV_TEST bit must be set for CCM to compare computed MAC with the received MAC when decryption is requested.
- The received MAC must be written to the input-data FIFO after message data and the FIFO data type must be set to ICV.
- Setting the ICV_TEST bit causes the received MAC to be decrypted and compared with the computed MAC.
- The number of MSBs to be compared is defined by the MAC size in the CCM IV ($B_0$) as described in the CCM specification.
- If the AS field is set to FINALIZE, but ICV = 0, AESA does not expect received ICV to be put on the input-data FIFO. In that case, MAC is computed and truncated to the specified size for decryption.
- For encryption, the computed MAC is encrypted and truncated to size. The illegal-mode error is generated if ICV = 1 and ENC = 1.
- If ICV = 1 and the decrypted received MAC do not match computed MAC, the ICV error is generated.
- The Algorithm State (AS) field is defined for CCM as follows:

**Table 11-96. Mode Register[AS] operation selections in AES CCM**

| Operation | Description |
|---|---|
| INITIALIZE | Message is processed in multiple sessions and the current session is the first one. During initialization, the initial counter CTR0 is encrypted in the CTR mode and the B0 is processed with the CBC-MAC mode. The resulting values are stored in the context. Also, the size of MAC is decoded from B0 and written to the context. This AS setting must be used whenever the first part (or whole) AAD is being processed |
| INITIALIZE/FINALIZE | Message is processed in a single CCM session and the final MAC is computed and encrypted. The initial counter CTR0 and B0 must be provided in the context |
| UPDATE | Message is processed in multiple sessions and the current session is neither the first nor the last. All context data is restored from the previous session and the key is written to the Key Register. If decryption is requested, and data size is not written or is set to 0, and ICV_TEST bit is 1 - AS=UPDATE means that a CICV-only job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, decrypts it and compares it to the computed MAC that is restored with the rest of the context from the previous session |
| FINALIZE | Message is processed in multiple sessions and the current session is the last one. All context data is restored from the previous session and the key is written to the Key Register. The final MAC is computed and encrypted |

- Whenever AS is set to Initialize or Initialize/Finalize, context registers must be zero.
- If the AS field is not set to either Initialize/Finalize or Finalize and the ICV_TEST bit is set to 1, the illegal-mode error is generated. This does not apply in case when only ICV check is requested as described for AS = UPDATE.
- The Additional Algorithm Information (AAI) field must be set to 80h for CCM to be activated. The C2K bit is used to select a key register. If C2K = 0, CCM uses the key in the Class 1 Key Register. If C2K = 1, CCM uses the key in the Class 2 Key Register. Setting the DK bit causes the illegal-mode error.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

### 11.9.11.4 AES CCM mode use of the Context Register

The AES CCM mode uses the Context Register as follows:

- B0 and the initial counter CTR0 must be provided in the context before the first chunk of the message is to be processed. During initialization, the initial counter CTR0 is encrypted in the CTR mode and B0 (which functions like a CBC-MAC IV in CCM) is processed with the CBC-MAC mode. The resulting values are stored in the context. Also, the size of MAC is decoded from B0 and written to the lower 32 bits of the context dword 6.
- If there is AAD, the first block of it defines its size, and that value is decoded and written to the upper 32 bits of context dword 6. All of the context data must be restored before the next chunk of the message is to be processed in multi-session processing.
- For CCM encryption, the ICV (encrypted final MAC) is written to context words 4 and 5. For CCM decryption, the ICV (received MAC), which is always encrypted, is decrypted to dwords 4 and 5. The final computed MAC is written (in clear) to dwords 0 and 1.

**Table 11-97. Context usage in CCM mode encryption**

| Context DWord | Initial-input definition | Intermediate definition | Final-output definition |
|---|---|---|---|
| 0 | B0[127:64] | intermediate MAC state | MAC[127:64] |
| 1 | B0[63:0] | intermediate MAC state | MAC[63:0] |
| 2 | CTR0[127:64] | CTR[127:64] | - |
| 3 | CTR0[63:0] | CTR[63:0] | - |
| 4 | - | E(CTR0)[127:64], [1] | E(MAC)[127:64] |
| 5 | - | E(CTR0)[63:0[1] | E(MAC)[63:0] |
| 6 | - | AAD size, MAC size; see Table 11-99 | - |

[1] E(x) means encrypted x

Table 11-98. Context usage in CCM mode decryption

| Context DWord | Initial-input definition | Context-switching Definition | Final-result definition |
|---|---|---|---|
| 0 | B0[127:64] | intermediate MAC state | MAC[127:64] |
| 1 | B0[63:0] | intermediate MAC state | MAC[63:0] |
| 2 | CTR0[127:64] | CTR[127:64] | - |
| 3 | CTR0[63:0] | CTR[63:0] | - |
| 4 | - | E(CTR0)[127:64] | Decrypted Received MAC[127:64] |
| 5 | - | E(CTR0)[63:0] | Decrypted Received MAC[63:0] |
| 6 | - | AAD size, MAC size | - |

Table 11-99. Format of Context DWord 6 in AES-CCM mode

| Bit 63 | Bits 62-48 | Bits 47-32 | Bits 31-3 | Bits 2-0 |
|---|---|---|---|---|
| AAD Presence Flag | 0 | AAD Size | 0 | Encoded MAC Size |

### 11.9.11.5 AES CCM mode use of the Data Size Register

The AES CCM mode uses the Data Size Register as follows:

- The byte-length of the message to be processed must be written to the Data Size register.
- The first write to this register initiates processing. It can also be written during processing in which case the value written will be added to the current state of the register.
- CCM decrements the value in this register with every processed block.
- The content of the Data Size register must be divisible by 16 after the last write to it if the AS mode field is set to either "Update" or "Initialize". Otherwise, the data-size error is generated. In other words, message splitting can be done only on a 16-byte boundary.

### 11.9.11.6 AES CCM mode use of the Key Register

CCM key must be written to this register; it is always an encryption key.

### 11.9.11.7 AES CCM mode use of the Key Size Register

The AES CCM mode uses the Key Size Register as follows:

- The total number of key bytes must be written to the Class 1 Key Size register by the time that MODE and DATA SIZE have been written.
- Any value other than 16, 24, or 32 causes a key-size error to be generated.

### 11.9.11.8 AES CCM mode use of the ICV check

The AES CCM mode uses ICV checking as follows:

- Automatic ICV checking is enabled by setting the ICV_TEST bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/Finalize"-otherwise the illegal-mode error is generated, unless data size is 0 indicating ICV check is only requested. Also, if ICV = 1, the ENC bit must be 0.
- The received ICV must be provided on the input data FIFO after the message data. In CCM, received ICV is always encrypted. The FIFO data type must be set to ICV when it is put on the FIFO. The size of the received and computed ICV is for CCM encoded in the B0.
- If the ICV check detects mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

## 11.9.12 AES GCM mode

The AES GCM provides the following:

- Data confidentiality using counter mode (CTR). Note that the counter value must be unique for each data block that is encrypted with the same key. SEC uses a 128-bit counter to ensure that the counter value does not overflow and "wrap around", but it is the user's responsibility to ensure that the same key value is not used again following a reset.
- Authentication (assurance of integrity) of the confidential data using a universal hash function (GHASH) that is defined over a binary Galois (that is, finite) field. GCM can also provide authentication assurance for additional data (AAD) that is not encrypted.
- Stronger authentication assurance than a (non-cryptographic) checksum or error detecting code; in particular, GCM can detect both of the following:
  - Accidental modifications of the data
  - Intentional, unauthorized modifications

#### 11.9.12.1 GMAC

If the GCM input is restricted to data that is not encrypted, the resulting specialization of GCM, called GMAC, is simply an authentication mode on the input data.

#### 11.9.12.2 GCM data types

In the GCM mode, the FIFO data type must be set to the message data type for textdata (payload), AAD type for additional data, IV type for IV data and ICV type for the received ICV. These data types must always be provided in the following order:

1. IV
2. AAD
3. Message data

Any of these may be missing.

#### 11.9.12.3 IV processing

IV is processed using GHASH function if the size of IV is not 12 bytes. The result of IV processing is the initial counter (Y0) value used for encryption/decryption. GHASH function is also performed on AAD and textdata before the MAC can be computed.

#### 11.9.12.4 GCM initialization

GCM initialization is completed when all of the IV data is processed and the initial counter value (Y0) is computed as a result. For that to happen, IV data needs to be supplied through the Input Data FIFO and the FIFO data type must be set to IV.

#### 11.9.12.5 AES GCM mode use of the Mode Register

The AES GCM mode uses the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption. Even though operations performed in either case are identical, the authentication is done of the cipher text in parallel with decryption when ENC = 0, and after encryption of each block when ENC = 1.

- The ICV_TEST bit must be set for GCM to compare computed MAC with the received MAC. The received MAC must be written to the input-data FIFO after message data and the FIFO data type must be set to ICV. If this bit is not set, GCM does not expect received ICV to be supplied after textdata. The illegal-mode error is generated if ICV = 1 and ENC = 1.
- The Algorithm State (AS) field is defined for GCM as shown in this table:

**Table 11-100. Mode Register[AS] operation selections in AES GCM**

| Operation | Value | Description |
|---|---|---|
| INITIALIZE | 1h | Message is processed in multiple sessions and the current session processes final part of IV or textdata; do the final GHASH step, but do not compute MAC.<br>**NOTE:** This AS state does not indicate initialization in GCM; instead, it means that the final step of the GHASH function is to be performed. In general, whenever the final GHASH iteration needs to be computed (either for GHASH(IV) or GHASH(AAD, ciphertext)), and the current message size provided in the Data Size Register is not equal to the total size for either IV, AAD, or textdata, AS should be set to INITIALIZE (1h). Consequently, an AS = 1h also indicates that the Context Registers 6-7 need to provide the total length of IV, AAD, or textdata for this to be accomplished. |
| INITIALIZE/ FINALIZE | 3h | Message is processed in multiple sessions and the current session is the last. The final MAC is computed. |
| UPDATE | 0h | Message is processed in multiple sessions (descriptors) and the current session is not the last. The descriptor contains a non-final part of IV, AAD, textdata (IV, AAD or textdata split between descriptors). If decryption is requested, and data size is not written or is set to 0, and ICV_TEST bit is 1 - AS = UPDATE means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session |
| FINALIZE | 2h | Message is processed in a single session. MAC is computed. |

- If the AS field is not set to either "Initialize/Finalize" or "Finalize" and the ICV_TEST bit is set to 1, the Illegal Mode error will be generated except for CICV-only jobs.

## Proper AS field settings

Assume that a message has IV, AAD, and textdata and each of these types is split into two sessions (descriptors). The first IV descriptor should have AS set to "Update", the second IV Descriptor should have AS set to "Initialize", both AAD Descriptors and the first textdata descriptor should have AS field set to "Update", and the final Descriptor sets AS to "Initialize/Finalize".

- The Additional Algorithm Information (AAI) field must be set to 90h for GCM to be activated. The C2K bit is used to select a Key Register. If C2K = 0, GCM uses the key in the Class 1 Key Register. If C2K = 1, GCM uses the key in the Class 2 Key Register. Setting the DK bit causes an illegal-mode error.
- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

### 11.9.12.6 AES GCM mode use of the Context Register

The AES GCM mode uses the Context Register as follows:

- New message processing does not need any data provided in the context. All of the context data is written back by the GCM mode and needs to be restored before the next data chunk is to be processed in the multi-session processing. The final MAC is written in the context dwords 0-1.
- The initial counter value required for encryption/decryption is derived from IV and written to dwords 4-5. It is also required for the MAC computation.
- The incremented counter is placed in dwords 2-3 and is updated with every encrypted/decrypted block.
- Bit sizes of IV, AAD and textdata are required for GHASH computation and are accumulated in dwords 6-7 when multi-session processing is used.

**Table 11-101. Context usage in GCM mode**

| Context DWord | Context-switching definition | Final-result definition |
|---|---|---|
| 0 | MAC[0:63] | MAC[0:63] |
| 1 | MAC[64:127] | MAC[64:127] |
| 2 | Yi[0:63] | - |
| 3 | Yi[64:127] | - |
| 4 | Y0[0:63] | - |
| 5 | Y0[64:127] | - |
| 6 | IV bit size (during GHASH of IV), AAD bit size (during message processing) | - |
| 7 | textdata bit size | - |

### 11.9.12.7 AES GCM Mode use of the Data Size Register

The AES GCM mode uses the Data Size Register as follows:

- The byte-length of the message to be processed (including IV, AAD and textdata) must be written to the Data Size register (IV and AAD sizes must include padding to the 16 byte boundary).
- he first write to this register initiates processing. It can also be written during processing in which case the value written will be accumulated to the current state of the register.
- GCM decrements the value in this register with every processed block.
- Message splitting must be done only on a 16-byte boundary.

### 11.9.12.8 AES GCM mode use of the Class 1 IV Size Register

The Class 1 IV Size register is written with the number of bytes in the last IV block. If the total IV size is written, only the low 4 bits are registered. GCM needs this information to determine correct byte size of the IV used in the GHASH computation. To do this, GCM also uses the fact that IV size padded to a 16-byte boundary is written to the Data Size register.

### 11.9.12.9 AES GCM mode use of the AAD Size Register

The AAD Size register is written with the number of bytes in the last AAD block. If the total AAD size is written, only the low 4 bits are registered. GCM needs this information to determine correct byte size of the AAD used in the GHASH computation. To do this, GCM also uses the fact that AAD size padded to a 16-byte boundary is written to the Data Size register.

### 11.9.12.10 AES GCM mode use of the Class 1 ICV Size Register

The AES GCM mode uses the Class 1 ICV Size Register as follows:

- This Class 1 register is used to provide ICV/MAC byte-size when it is other than 16 bytes. In that case, the remaining bytes of the ICV/MAC written to the context is zero.
- If the ICV mode bit is set, the Class 1 ICV Size register also determines the number of bytes in the received ICV. Supported values for ICV size are 4 to 16 bytes. If this register is 0, ICV size will be 16 bytes.

### 11.9.12.11 AES GCM mode use of the Key Register

GCM key must be written to this register; it is always an encryption key.

### 11.9.12.12 AES GCM mode use of the Key Size Register

The AES GCM mode uses the Key Size Register as follows:

- The total number of key bytes must be written to the Key Size register.
- Any value other than 16, 24, or 32 causes key-size error to be generated.

### 11.9.12.13 AES GCM mode use of the ICV check

The AES GCM mode uses ICV checking as follows:

- Automatic ICV checking is enabled by setting ICV_TEST bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/Finalize"; otherwise the Illegal Mode error is generated except for CICV-only jobs. Also, if ICV = 1, the ENC bit must be 0.
- The received ICV must be provided on the input-data FIFO after the message data. The FIFO data-type must be set to ICV when it is put on the FIFO. The size of the received and computed ICV is for GCM written to the Class 1 ICV Size register.
- If the ICV check detects mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

## 11.9.13 AESA optimization modes

The AESA optimization modes are as follows:

- CBC-XCBC
- CTR-XCBC
- CBC-CMAC
- CTR-CMAC
- CTR-CMAC-LTE

These modes are described together because of their similarities. Unlike CCM and GCM, these optimization modes are not actual AES modes, but instead are hardware modes that AESA implements to perform two block cipher modes of AES on the same data in the same hardware. These modes are typically used to support specific networking protocols.

CBC-XCBC, CTR-XCBC, CBC-CMAC, and CTR-CMAC modes combine a confidentiality mode with an authentication mode in a way suitable for IPsec. In particular, the encrypted data is processed with XCBC-MAC or CMAC mode.

### 11.9.13.1 CTR-XCBC and CTR-CMAC modes data format

The CTR-XCBC and CTR-CMAC modes data format is a 24-byte header processed with XCBC, followed by 1 or more 16-byte blocks of data processed with CBC and then XCBC, followed by 0 or 1 XCBC-only words of 4 bytes.

#### 11.9.13.2 CTR-XCBC and CTR-CMAC modes message format

The CTR-XCBC and CTR-CMAC modes message format is a 16-byte header processed with XCBC, followed by 1 or more 4-byte words of data processed with CTR and then XCBC, followed by 0 or 1 XCBC-only words of 4 bytes.

#### 11.9.13.3 CTR-CMAC-LTE for LTE PDCP control-plane processing

CTR-CMAC-LTE is designed for LTE PDCP control-plane processing. In particular, unencrypted data is processed with CMAC mode. The data format for CTR-CMAC-LTE is that of a 9-byte segment that is only authenticated, followed by any number of bytes that is both authenticated and encrypted.

#### 11.9.13.4 Authentication-only data

Authentication- or MAC-only data requires special authentication data (SAD) type to be used as FIFO data type. The CBC or CTR data uses message data type, while ICV is using ICV data type.

#### 11.9.13.5 AES optimization modes use of the Mode Register

The AES optimization modes use the Mode Register as follows:

- The Encrypt (ENC) bit must be set to 1 for encryption and 0 for decryption.
- The ICV bit must be set for computed MAC to be compared with the received MAC. The received MAC must be written to the Input Data FIFO after message data and the FIFO data type must be set to ICV. If this bit is not set, AESA does not expect received ICV to be supplied after textdata. The illegal-mode error is generated if ICV = 1 and ENC = 1.
- The Algorithm State (AS) field is defined for all these optimization modes as described in this table.

**Table 11-102. Mode Register[AS] operation selections in AES optimization modes**

| Operation | Value | Description |
|---|---|---|
| INITIALIZE | 1h | Message is processed in multiple sessions and the current session is the first one.<br>• During CBC-XCBC initialization, derived key K3 for XCBC-MAC that is XOR-ed with the last message block is computed and stored in the context to be used in the last processing session. Derived key K1 used as the AES key for XCBC-MAC processing is computed and replaces the original XCBC-MAC key in the Class 2 Key register. XCBC-MAC derived key K2 is never computed, because the message length cannot be divisible by 16 bytes. |

*Table continues on the next page...*

### Table 11-102. Mode Register[AS] operation selections in AES optimization modes (continued)

| Operation | Value | Description |
|---|---|---|
| | | • During CTR-XCBC initialization both K2 and K3 XCBC-MAC keys are computed. Derived key K1, used as the AES key for XCBC-MAC processing, is computed and replaces the original XCBC-MAC key in the Class 2 Key register.<br>• During CMAC-based mode initialization, the key-derived value L is computed and written to Context words 4 and 5. Derived keys K1 and K2, which are derived from L, are computed as needed but not saved into Context.<br>• CTR-CMAC, CTR-XCBC and CTR-CMAC-LTE will generate the Data Size Error if the data size value is not divisible by 16. |
| INITIALIZE/ FINALIZE | - | Message is processed in a single session and the final MAC is computed. For CBC-based modes, only CBC IV must be written to the context. For CTR-based modes, only initial CTR must be written to the context. |
| UPDATE | - | Message is processed in multiple sessions and the current session is neither the first nor the last. For XCBC-MAC-based modes, keys K2 and K3 are provided in the context and the derived key K1 is provided in the Class 2 Key Register. For CMAC-based modes, Key-derived value L is provided in the Context Register, and is used to compute K1 and K2. If decryption is requested, and data size is written to 0, and ICV bit is 1 - AS=UPDATE means that Check ICV (CICV) job is requested. Data size must be written, even if written with 0. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session. CTR-CMAC, CTR-XCBC and CTR-CMAC-LTE will generate the Data Size Error if the data size value is not divisible by 16. Note that a data size of 0 is not supported for CTR-CMAC-LTE |
| FINALIZE | - | Message is processed in multiple sessions and the current session is the last one. For XCBC-MAC based modes keys K2 and K3 are provided in the context and the derived key K1 is provided in the Class 2 Key Register. For CMAC-based modes key-derived value L is provided in the context. The final MAC is computed using either K1 or K2, derived from L as needed |

- If the AS field is not set to either "Initialize/Finalize" or "Finalize" and the ICV bit is set to 1, the Illegal Mode error will be generated, except for CICV-only jobs.
- The Additional Algorithm Information (AAI) field's lower 8 bits must be set as shown in Table 11-103 for the desired Optimization Mode to be activated.
- The Decrypt Key bit (DK) can be used in CBC-based Optimization Modes for decryption to avoid the time required for key expansion by providing already expanded key in the Class 1 Key Register. This must be used in multi-session processing if the expanded key is saved and later restored to the Key Register. For any CTR-based Optimization Mode, setting the DK (Decrypt Key) bit causes the illegal-mode error.

### Table 11-103. Optimization modes

| AAI value (mode [19:27]) | Optimization mode name | Confidentiality mode | Authentication mode |
|---|---|---|---|
| 0A0h | CBC-XCBC | CBC | XCBC-MAC |
| 0B0h | CTR-XCBC | CTR | XCBC-MAC |
| 0C0h | CBC-CMAC | CBC | CMAC |
| 0D0h | CBC-CMAC-LTE | CBC | CMAC |
| 0E0h | CTR-CMAC | CTR | CMAC |

- The Algorithm (ALG) field is used to activate AESA by setting it to 10h.

#### 11.9.13.6 AES optimization modes use of the Context Register

The AES optimization modes use the Context Register as follows:

- A new message processing session needs CBC IV to be provided in the context for CBC-based Optimization Modes, and the initial counter (CTR0) for CTR-based Optimization Modes.
- The final MAC is in all cases written to context words 0-1.
- The XCBC derived keys K3 and K2 are written back to the context by CTR-XCBC and only K3 by CBC-XCBC. Because the data format for the CBC-XCBC guarantees that the message size is never divisible by 16, only K3 XCBC key is needed. For the same reason, splitting of the CBC processed data on a 16 byte boundary means that XCBC/CMAC processed data is not split on the 16 byte boundary. Hence, the context DWord 6 is used by CBC-XCBC and CBC-CMAC to save the least significant word (8 bytes) of the last CBC encrypted data block from the previous session because it cannot be processed until the next 8 bytes are known. Similarly, for CTR-CMAC-LTE, context DWords 6-7 are used to save 9 bytes of previous data/header for the next processing session.
- This mode's 9-byte authenticate-only header cannot be processed until CTR message data is available. Hence, if only header is provided in the current session, it will be saved to the context for processing in the next session. The same applies for message data in later sessions.
- The data continuation flag in context DWord 7 is used only for CTR-CMAC-LTE encryption to signal the presence of data from the previous session and is set/reset by the mode logic automatically in case of context switching.
- For decryption, the context DWords 6-7 are always used to store the remainder of the previous block of data, because the current block on the input needs to be decrypted before a new AES block can be formed by concatenating 9 bytes from the previous block with 7 bytes from the current input block.
- In CTR-XCBC, CTR-CMAC, CTR-CMAC-LTE modes, message splitting must be done on a 16-byte boundary for multi-session message processing.
- The CBC-XCBC and CBC-CMAC modes require that the first session processes the whole authenticate-only header and at least 16 bytes of the CBC data. From that point on, message splitting is supported on a 16-byte boundary of CBC data.

### Table 11-104. Context usage in CBC-XCBC mode

| Context DWord | Initial-input definition | Context-switching definition | Final-result definition |
|---|---|---|---|
| 0 | - | MAC[127:64] | MAC[127:64] |
| 1 | - | MAC[63:0] | MAC[63:0] |
| 2 | CBC-IV[127:64] | CBC-IVi[127:64] | - |
| 3 | CBC-IV[63:0] | CBC-IVi[63:0] | - |
| 4 | - | K3[127:64] | - |
| 5 | - | K3[63:0] | - |
| 6 | - | LS word of CBC encrypted data from previous session | - |

### Table 11-105. Context usage in CTR-XCBC mode

| Context DWord | Initial-input definition | Context-switching definition | Final-result definition |
|---|---|---|---|
| 0 | - | MAC[127:64] | MAC[127:64] |
| 1 | - | MAC[63:0] | MAC[63:0] |
| 2 | CTR0[127:64] | CTRi[127:64] | - |
| 3 | CTR0[63:0] | CTRi[63:0] | - |
| 4 | - | K3[127:64] | - |
| 5 | - | K3[63:0] | - |
| 6 | - | K2[127:64] | - |
| 7 | - | K2[63:0] | - |

### Table 11-106. Context usage in CBC-CMAC mode

| Context DWord | Initial-input definition | Context-switching definition | Final-result definition |
|---|---|---|---|
| 0 | - | MAC[127:64] | MAC[127:64] |
| 1 | - | MAC[63:0] | MAC[63:0] |
| 2 | CBC-IV[127:64] | CBC-IVi[127:64] | - |
| 3 | CBC-IV[63:0] | CBC-IVi[63:0] | - |
| 4 | - | L[127:64] | - |
| 5 | - | L[63:0] | - |
| 6 | - | LS word of CBC encrypted data from previous session | - |

### Table 11-107. Context usage in CTR-CMAC mode

| Context DWord | Initial-input definition | Context-switching definition | Final-result definition |
|---|---|---|---|
| 0 | - | MAC[127:64] | MAC[127:64] |
| 1 | - | MAC[63:0] | MAC[63:0] |
| 2 | CTR0[127:64] | CTRi[127:64] | - |
| 3 | CTR0[63:0] | CTRi[63:0] | - |

*Table continues on the next page...*

Table 11-107. Context usage in CTR-CMAC mode (continued)

| Context DWord | Initial-input definition | Context-switching definition | Final-result definition |
|---|---|---|---|
| 4 | - | L[127:64] | - |
| 5 | - | L[63:0] | - |

Table 11-108. Context usage in CTR-CMAC-LTE mode

| Context DWord | Initial-input definition | Context-switching definition | Final-result definition |
|---|---|---|---|
| 0 | - | MAC[127:64] | MAC[127:64] |
| 1 | - | MAC[63:0] | MAC[63:0] |
| 2 | CTR0[127:64] | CTRi[127:64] | - |
| 3 | CTR0[63:0] | CTRi[63:0] | - |
| 4 | - | L[127:64] | - |
| 5 | - | L[63:0] | - |
| 6 | - | 9 LS bytes of previous session's data and data continuation flag | Encrypted MAC[0:31] |
| 7 | - | | - |

This table summarizes all the ways in which a message can be split for processing in multiple sessions and what settings should be used for each.

Table 11-109. Context DWord 6-7 for CTR-CMAC-LTE mode

| Bits 127-56 | Bits 55-8 | Bit 7 | Bits 6-0 |
|---|---|---|---|
| 9 LSBs of previous session's data | 0 | Data continuation flag | 0 |

## NOTE

The check-ICV-only session assumes that a complete message has been processed and final ICV computed in one or more previous session. When combining sessions from this table to process a message, note that only session 2.1 can be repeated multiple times. Also, if there is a session with AS = 1, then there can only be one such session and there must be one and only one with AS = 2 but none with AS = 3. If there is a session with AS = 3, then there cannot be any other sessions except check-ICV-only.

Table 11-110. Multi-session processing options for AES optimization modes

| Session number | Session description | Mode supported | Mode AS field | Mode ICV field | AAD size | Data size |
|---|---|---|---|---|---|---|
| 1.1 | Initialization-only | CTR-XCBC<br>CTR-CMAC | 1 | 0 | 0 | 0 |

Table continues on the next page...

### Table 11-110. Multi-session processing options for AES optimization modes (continued)

| | | | | | | |
|---|---|---|---|---|---|---|
| | (Computing L for CMAC or K2/K3 for XCBC) | CTR-CMAC-LTE | | | | |
| 1.2 | Header-only | CTR-XCBC<br>CTR-CMAC | 1 or 0 | 0 | 0 | 16 |
| | | CTR-CMAC-LTE | | | 9 | |
| 1.3 | Header and partial message | CTR-XCBC<br>CTR-CMAC | 1 or 0 | 0 | 0 | 16 + 16 * (Number of MDATA blocks) |
| | | CTR-CMAC-LTE | | | 9 | |
| | | CBC-XCBC<br>CBC-CMAC | 1 | | 8 | 24 + 16 * (Number of MDATA blocks) |
| 2.1 | Message-only | All | 0 | 0 | 0 | 16 * (Number of MDATA blocks) |
| 2.2 | Final message | CTR-XCBC<br>CTR-CMAC | 2 | 1 (ENC=0) or 0 | 0 | 4 * (Number of MDATA nibbles) |
| | | CTR-CMAC-LTE | | | | > 0 |
| | | CBC-XCBC<br>CBC-CMAC | | | | 16 * (Number of MDATA blocks) |
| 2.3 | Final message and ESN | CTR-XCBC<br>CTR-CMAC | 2 | 1 (ENC=0) or 0 | 4 | 4 + 4 * (Number of MDATA nibbles) |
| | | CBC-XCBC<br>CBC-CMAC | | | | 4 + 16 * (Number of MDATA blocks) |
| 3.1 | ESN-only | CTR-XCBC<br>CTR-CMAC<br>CBC-XCBC<br>CBC-CMAC | 2 | 1 (ENC=0) or 0 | 4 | 4 |
| 4.1 | Header and final message | CTR-CMAC-LTE | 3 or 2 | 1 (ENC=0) or 0 | 9 | > 16 |
| | | CTR-XCBC | 3 or 2 | | 0 | 16 + 4 * (Number of MDATA nibbles) |
| | | CTR-CMAC | 1 | 0 | | |
| | | CBC-XCBC | 3 | 1 (ENC=0) or 0 | 8 | 24 + 16 * (Number of MDATA blocks) |
| | | CBC-CMAC | 1 | 0 | | |
| 4.2 | Header, final message and ESN | CTR-XCBC<br>CTR-CMAC | 2 | 1 (ENC=0) or 0 | 4 | 20 + 4 * (Number of MDATA nibbles) |
| | | CBC-XCBC<br>CBC-CMAC | | | 12 | 28 + 16 * (Number of MDATA blocks) |
| 5 | Check-ICV-only<br>(For ENC = 0) | All | 0 | 1 | 0 | 0 |

### 11.9.13.7 AES optimization modes use of the Data Size Register

The AES optimization modes use the Data Size Register as follows:

- The byte-length of the message to be authenticated must be written to the Data Size Register. In case of CTR-CMAC-LTE, 16 must be written to this register for the 9-byte, authenticate-only header as the header is provided in 2 FIFO DWords to AES.
- The first write to this register initiates processing. It can also be written during processing, in which case the value written is accumulated to the current state of the register.
- An additional restriction applies to CTR-XCBC and CTR-CMAC modes: the final value of the Data Size Register must be a multiple of 4-byte words, otherwise the data-size error is generated.
- For all CMAC-based optimization modes, the data-size error is generated if the data size value is not divisible by 16 when AS mode field is set to INITIALIZE or UPDATE. CBC-based modes generate the same error if the data size value is 0 and the AS field has INITIALIZE bit set. This is to enforce that the header must be processed with INITIALIZE bit set.

### 11.9.13.8 AES optimization modes use of the AAD Size Register

The AES optimization modes use the AAD Size Register as follows:

- The AAD Size Register is written with the number of bytes in the Authentication-only data. Only the low 4 bits will be registered.
- For CTR-XCBC and CTR-CMAC, because authenticate-only header is always 16 bytes long, the value of the AAD Size Register reflects the presence or absence of the optional trailing 4 bytes.
- For CBC-XCBC and CBC-CMAC, the content of this register is either 8 or 12 based on the fixed size of the 24-byte header and the optional 4 trailing bytes.
- For CTR-CMAC-LTE, 9 must be written to this register for any session that processes the 9-byte header.
- The AAD Size Register must be written before the last write to the Data Size Register.
- Writes to the AAD Size Register are cumulative.

### 11.9.13.9 AES optimization modes use of the Class 1 ICV Size Register

The AES optimization modes use the Class 1 ICV Size Register as follows:

- For optimization modes other than CTR-CMAC-LTE, this Class 1 register is used to provide received ICV/MAC byte-size when it is other than 16 bytes. The Class 1 ICV Size register also determines the number of bytes in the computed ICV/MAC that is provided in context DWords 0-1.
- The computed MAC written to the context is always 16 bytes long.
- Supported values for the ICV size are 4 to 16 bytes. If this register is 0, the ICV size is 16 bytes.
- For CTR-CMAC-LTE, the ICV length is always 4 bytes.
- As long as any bytes trailing the received ICV are zero, there is no need to write this register.
- The encrypted MAC provided in context DWord 6 is also always 4 bytes long.

### 11.9.13.10 AES optimization modes use of the Class 1 Key Register

The AES optimization modes use the Class 1 Key Register as follows:

- The confidentiality key must be written to this register.
- For CTR-based Optimization Modes, it is always an encryption key.
- For CBC-based Optimization Modes, this key must be a decryption key when DK mode bit (AAI field MSB) is set.

### 11.9.13.11 AES optimization modes use of the Class 2 Key Register

The AES optimization modes use the Class 2 Key Register as follows:

- The Class 2 Key Register is used to provide a 16 byte Authentication key.
- It is overwritten by the derived key K1 during XCBC-MAC initialization.
- It is not overwritten for CMAC-based Optimization Modes.

### 11.9.13.12 AES optimization modes use of the Class 1 Key Size Register

The AES optimization modes use the Class 1 Key Size Register as follows:

- The total number of confidentiality-key bytes must be written to the Class 1 Key Size register by the time that Mode Register and Data Size Register have been written.
- Any value other than 16, 24, or 32 causes key-size error to be generated.
- The out-of-sequence error is generated if this Key Size Register is not written by the time both Mode Register and Data Size Register are written.

### 11.9.13.13 AES optimization modes use of the Class 2 Key Size Register

The AES optimization modes use the Class 2 Key Size Register as follows:

- This value is the length of the written authentication key.
- For XCBC-MAC-based modes, they must be 16 bytes long, otherwise the key-size error is generated.
- For CMAC-based modes, 16-, 24-, and 32-byte keys are supported.

### 11.9.13.14 AES optimization modes use of the ICV check

The AES optimization modes use ICV checking as follows:

- Automatic ICV checking is enabled by setting ICV bit of the Mode Register to 1. When ICV is set to 1, the AS mode field must be set to either "Finalize" or "Initialize/Finalize"; otherwise the illegal-mode error is generated, except for CICV-only jobs. Also, if ICV = 1, the ENC bit must be 0.
- The received ICV must be provided on the input-data FIFO after the message data.
- The FIFO data type must be set to ICV when it is put on the FIFO.
- The size of the received ICV is written to the Class 1 ICV Size register.
- If the ICV check detects mismatch between the decrypted received ICV and the computed ICV, the ICV error is generated.

### 11.9.13.15 AES optimization modes error conditions

This table shows all the error checks implemented for the AES optimization modes.

**Table 11-111. AES optimization modes error conditions**

| Description | Modes affected | Mode DK bit | Mode AS field | Mode ICV bit | Mode ENC bit | Data size | Key size | Error type generated |
|---|---|---|---|---|---|---|---|---|
| ICV check can be requested only when decrypting. | All | - | - | 1 | 1 | - | - | Mode error |
| Must compute final ICV/MAC before checking ICV except for check-ICV-only jobs that have no data to be processed (AS = UPDATE, DS = 0). | All | - | 1 or 0 | 1 | - | > 0 | - | |

Table continues on the next page...

### Table 11-111. AES optimization modes error conditions (continued)

| Description | Modes affected | Mode DK bit | Mode AS field | Mode ICV bit | Mode ENC bit | Data size | Key size | Error type generated |
|---|---|---|---|---|---|---|---|---|
| CTR mode does not use the decrypt key. | CTR-XCBC<br>CTR-CMAC<br>CTR-CMAC-LTE | 1 | - | - | - | - | - | |
| Using the decrypt key for encryption would produce wrong result. | CBC-XCBC<br>CBC-CMAC | 1 | - | - | 1 | - | - | |
| Only a messages with integral number of bytes allowed. | All | - | - | - | - | Bit-size not divisible by 8 | - | Data-size error |
| CTR message data must consist of 1 or more 4-byte chunks, that is, ESN cannot be split between blocks. | CTR-CMAC<br>CTR-XCBC | - | - | - | - | Last block of CTR message data does not have 4, 8, 12 or 16 bytes | - | |
| Finalize-only jobs are not supported. | All | - | 3 or 2 | - | - | 0 | - | |
| Header must be processed in an initialize session. | CBC-XCBC<br>CBC-CMAC | - | 1 | - | - | 0 | - | |
| Must switch the context on a block boundary. | CTR-XCBC<br>CTR-CMAC<br>CTR-CMAC-LTE | - | 1 or 0 | - | - | Not divisible by 16 | - | |
| Wrong Class 1 key size | All | - | - | - | - | - | Class 1 key size not 16, 24, or 32 | Key-size error |
| Wrong Class 2 key size | CBC-XCBC<br>CTR-XCBC | - | - | - | - | - | Class 2 key size not 16 | |
| | CBC-CMAC<br>CTR-CMAC<br>CTR-CMAC-LTE | - | - | - | - | - | Class 2 key size not 16, 24, or 32 | |
| Key sizes are not written before both mode and data size are written. | All | - | - | - | - | - | - | Out-of-sequence error |
| The header is loaded for processing in a non-initialize session. | CBC-XCBC<br>CBC-CMAC | - | 2 or 0 | - | - | - | - | |
| The computed ICV does not match received ICV. | All | - | 3 or 2 | 1 | 0 | > 0 | - | ICV error |

Table 11-111. AES optimization modes error conditions

| Description | Modes affected | Mode DK bit | Mode AS field | Mode ICV bit | Mode ENC bit | Data size | Key size | Error type generated |
|---|---|---|---|---|---|---|---|---|
| | | | (or 0 for DS=0) | | | (or 0 for AS=0) | | |

## 11.10 ZUC encryption accelerator (ZUCE) functionality

The ZUCE hardware accelerator implements the encryption mode of operation of the ZUC algorithm. The encryption mode confidentiality algorithm is defined as a stream cipher that encrypts/decrypts blocks of data between 1 and 20000 bits in length. The f9 authentication mode of the ZUC algorithm is implemented in the ZUCE CHA. See Differences between ZUCE and ZUCA for more information. The features of the ZUCE accelerator include the following:

- Message encryption and decryption in encryption (UEA2) mode
- Throughput of up to 4 bytes per cycle
- Support for multiple session message processing through context switching
- Support for Descriptor sharing
- Total message size of up to $2^{32}$ bits (processed in chunks of no more than $2^{17}$-1 bytes (per session)
- Support for any number of bits in the last byte of the message
- Automatic zeroization of the invalid bits in the last incomplete byte of the message

### 11.10.1 Differences between ZUCE and ZUCA

ZUC is a proposed third radio interface cryptographic algorithm set for LTE (in addition to Kasumi and SNOW), which was submitted as a candidate for standardization by a 3GPP member company. ZUC forms the basis of the f8 encryption algorithm and f9 integrity/authentication algorithm. ZUC encryption (ZUCE) is a word-oriented stream cipher that generates a sequence of 32-bit words under the control of a 128-bit key and a 128-bit initialization value. ZUCE is programmed using Class 1 CCB registers, whereas ZUCA is programmed using Class 2 CCB registers. Note that it is possible to encrypt or decrypt data using ZUC and also hash the same data using ZUC authentication via "snooping", that is, passing the same data simultaneously to both CHAs ("in-snooping"), or passing the output of one CHA directly to the input of the other CHA ("out-snooping"). But in those versions of SEC that implement more than one DECO but only one ZUCE CHA and one ZUCA CHA, the descriptor must select the ZUCA CHA first. Selecting the ZUCE CHA and then selecting the ZUCA CHA within the same descriptor results in an error indication.

### 11.10.2 ZUCE use of the Mode Register

ZUCE uses the Mode Register as follows:

- The ZUC encryption accelerator (ZUCE) is enabled by setting the Algorithm (ALG) field of the Class 1 Mode Register to B0.
- The encryption mode is enabled by setting the Additional Algorithm Information (AAI) field to C0.
- The Algorithm State (AS) field should be set to "Initialize" state when a new message is to be processed. The ZUCE accelerator initializes the core engine (keystream generator) based on the key and initialization parameters COUNT-C, BEARER and DIRECTION in a 32-step initialization process. This is a necessary step before keystream generation can begin. It is possible to perform this initialization in advance without the need to provide any input data by writing 0 to the Data Size register. The AS field should be reset (or set to "Update" state) after context switch, assuming that Context Registers are restored, when continuing message processing. In this case, the state of the keystream generator necessary for continuation of message processing is in the Context Registers and initialization is not needed.
- Other fields in the Mode Register have no effect on encryption mode.
- If the AAI field is set to a value that does not correspond to encryption mode, the illegal-mode error is generated. The message processing is initiated by writing a message size to the Data Size Register.

### 11.10.3 ZUCE use of the Context Register

ZUCE uses the Class 1 Key and Context registers. The usage of the Key and Context registers in the encryption mode is shown in Table 11-113. The symbols in the Update Input Definition column represent values written back by ZUCE. These values comprise the state of the keystream generator that must be restored after context switch for the message processing to continue. In encryption mode, the Context Register is treated as an extension of the Key Register, tat is, it is automatically encrypted when saved and decrypted when restored. The symbol IV represents a value that must be written to the Context Register when starting a new Job in the encryption mode. This value consists of ZUCE initialization parameters in the following order:

Table 11-112. Context 1 usage in ZUCE mode

| 0-31 | 32-36 | 37 | 38-63 |
|---|---|---|---|
| Count-C | Bearer | Direction | 0 |

Table 11-113. Key/Context Register usage in ZUCE mode

| Register | DWord Number | Initialize-input definition | Update-input definition |
|---|---|---|---|
| Key Register | 0 | Key[0:63] | s0, s1 |
| | 1 | Key[64:127] | s2, s3 |
| | 2 | - | s4, s5 |
| | 3 | - | s6, s7 |
| Context Register | 0 | Count-C \|\| Bearer \|\| Direction \|\| 0 | s8, s9 |
| | 1 | - | - |
| | 2 | - | - |
| | 3 | - | r1 |
| | 4 | - | r2, r3 |
| | 5 | - | s10, s11 |
| | 6 | - | s12, s13 |
| | 7 | - | s14, s15 |

### 11.10.4 ZUCE use of the Data Size Register

ZUCE uses the Data Size Register as follows:

- ZUCE uses the 16 lsbs of the Class 1 Data Size register to indicate the number of bytes of input data, and the 3 msbs to indicate the number of valid bits in the last byte.
- Writing to the LSB of the Class 1 Data Size register initiates processing. ZUCE internally decrements this value as it processes the message. It continues to process data until the value in the Data Size register reaches zero.
- If 0 is written to the Data Size Register and the AS field of the Mode Register is set to "Initialize", ZUCE keystream generator is initialized and the Context Register contains this initialized state.

### 11.10.5 ZUCE use of the Key Register

A 128-bit key must be written to the Class 1 Key Register with offset of 0 if the AS field of the Mode Register is set to "Initialize". The key is necessary for the initialization of the keystream generator but it is not needed when the AS field of the Mode Register is set to

"Update", that is, when a message processing is continued after context switch. However, the Key Register is used to implement internal state of the keystream generator as depicted in Table 11-113.

### 11.10.6 ZUCE use of the Key Size Register

Writing to this register is not required by ZUCE, because the ZUC encryption key is always 16 bytes long. However, writing a value of 16 to this register is allowed, but writing a value other than 16 causes a key-size error to be generated.

## 11.11 ZUC authentication accelerator (ZUCA) functionality

The ZUCA hardware accelerator implements the f9 authentication mode of the ZUC algorithm. The ZUCA algorithm is based on the same stream cipher (ZUC) as is used by the encryption algorithm (ZUCE). The f8 encryption mode of the ZUC algorithm is implemented in the ZUCE CHA. See Differences between ZUCE and ZUCA for more information. ZUC Authentication is a word-oriented stream cipher that generates a 32-bit Message Authentication Code (MAC) under the control of a 128-bit key and a 128-bit initialization value. The message may be between 1 and 20000 bits in length.

The features of ZUCA include the following:

- Computation of 32-bit MAC in ZUC authentication (UIA2) mode
- Automatic comparison of the received and computed MAC values (ICV check)
- Throughput of up to 2 bytes per cycle
- Support for multiple session message processing through context switching
- Support for Descriptor sharing
- Total message size of up to $2^{32}$ bits (processed in chunks of no more than $2^{17}$-8 bytes per session)
- Supports any number of bits in the last byte of the message
- Automatic zeroization of the invalid bits in the last incomplete byte of the message

### 11.11.1 ZUCA use of the Mode Register

The ZUCA uses the Mode Register as follows:

- The ZUC authentication accelerator (ZUCA) is enabled by setting the Algorithm (ALG) field of the Class 2 Mode Register to C0.

- The authentication mode is enabled by setting the Additional Algorithm Information (AAI) field to C8.
- The Algorithm State (AS) field must be set to "Initialize" state when the first session of message processing is to be performed. This assumes that message processing is split into multiple sessions, that is, that the first one is not also the final session. The ZUCA accelerator initializes the core engine (keystream generator) based on the key and an IV built from initialization parameters COUNT-I, DIRECTION, BEARER and FRESH in a 32-step initialization process. This is a necessary step before keystream generation can begin. It is possible to perform this initialization in advance without the need to provide any input data by writing 0 to the Data Size register.
- If an initialization-only job is executed (data size is 0 and AS set for initialization), "Finalize" in the AS setting is ignored, as well as the ICV mode bit.
- If decryption is requested, and data size is not written or is set to 0, and ICV bit is 1 - AS = "Update" means that Check ICV (CICV) job is requested. The CICV-only job does not process any data, it just pops received ICV/MAC from the Input Data FIFO, and compares it to the computed MAC that is restored with the rest of the context from the previous session.
- The AS field must be set to "Finalize" state when the last session of message processing is to be performed. This enables computation of the MAC.
- The AS field must be set to "Initialize/Finalize" state when the whole message is processed in one session.
- The ICV bit of the Mode Register must be set for the authentication mode to compare computed MAC/ICV with the received ICV. The received ICV must be provided through the Input Data FIFO following the message data at which time the FIFO data type must be set to ICV. If this bit is reset, the authentication mode does not expect ICV to be put on Input Data FIFO.
- The illegal-mode error is generated if ICV bit is set but AS field is not set to either "Initialize/Finalize" or "Finalize" state, except for CICV-only jobs, that is, the data size is non-zero and valid (there will be no more writes to the Data Size Register).
- If the AAI field is set to a value that does not correspond to authentication mode, the illegal-mode error is generated. The message processing is initiated by writing a message size to the Data Size Register.
- When SEC descriptor sharing mode is used with ZUCA, clear mode, followed by clear done interrupt command, must be issued between ZUCA jobs.

## 11.11.2 ZUCA use of the Context Register

ZUCA uses the Context Register as follows:

- This table shows context usage in the authentication mode that is relevant for its programming.

**Table 11-114. Context usage in ZUCA mode**

| Register | Word number | Initialization-input definition | Update state (for context switching) | Finalize-output definition |
|---|---|---|---|---|
| Key Register | 0 | Key[0:63] | - | - |
| | 1 | Key[64:127] | - | - |
| | 2 | - | - | - |
| | 3 | - | - | - |
| | 4 | - | IV | - |
| | 5 | - | z1, z2 | - |
| | 6 | - | z3, z4 | - |
| | 7 | - | z5, bit length | - |
| Context Register | 0 | Count-C \|\| 0 \|\| Direction \|\| 0 | - | {MAC,32'h0} |
| | 1 | {FRESH, 32'h0} (3G) {Bearer, 59'h0} (LTE) | - | - |

- For 3G, the IV value is built as shown in this table.

**Table 11-115. Usage of Context 2 for 3G in ZUCA mode**

| 0-31 | 32-36 | 37 | 38-63 | 64-95 |
|---|---|---|---|---|
| Count-C | 0 | Direction | 0 | FRESH |

- LTE systems do not include a FRESH value in the authentication IV value. It is instead built as shown in this table.

**Table 11-116. Usage of Context 2 for LTE in ZUCA mode**

| 0-31 | 32-36 | 37 | 38-63 | 64-68 | 69-95 |
|---|---|---|---|---|---|
| Count-C | 0 | Direction | 0 | Bearer | 0 |

- At the end of processing, ZUCA overwrites IV in the context word 0 with the MAC/ICV. Because MAC is a 32-bit value, it is written to low-order bit locations (right-justified) and the remaining bits are cleared.
- Values z1-z5 are the keystream words computed during initialization of the authentication mode by the keystream generator. After the initialization stage is complete, the keystream generator is not active any more in authentication mode. The processing is based on the Galois Field (GF) multiplier implemented as part of the authentication mode logic. The bit length is a value copied from the Data Size Register to be used to compute the final MAC. In case of multi-session message

processing, this value represents the total message length as each session's data size is accumulated.
- To read only the final MAC value, the "finalize" option must be present in the AS mode setting. When saving context, the starting address must be the address of the first double word of the Key Registers.

### 11.11.3 ZUCA use of the Data Size Register

ZUCA uses the Data Size Register as follows:

- ZUCA uses the 16 lsbs of the Class 2 Data Size Register to indicate the number of bytes of input data, and bits 63-61 to indicate the number of valid bits in the last byte.
- Writing to the LSB of the Class 2 Data Size Register initiates processing.
- The number of input data bits that ZUCA is to process must be written into the lower 20 bits of the Class 2 Data Size register.
- ZUCA internally decrements this value as it processes the message. It continues to process data until the value in the Data Size register reaches zero. If 0 is written to the Data Size Register and the AS field of the Mode Register is set to "Initialize", ZUCA keystream generator is initialized and the context contains this initialized state.
- In authentication mode, the data size must be divisible by 64 except when the AS field of the Mode Register is set to "Finalize" or "Initialize/Finalize". In other words, the message can be split for multi-session processing only on a 64-bit boundary. If this rule is violated, the illegal data-size error is generated.

### 11.11.4 ZUCA use of the Key Register

A 128-bit key must be written to the Class 2 Key Register with offset of 0 if the AS field of the Mode Register is set to "Initialize" or "Initialize/Finalize". The key is necessary for the initialization of the keystream generator.

### 11.11.5 ZUCA use of the Key Size Register

Writing to this register is not required by ZUCA, because the ZUC authentication key is always 16 bytes long. However, writing a value of 16 to this register is allowed, but writing a value other than 16 causes a key-size error to be generated.

### 11.11.6 ZUCA use of ICV checking

ZUCA uses ICV checking as follows:

- Authentication mode can automatically compare received ICV with the computed ICV at the end of processing if the ICV bit of the Mode Register is set and the AS field is set to "Finalize" or "Initialize/Finalize".
- The received ICV must be supplied after message data through the Input Data FIFO. The FIFO data type for it must be set to ICV. The ZUCA mode ICV/MAC is always a 32-bit value.
- If the ICV mode bit is set but the AS field is set to "Initialize" or "Update", the illega-mode error is generated, except for CICV-only jobs where no processing is done and only ICV check is performed as indicated by data size being 0. ZUCA generates ICV error if received and computed ICVs do not match. It is allowed to create jobs where there is no data to be processed, and only ICV is being checked. For this, the AS mode field should be reset.

# Chapter 12
# Trust Architecture modules

The SEC Trust Architecture functions are performed in the run-time integrity checker (RTIC), and the secure key module.

## 12.1 Run-time integrity checker (RTIC)

The run-time integrity checker (RTIC) is a component of SEC that is used to ensure the integrity of the peripheral memory contents and assist with boot authentication. The RTIC has the ability to verify the memory contents during system boot and during run-time execution. If the memory contents at runtime fail to match a reference hash signature, then a security violation is asserted. This security violation should then be captured by a monitoring device on the platform.

### 12.1.1 RTIC modes of operation

The RTIC modes of operation are described in this table.

**Table 12-1. RTIC modes of operation**

| Mode | Description |
|---|---|
| One-time hash mode | • Used during high assurance boot for code authentication or one time integrity checking<br>• Stores a reference hash result internally and signals an interrupt to the processor |
| Continuous hash mode | • Used at run time to continuously verify the integrity of memory contents<br>• Checks a re-generated hash against an internally stored reference value and interrupts the processor only if an error occurs |

### 12.1.2 RTIC initialization and operation

RTIC supports integrity checking of up to four, independent memory blocks. RTIC's Hash Register File stores a reference hash for each memory block. During the boot stage integrity checking, each independent memory block's content is hashed and the result is stored in the hash register file. At boot-time, the memory contents are read and hashed (authenticated) as quickly as possible (RTIC Throttle Register should be set to 00h) to minimize the performance impact at startup. The reference hash result for each memory block is stored in RTIC for the processor to compare against the signed code hash value. Once RTIC has finished hashing the boot image, RTIC interrupts software, which then can check the generated hash value against digitally-signed hash value(s) stored with the code. Software policy determines what actions to take in the event of a hash mismatch at boot time. Note that in chips supporting high-assurance boot, RTIC's boot-image hashing may take place after secure-boot software validates the first code to execute. This means that any unauthorized code modification would either be caught by the secure boot software before RTIC runs, or trusted software would detect the hash mismatch after RTIC had integrity-checked the boot image.

After the trusted boot software has verified the boot image, the software can put RTIC into run-time mode to ensure that the boot image remains uncorrupted. In run-time mode, RTIC periodically reads a small section of memory, waits for a specified period of time, and then reads another small section of memory. During this process, RTIC computes a hash of the software image. When RTIC has eventually read the entire software image, it compares the newly computed hash with the reference hash that was validated at boot time. If the RTIC hardware detects a hash mismatch, RTIC generates an interrupt and signals a security violation to the chip's security monitor (SecMon) hardware. If the hash matches, RTIC starts over and re-validates the software image. This process repeats until the chip is powered down or RTIC checking is turned off.

### 12.1.3 RTIC use of the Throttle Register

The RTIC scan rate is controlled using the Throttle Register. This allows the user to trade off the software image revalidation rate against memory bandwidth utilization. Depending on the settings, the software image might be revalidated every few seconds or every few days. RTIC also implements a watchdog timer that can be used to ensure that an attacker isn't able to block RTIC's access to memory for an extended period of time. If a DMA read error, illegal address/length error, RTIC Watchdog time-out, or hash mismatch occurs, the RTIC enters an error state and signals a security violation. A hardware reset is required to resume operation.

## 12.1.4 RTIC use of command, configuration, and status registers

The RTIC controller holds the command/configuration registers, which are programmed through SEC's register interface. RTIC uses SEC's DMA interface only to read memory contents. The command/control registers are used to:

- Set the DMA burst and throttle level
- Specify which memory blocks to hash (one time and continuously)
- Enable/Disable/Clear interrupts
- Enable one-time or run-time hashing, software reset, and clear interrupts

A status register in the RTIC indicates the current state of the controller, which includes:

- Interrupt status
- Processing status
- Error status

The controller also contains a comparator to check the generated hash value against the reference hash value.

## 12.1.5 Initializing RTIC

At boot time, RTIC can be used to accelerate software-image verification. This is accomplished by first selecting the hash algorithm (SHA-256 or SHA-512) and one or more RTIC memory blocks by writing to the RTIC Control Register, specifying the areas of memory to be hashed by writing one or more pairs of RTIC Memory Block Address and Length registers and, if necessary, altering the endianness settings via the RTIC Control Register and then writing to the RTIC Command Register to initiate the hashing operations.

At chip-initialization time, RTIC is configured for run-time mode operation by writing to the RTIC Control Register), the RTIC Throttle Register the RTIC Watchdog Register and the RTIC Memory Block Address and Length registers and, if necessary, writing to the RTIC Control Register to change the endianness settings, and then writing to the RTIC Command Register to put RTIC into run-time mode.

## 12.1.6 RTIC Memory Block Address/Length Registers

Up to four independent memory blocks can be hashed by the RTIC, each with its own message digest (reference hash value). The RTIC scans through the memory blocks in the order they are defined in the RTIC Memory Block Address registers. Each of the four

Memory Blocks can be divided into two separate segments, with separate starting addresses and segment lengths. RTIC computes the hash over each Memory Block that is enabled by first reading segment 0 of the Memory Block and then appending segment 1.

Since there can be two segments per Memory Block, each memory block (A, B, C, D) is defined by two address/length register pairs (RTIC Memory Block Address 0 Register / RTIC Memory Block Length 0 Register and RTIC Memory Block Address 1 Register / RTIC Memory Block Length 1 Register). For each memory block, starting at the address indicated in the RTIC Memory Block Address 0 Register RTIC reads the number of memory bytes specified in the RTIC Memory Block Length 0 Register. When that is complete, RTIC starts at the address indicated in the RTIC Memory Block Address 1 Register and reads the number of memory bytes specified in the RTIC Memory Block Length 1 Register. If a Length Register is set to zero, RTIC skips over that memory segment. Once the specified number of bytes are read from both segments within the memory block, the data is hashed and stored (Hash-Once mode) or compared to the reference value (Run-Time mode). Information on additional registers used for RTIC configuration can be found in the sections describing the RTIC registers in SEC register page 6.

## 12.2 SEC virtualization and security domain identifiers (SDIDs)

This section describes the SEC features that are intended to support virtualization of the SEC hardware; that is, the ability to share the SEC functionality among multiple software entities.

### 12.2.1 Virtualization

SEC has been designed so that it can be "virtualized", that is, it can be shared among multiple software entities while still maintaining individual security protections for each of these entities. These software entities might include guest operating systems running under a hypervisor that allocates the chip's hardware resources among the guest OSs.

### 12.2.2 Security domain identifiers (SDIDs)

SEC implements 4096 security domain identifier (SDID) values that system control software (for example: hypervisor, kernel, operating system) can associate with different software entities. SDIDs are used to provide separation between software entities, and are used with black keys, blobs, and trusted descriptors. SDIDs are used to provide separation between software entities, and are used with black keys, blobs, and trusted descriptors. An SDID is a static value that must be maintained across power cycles as it is used to provide separation even across power cycles. A unique SDID value can be associated with each software entity, or the same SDID value could be associated with multiple software entities, or multiple SDID values could be associated with a single software entity. The SDID values must be assigned at boot configuration time, and the same SDID always assigned to the same software entity. In the case of a guest OS running under a hypervisor, the guest OS may assign some of its different SDID values to processes under the guest OS's control. Note that the hypervisor is itself one of the software entities that can utilize SEC's functionality, and can assign itself as many SDID values as it wishes. SEC imposes no restrictions on how these SDID values are assigned, but simply uses the SDID values to control how data is shared among SDID assignees, or kept private to each assignee. SDID values are assigned to the queue manager interface, RTIC and to job rings by writing to registers in page 0 of SEC's register address space. The sections below describe how SEC uses the SDID values to "virtualize" black keys, trusted descriptors and blobs.

### 12.2.3 TrustZone SecureWorld

SEC recognizes TrustZone SecureWorld as a unique software entity with special privileges, and identifies SecureWorld using a special "TZ" security identifier. All SEC registers that are used to hold 12-bit SDID values also have a separate TZ bit. Hardware signals ensure that only TrustZone SecureWorld can write to TZ bits. TrustZone SecureWorld indicates that specific SEC resources belong to SecureWorld by setting the TZ bit to 1. This allows SecureWorld to generate black keys that cannot be encrypted or decrypted by non-SecureWorld, to encapsulate and decapsulate blobs that cannot be encapsulated or decapsulated by non-SecureWorld, and to claim SEC job rings and secure memory partitions for its exclusive use. The sections below describe how SEC enforces these SecureWorld privileges.

## 12.3 Special-purpose cryptographic keys

SEC provides protection of session keys by means of black keys (see Black keys), integrity protection of SEC descriptors by means of trusted descriptors (see Trusted descriptors), and protection of long-term secrets by means of blobs (see Blobs). All of these protection mechanisms make use of special-purpose cryptographic keys that are managed by SEC.

### 12.3.1 Initializing and clearing black and trusted descriptor keys

The SEC hardware implements special black key key encryption keys (see Black keys) and trusted descriptor signing keys (see Trusted descriptors). These must be initialized to random values each time that SEC powers up. These keys are cleared when SecMon enters a FAIL state. The hardware implementation ensures that only SEC itself can use these keys. The values cannot be read or extracted from the chip by any means. However, test values can be written or read by software for debug purposes when SEC is in non-secure mode, (see Keys available in non-secure mode).

### 12.3.2 Black keys and JDKEK/TDKEK

One special cryptographic key used with the black key mechanism is the 256-bit job descriptor key encryption key (JDKEK) (see Black keys). When a job descriptor instructs SEC to store a Key Register into memory, the hardware first encrypts the content of the Key Register using the JDKEK and then stores the resulting black key into memory. When a job descriptor later references that key, the descriptor identifies the key as a black key, causing the hardware to decrypt the key using the JDKEK before loading the key into a Key Register. Trusted descriptors can also use the JDKEK, but they are permitted to choose the 256-bit trusted descriptor key encryption key (TDKEK) instead of the JDKEK. Using the TDKEK ensures that only trusted descriptors can use particularly sensitive keys, such as keys that are used to derive session keys. If a TDKEK-encrypted key is embedded as immediate data within a trusted descriptor, this ensures that no other key could be substituted for that particular key.

### 12.3.3 Trusted descriptors and TDSK

The SEC hardware controls use of the 256-bit trusted descriptor signing key (TDSK) that is used to compute the signature (keyed hash) over trusted descriptors (see Trusted descriptors.). The TDSK is used for verifying the signature whenever a trusted descriptor is executed. The TDSK is used to sign a descriptor only if the descriptor is executed in a specially privileged job ring, or if a trusted descriptor modifies itself during execution.

### 12.3.4 Master key and blobs

The special cryptographic key used for blobs is the 256-bit master key that SEC receives from SecMon. The SEC hardware uses this master key to derive keys that are used for blob encryption and decryption when SEC is in secure mode or trusted mode, but uses a known test key for key derivation when SEC is in non-secure mode or fail mode.

## 12.4 Black keys

SEC's black key mechanism is intended for protection of user keys against bus snooping while the keys are being written to or read from memory external to the SoC. The black key mechanism automatically encapsulates and decapsulates cryptographic keys on-the-fly in an encrypted data structure called a black key. Before a value is copied from a key register to memory, SEC automatically encrypts the key as a black key (encrypted key) using as the encryption key the current value in the JDKEKR or TDKEKR, modified via the appropriate TZ/SDID value. Thus, each security domain (and TrustZone SecureWorld) has its own private black keys, which cannot be decrypted by the user of a different security domain identifier. When SEC is instructed to use a black key as an encryption key, SEC automatically decrypts the black key and places it directly into a key register before using the decrypted value in the user-specified cryptographic operation.

### 12.4.1 Black key encapsulation schemes

SEC supports two different black key encapsulation schemes, one intended for quick decryption, and another intended for high assurance.

- The quick decryption scheme uses AES-ECB encryption.
- The high-assurance black key scheme uses AES-CCM encryption. The AES-CCM mode is not as fast as AES-ECB mode, but AES-CCM includes an "MAC tag" (integrity check value) that ensures the integrity of the encapsulated key. SEC does not mix the length of the encrypted key into the value of the key encryption key when using the high assurance black key scheme, because the MAC-tag prevents misrepresenting the length of the encrypted key. In AES-CCM encryption the AES algorithm is always used in the "encryption" direction regardless of whether the key is being encrypted or decrypted, so in the high-assurance black key scheme encapulation and decapsulation require approximately the same amount of time.

### 12.4.2 Differences between black and red keys

Differences between black keys and red keys include the following:

- Black keys are encrypted, while red keys are un-encrypted.
- A black key is usually longer than the red key that is encapsulated. ECB encrypted data is a multiple of 16 bytes long, because ECB is a block cipher with a block length of 16 bytes. So if the red key that is to be encapsulated in an ECB-black key is not already a multiple of 16 bytes long, it is padded with zeros to make it a multiple of 16 bytes long before it is encrypted, and the resulting black key is this length.
- A CCM-encrypted black key is always at least 12 bytes longer than the encapsulated red key, because the encapsulation uses a 6-byte nonce and adds a 6-byte ICV. If the key is not already a multiple of 8 bytes long, it is padded as necessary so that it is a multiple of 8 bytes long. The nonce and ICV add another 12 bytes to the length.

### 12.4.3 Loading red keys

Red keys can be loaded into Key Registers using either a LOAD command or a KEY command with ENC = 0. But keys cannot be stored from Key Registers back to memory in red form. The only way to store keys back out to memory is in black form. This is accomplished by using the FIFO STORE command with an appropriate OUTPUT DATA TYPE value (see Table 7-31, values 10h-27h).

### 12.4.4 Loading black keys

The only way that black keys can be successfully loaded is by using a KEY command with ENC = 1 and the proper setting of the EKT bit. The EKT bit in the KEY command indicates which encryption algorithm (AES-ECB or AES-CCM) should be used to decrypt the key. An ECB-encrypted black key can be successfully loaded only with EKT = 0 (ECB mode), and a CCM-encrypted black key can be successfully loaded only with EKT = 1 (CCM mode).

### 12.4.5 Avoiding errors when loading red and black keys

There are ways to unsuccessfully load red and black keys that do not produce error messages, so take special care when loading these keys. Note the following known ways of unsuccessfully loading red and black keys:

- If any type of black key is loaded into a key register using a LOAD command or a KEY command with ENC = 0, no error message is generated.

    - Because these commands instruct SEC to not perform any decryption when loading the key, this simply places the encrypted form of the black key in the key register. The only indication that something is wrong is that incorrect results are produced if the key is then used in an encryption or decryption operation.
    - No error message is generated if a red key or a CCM-encrypted black key is loaded with a KEY command with ENC = 1 and EKT = 0 (indicating ECB mode), but the wrong value is placed in the key register because the key is decrypted using the wrong mode. Again, the only indication that something is amiss are incorrect results.
- If a red key or an ECB-encrypted black key is loaded using a KEY command with ENC = 1 and EKT = 1 (indicating CCM mode), an error is generated because the CCM-mode ICV check fails.
- An error is also generated if a CCM-mode black key is decrypted by the wrong security domain because the TZ/SDID modification to the JDKEK or TDKEK causes the CCM-mode ICV check to fail. If an ECB-mode encrypted black key is decrypted by the wrong security domain, no error is generated, but the key value is incorrect, that is, effectively still encrypted.

## 12.4.6 Encapsulating and decapsulating black keys

SEC's key-protection policy imposes restrictions on creating black keys and converting between black key types. When loading a red or black key into a Key Register, it is possible to prohibit the key from being written back out to memory at all. Executing a KEY command with NWB = 1 prohibits writing the key out, whereas NWB = 0 permits the key to be stored to memory as a black key. If a red key is loaded into a key register, it can be stored as either an ECB or CCM-encrypted black key (assuming NWB = 0). But if a black key is loaded into a key register, it can be stored out only as the same type of black key as was loaded, as shown in this figure.

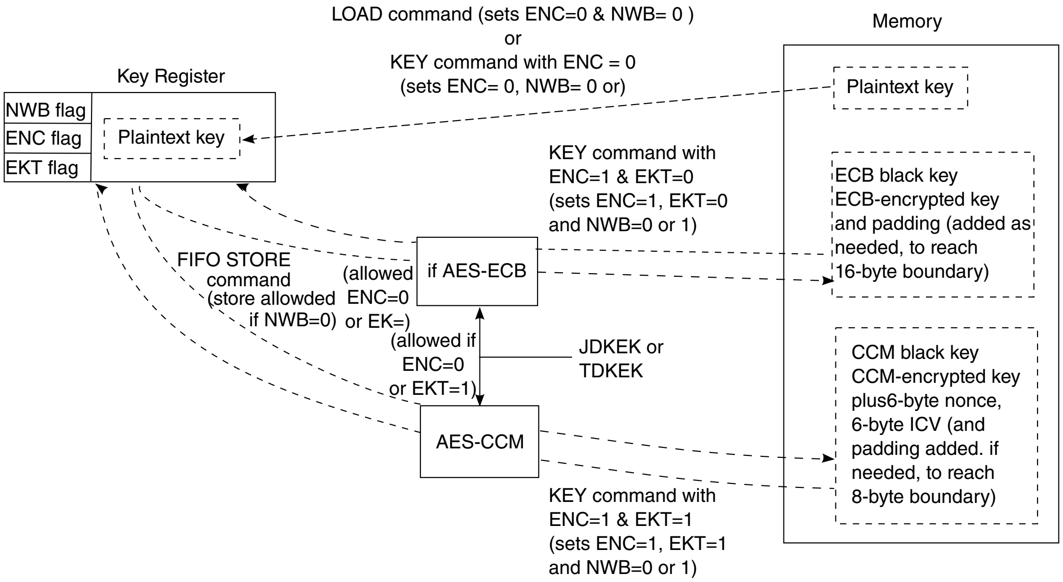


**Figure 12-1. Encapsulating and decapsulating SEC black keys**

The cryptographic key used to encrypt or decrypt black keys is held in the 256-bit JDKEKR or TDKEKR, and this key's value is modified via the appropriate TZ/SDID value before being used in a black key operation on behalf of some descriptor. The TZ/SDID value is taken from the job ring or Queue Manager Interface SDID register, depending upon where the descriptor is executed. Job descriptors or their shared descriptors always use the JDKEKR key, but trusted descriptors, or shared descriptors referenced by trusted descriptors, can use either the JDKEKR key or the TDKEKR key. Use of the TDKEKR allows trusted descriptors to encapsulate keys that cannot be decrypted by job descriptors.

The black keys used by each SDID value are encrypted using a different modification of the JDKEK or TDKEK in order to provide cryptographic separation between the keys used in different security domains. Note that TrustZone trusted descriptors always use TZ = 1b, SDID = 000h for the JDKEK/TDKEK modification, regardless of the job ring in which the descriptor is executed. All QI queues use the same SDID value, so all queues can share black keys.

Because black keys are not intended for storage of keys across chip power cycles (SEC's blob mechanism (section Blobs) is intended for this purpose), the values in the JDKEKR and TDKEKR are not preserved at chip power-down. Instead, new 256-bit secret values are loaded into the JDKEKR and TDKEKR from the RNG following power-on for use during the current power-on session. That means that a black key created during one power-on session cannot be decrypted on subsequent power-on sessions.

### 12.4.7 Types of black keys and their use

The four types of black keys that SEC's black-key mechanism implements are listed in this table.

Table 12-2. Black key types

| Black key type | Key used | Encryption mode |
|---|---|---|
| JDKEK-ECB | JDKEKR | AES-ECB |
| TDKEK-ECB | TDKEKR | |
| JDKEK-CCM | JDKEKR | AES-CCM |
| TDKEK-CCM | TDKEKR | |

Note that it is possible to inadvertently load a black key as the wrong type, resulting in an incorrect key value in the key register. No error message is generated when any of the black key types listed in this table are loaded in ECB mode. But an ICV check failure error message is generated if the wrong black key type (or a red key) is loaded in CCM mode.

It is possible to load a JDKEK-encrypted black key and save it out as a TDKEK-encrypted black key, or vice versa. This is permitted because only trusted descriptors have access to TDKEK encryption, and they are trusted to operate only in a secure manner. Such conversions might be used during a key provisioning procedure. (But as noted earlier, conversion between ECB-black keys and CCM-black keys is not permitted.)

### 12.4.8 Types of blobs for key storage

As described in Blobs, SEC implements different types of blobs that are intended for storage of keys across power cycles. Because encapsulation or decapsulation of blobs takes longer than encapsulation and decapsulation of black keys, if a long-term key is stored in a blob and must be used multiple times during a power-on session, for performance reasons it is preferable to decapsulate the blob at power-up and re-encapsulate the key as a black key.

SEC implements operations that convert between blob encapsulation and black-key encapsulation without exposing the key in plaintext. There are several different blob types dedicated to key storage that correspond to the different types of black keys. A specific type of black key converts into a specific type of black-key blob. If this were not enforced by SEC, a hacker could attempt to convert one black key type to another black key type by first exporting the black key as a black key blob, and then re-importing the blob as if it were a different type of black key blob. But because each blob type uses a different derivation for the blob key encryption key, such an attempt at misrepresenting the blob type fails with a MAC-tag error when the blob is decapsulated.

## 12.5 Trusted descriptors

Trusted descriptors provide a means for trustworthy software to create trusted "applets" that can be safely executed by less trustworthy software.

### 12.5.1 Why trusted descriptors are needed

Software utilizes the cryptographic features of SEC by building a descriptor, and then adding this descriptor to a job ring. Usually the same software entity performs both operations, that is, building the descriptor and adding it to a job ring. But there are cases in which different software entities perform the two operations. One important case is when the descriptor builder is more trustworthy than the job ring owner. For example, the boot software or TrustZone SecureWorld software might be trusted to properly handle particularly sensitive data, such as digital-rights management keys, but the content-rendering software that needs to use those keys may not be as trustworthy.

SEC implements a trusted descriptor mechanism to be used in these cases. These trusted descriptors are granted special privileges that ordinary job descriptors are not, and to ensure that these special privileges are not abused by tampering with the trusted descriptor, SEC ensures the integrity of the trusted descriptor with a cryptographic signature.

### 12.5.2 Trusted-descriptor key types and uses

When SEC is in trusted mode or secure mode, the hardware (Special-purpose cryptographic keys) allows SEC to use the trusted-descriptor key encryption key and the trusted-descriptor signing key. These keys are available only to SEC and cannot be read or written. (For testing purposes, these registers, but not the trusted mode or secure mode values of these keys, can be read and written in non-secure mode.) Furthermore, these keys can be used only for key encryption/decryption or signing/signature verification; users cannot use them for anything else. In addition, these keys are changed every boot cycle so that any keys encrypted with the trusted-descriptor key encryption key are lost when the system is rebooted. Likewise, following a reboot, any trusted descriptors signed (HMAC'd) during the previous power-on cycle fail the integrity check and do not execute.

### 12.5.3 Trusted descriptors encrypting/decrypting black keys

SEC implements both trusted and normal (non-trusted) black keys, which are encrypted with different key-encryption keys. Both trusted and normal descriptors are allowed to encrypt or decrypt normal black keys, but only trusted descriptors are allowed to encrypt or decrypt trusted black keys. Note that if any black keys are included as immediate data within the trusted descriptor, it is the encrypted version of the key that is verified when computing the signature. When executing the trusted descriptor, the black key is not decrypted unless the signature is valid.

Trusted software can decapsulate master secrets from trusted-descriptor blobs and can use these master secrets to derive keys that it embeds as trusted black keys within trusted descriptors. Untrusted software can then cause SEC to execute these trusted descriptors to encrypt or decrypt data, without the master secrets or derived keys ever being directly accessible to the untrusted software.

In addition, trusted descriptors can be written to ensure that these keys cannot be misused. This mechanism would be useful in certain IKE key exchange processes, or for supporting trusted-computing group, trusted-platform module operations, or various data rights-management standards.

See Black keys for more information.

### 12.5.4 Trusted-descriptor blob types and uses

SEC implements both trusted-descriptor blobs and normal (non-trusted descriptor) blobs, which use different key derivations for the blob-key encryption keys. Both trusted and normal descriptors are allowed to encapsulate or decapsulate normal blobs, but only trusted descriptors are allowed to encapsulate or decapsulate trusted blobs. When executing the trusted descriptor, the blob is not decapsulated unless the integrity check is valid.

See Blobs for more information.

### 12.5.5 Configuring the system to create trusted descriptors properly

**NOTE**

Trusted descriptors use the descriptor commands defined in Using descriptor commands. The SIGNATURE command (SIGNATURE command) is used only by trusted descriptors.

Although trusted descriptors cannot be forged or altered in unauthorized ways after they are generated and signed, to be truly considered "trusted," the system must be configured so that trusted descriptors can be created only by trusted software. Trusted descriptors can be created only via a job ring that has the Allow Make Trusted Descriptor (AMTD) bit set in the job ring's JRaICID register. Proper configuration is required to ensure that only trusted software can write to any JRaICID register (because this would allow the AMTD bit to be set). This can be ensured in any of the following ways:

- The register is written and then locked (via its LAMTD bit) by trusted boot software.
- The system uses the operating system or hypervisor to control access to the address block that includes the JRaICID registers.

Proper configuration for the use of trusted descriptors must also ensure control of access to the trusted-descriptor-creation job rings, that is, those job rings whose JRaICID registers have been configured with the AMTD bit set. The operating system or hypervisor can provide access control by granting certain processes access to the register address block containing a particular job ring's control registers, and denying access to that block to other processes.

### 12.5.6 Creating trusted descriptors

To create a trusted descriptor, trustworthy software builds a candidate trusted descriptor that uses the extra privileges properly. For example, the trusted descriptor might utilize cryptographic keys that an ordinary job descriptor cannot access, but the trusted descriptor would be designed so that the key values cannot be exposed.

The candidate trusted-descriptor is converted to a trusted descriptor by executing the candidate trusted-descriptor in a specially-privileged job ring. This causes SEC to cryptographically sign the descriptor. The trusted descriptor can later be executed by less trustworthy software. When the trusted descriptor is executed, SEC executes the commands within the trusted descriptor only if the signature is correct. This ensures that the trusted descriptor has not been tampered with after it was created.

### 12.5.6.1 Trusted descriptors and descriptor-header bits

Descriptor headers contain a 2-bit field related to trusted descriptors. See HEADER command for a full explanation of the descriptor header.

The 00 value in the TDES field indicates an ordinary job descriptor. The 11 value indicates a candidate trusted descriptor, that is, a descriptor that SEC should convert into a trusted descriptor by affixing a signature. When SEC executes a candidate trusted descriptor, it checks to see if the AMTD (allow make trusted descriptor) bit is set in the job ring's JRaICID register. If not, the candidate trusted descriptor is not converted to a trusted descriptor and the job terminates with an error. If AMTD=1, SEC changes the TDES field value to 10 if the candidate trusted descriptor is being created in a job ring owned by TrustZone nonSecureWorld, but changes the TDES field to 01 if the job ring is owned by TrustZone SecureWorld. SEC then either affixes a signature to the new trusted descriptor, or executes the trusted descriptor, or both, depending upon the option in the SIGNATURE command at the end of the descriptor.

### 12.5.6.2 Trusted-descriptor execution considerations

Important rules of use and things to consider when executing trusted descriptors are as follows:

- When a trusted descriptor is executed, SEC first checks the signature (HMAC) to verify that the trusted descriptor has not been modified. If the trusted descriptor references a shared descriptor, it is included in the computation of the signature. If the signature is valid, the trusted descriptor is executed. If the signature is invalid, the job is aborted with an error indication.
- A TrustZone non-SecureWorld trusted descriptor can be executed only within a job ring that has the same SDID value as the job ring in which the trusted descriptor was created. The reason for this restriction is that job rings may be owned by different security domains that do not trust each other's TrustZone non-SecureWorld trusted descriptors. This restriction is enforced by including the SDID of the job ring's JRaICID register in the signature computation, both when creating the trusted descriptor and before executing the trusted descriptor.
- A TrustZone SecureWorld trusted descriptor can be executed within any job ring, regardless of that job ring's SDID or TZ value. TrustZone SecureWorld trusted descriptors can be created only in a job ring owned by TrustZone SecureWorld. This allows TrustZone SecureWorld to create special trusted descriptors that are trusted by all security domains. The TDES field in the trusted descriptor's HEADER command is used to distinguish SecureWorld trusted descriptors from non-SecureWorld trusted descriptors.

- If a trusted descriptor contains a jump to another descriptor, it must also be trusted. Jumping from a trusted descriptor to a job descriptor results in an error and processing stops. Because all CHAs, all MODEs, and the Class 2 Key and Key Size Registers are reset before a trusted descriptor's signature is checked, care must be taken when transferring to a trusted descriptor from another descriptor (whether trusted or not) via Non-Local JUMP (see Section JUMP (HALT) command), In-Line Descriptor (see INL field in Table 7-97), or Replacement Job Descriptor (see RJD field in Table 7-97).
- Note that although address pointers within a trusted descriptor are protected against modification, any data referenced by an address pointer is not protected against modification. Therefore, keys and other information that must be protected against modification should be contained as immediate data within the trusted descriptor. When a trusted descriptor executes, it is permitted to modify itself just like a non-trusted descriptor can. This ability can be useful if the trusted descriptor is maintaining an integrity-protected value that changes, such as a usage count, sequence number, and so on. Because modifying the trusted descriptor renders the signature invalid, the signature must be recomputed after the modification. This can be accomplished by placing a SIGNATURE Command at the end of the trusted descriptor. This directs SEC to recompute the trusted descriptor's signature.

## 12.6 Blobs

SEC can protect data in a cryptographic data structure called a blob, which provides both confidentiality and integrity protection.

### 12.6.1 Blob protocol

SEC's built-in blob protocol provides a method for protecting user-defined data across system power cycles. The data to be protected is encrypted so that it can be safely placed into non-volatile storage before the chip is powered down. Each time that the blob protocol is used to protect data, a different randomly generated key is used to encrypt the data. This random key is itself encrypted using a key encryption key and the resulting encrypted key is then stored along with the encrypted data. The key-encryption key is derived from the chip's master secret key so the key-encryption key can be recreated when the chip powers up again. The combination of encrypted key and encrypted data is called a blob.

Table 7-59 shows the format of the PROTINFO field for the blob protocol, and Table 7-60 describes the bit values.

### 12.6.2 Why blobs are needed

To retain data across power cycles, the data must be stored in non-volatile memory. But data stored in this manner is potentially vulnerable to disclosure or modification when the SoC's software and hardware security-mechanisms are not functioning, for example, during debug operations. SEC is able to protect data for long term storage by encrypting that data using a secure non-volatile key. [1] Using a unique non-volatile key for each device prevents data encrypted on one device from being copied and decrypted on a different device, which might compromise the secrecy of the data.

### 12.6.3 Blob conformance considerations

Generation of private blobs is not considered in any governmental security specification. However, there are several steps in the process that can be viewed as having approved methods. These methods were chosen to conform to the following specifications, (except where noted).

- FIPS PUB 197, *Advanced Encryption Standard (AES)*, November 26, 2001.
- FIPS PUB 180-2, SECURE HASH STANDARD, August 1, 2002.
- SP800-90A, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, January 2012. Draft SP800-90B, *Recommendation of the Entropy Sources Used for Random Bit Generation*, August 2012.
- SP800-38c, *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*, May 2004.
- SP800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, March, 2007.
- SP800-57, Recommendation for Key Management - Part 1: General, March, 2007.

In the context of SEC, a blob is encrypted data that is bound to a specific device by virtue of using a secret non-volatile, device-specific master key. This master key is used only for the purpose of creating and extracting blob data, and the value of this key cannot itself be extracted from a device. To protect data requiring high-security strength, blob creation is performed in hardware using 256-bit security strength. AES-256 is used as the encryption algorithm. SHA-256 is used for key derivation (SP800-57 specifies that SHA-256 has 256-bit security strength when used in key derivation).

[1] The data is actually encrypted with a randomly generated blob key, and it is that blob key that is encrypted using the secure non-volatile key

The random number generator is specified in SP800-90A, using the Hash_DRBG with SHA-256 as the hash function. It gets entropy from a live entropy source intended to comply with SP800-90B. The random number generator has a security strength of 256 bits.

SEC blobs provide both confidentiality and integrity protection for the encapsulated data. Because a blob protects both confidentiality and integrity, it may be stored in external long-term storage such as flash. Counter with cipher block chaining-message authentication code (AES-CCM) is used as the bulk encryption algorithm. Note that the MAC associated with a blob provides integrity protection not only for the encrypted data the blob contains, but also for all intermediate keys used in the creation of a blob.

There may be many different blobs existing at the same time, used for many different purposes, and subject to different security policies. To guarantee that blobs are not inadvertently or intentionally swapped, SEC encrypts different blobs with different keys. Two mechanisms are used to guarantee that a single key is not used to encrypt unrelated data and to ensure that each key is used to encrypt as little data as possible. One of these mechanisms is random-key generation. Each time that a blob is created, SEC generates a different, random 256-bit key using SEC's internal hardware random-number generator (RNG). This blob key is used to encrypt the blob data using AES-CCM, which provides both confidentiality and integrity protection. The second mechanism is key derivation, using a device-unique, non-volatile master key as the key-derivation key. The (volatile) random blob key is encrypted with the non-volatile key derived from the master key, and then stored with the blob so that the blob data can be decrypted during subsequent power-on cycles. Different types of blobs are encrypted using different keys derived from the master key. The derived keys are further differentiated by a key modifier supplied by software, which can be used to guarantee that one blob cannot be inadvertently or maliciously substituted for another blob. Software can use these key modifiers to differentiate specific data, or to prevent replay attacks (the replacement of the current blob with an out-of-date version of the blob).

The master key is used in a key derivation function (KDF) similar to that specified in SP800-56 (sec. 5.8.1). That function includes two parties U and V, who both add information for use in deriving their shared key. Here the derived key is used for storage, and so there is only a single involved party, and hence only one block of public and private information. In the current key-derivation function, only a single iteration is required, because the size of the derived key is the same as the hash function used. Therefore, the counter is not used. The master key is concatenated with a key modifier (which may be a public or private nonce), an AlgorithmID (the blob type) and a security state indicator (that is, non-secure, secure or trusted). This message is then hashed with SHA-256, and the output is used as a blob-key encryption key.

AES-CCM mode uses a nonce and initial counter value as inputs, along with the key and data. SP800-38c requires that the nonce and counter values be unique across all invocations of AES-CCM under a given key. This requirement is met by virtue of a random key being generated for each blob. Because a key is never used more than once, there are no requirements on the nonce and initial counter value. Therefore, both nonce and initial counter value are fully specified, and the same values are used for all blobs. Blob creation uses the formatting function specified in SP800-38c, Appendix A.

The entire 16-byte MAC is stored along with the encrypted data, to provide a strong assurance of integrity. Note that due to the design of the blobs, the MAC provides integrity protection for the data, blob key and blob-key encryption key.

### 12.6.4 Encapsulating and decapsulating blobs

When encapsulating a blob, SEC:

1. Obtains a random blob key (BK) value from the RNG
2. Encrypts the data with that BK
3. Derives a blob-key encryption key (BKEK) from the master key
4. Encrypts the BK using that BKEK

When decapsulating a blob, SEC:

1. Derives a BKEK from the master key
2. Decrypts the BK using that BKEK
3. Decrypts the data with the BK

### 12.6.5 Blob types

SEC supports different types of blobs, and a coded value of the blob type is used as an input to the key-derivation function. This prevents a blob that was exported as one type from being imported as another type because it would decrypt improperly and so would fail the MAC tag check. This table lists the types of blobs that SEC supports. Note that the type categories are orthogonal, that is, a blob has one type from each type category. For instance, one blob may be a (normal format/black key/secure state blob), while another blob may be a (test format/general data/trusted state) blob. In addition, black key blobs are differentiated by encryption mode and encryption key, so one black key blob may be a (AES-ECB/TDKEK) type and another black key blob may be a (AES-CCM/JDKEK) type.

Table 12-3. Blob types

| Type Category | Type | Cross-reference |
|---|---|---|
| Formats | Normal format | Blob types differentiated by format |
| | Test format | |
| | Master key verification format | |
| Contents | General data (that is, red blobs) | Blob types differentiated by content |
| | Black keys (that is, black blobs)<br>• Encryption modes: AES-ECB and AES-CCM<br>• Encryption keys: JDKEK and TDKEK | |
| Security states | Trusted state | Blob types differentiated by security state |
| | Secure state | |
| | Non-secure state | |

#### 12.6.5.1 Blob types differentiated by format

SEC supports three different formats for blobs, usable for all blob content types, and all blob security state types. This figure describes the blob formats and how they work.

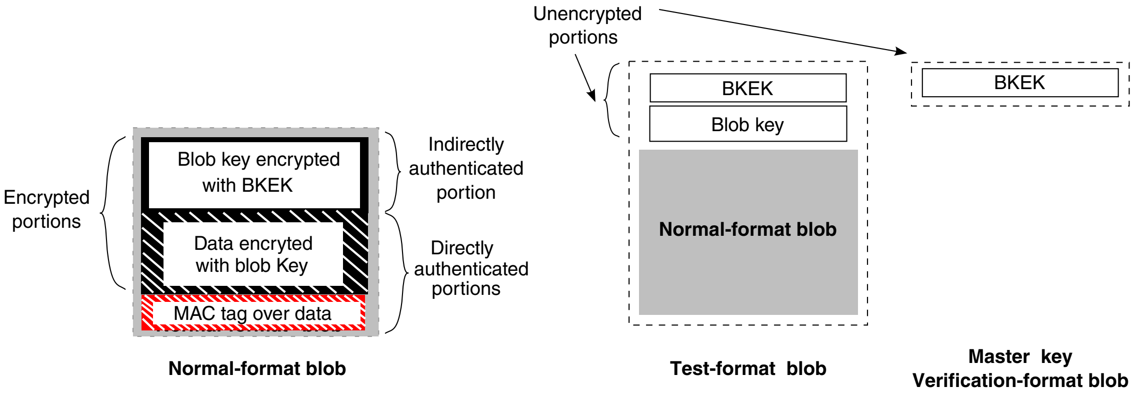

Unencrypted portions
BKEK
Blob key
Normal-format blob
BKEK
Encrypted portions
Blob key encrypted with BKEK
Data encryted with blob Key
MAC tag over data
Indirectly authenticated portion
Directly authenticated portions
Normal-format blob
Test-format blob
Master key Verification-format blob

Figure 12-2. Formats of SEC blobs

- A normal-format blob consists of the encrypted blob key, the encrypted data, and a message authentication code (MAC) tag, as shown on the left side of the figure. A randomly-generated, 256-bit blob key is used to encrypt the data using the AES-CCM cryptographic algorithm. AES-CCM encrypts the data and also yields a MAC tag that is used to protect the data's integrity. The blob key itself is encrypted in AES-ECB mode using a 256-bit blob-key encryption key (BKEK). Checking the MAC directly authenticates the data encapsulated in the blob. The blob key is indirectly authenticated because substitution or corruption of the encrypted blob key yields an incorrect plaintext blob key, which causes the blob content to be decrypted incorrectly, which is detected by the MAC check. Because a normal-format blob is used to protect actual data, the blob-key encryption key (BKEK) that is used to encrypt the blob key for a normal format blob is secret, by virtue of having been derived from the secret master key.

- As shown in the middle of the figure, a test-format blob consists of a normal-format blob, with the unencrypted BKEK and unencrypted blob key prepended. Because the purpose of a test-format blob is to facilitate testing blob encapsulation and decapsulation, the BKEK for a test-format blob is derived from a known test key. SEC permits test-format blobs to be encapsulated or decapsulated only when SEC is in non-secure mode.
- As shown on the right side of the figure, a master key verification format blob consists of only the unencrypted BKEK. Because the purpose of a master key verification format blob is to verify that the master key has been properly programmed, the BKEK for a master key verification format blob is derived from the secret master key. In order to ensure the secrecy of BKEKs used for normal format blobs, the derivation is different from the derivation used for normal format blobs. This ensures that the BKEKs used to protect data cannot be exposed by examining the BKEK values in master key verification format blobs.

### 12.6.5.2 Blob types differentiated by content

One of the blob content types is intended for general data (see Red blobs (for general data)), and four content types are intended for cryptographic keys (see Black blobs (for cryptographic keys)).

#### 12.6.5.2.1 Red blobs (for general data)

Unencrypted data that should be protected is sometimes referred to as "red data", so the type of blob intended for general data (which is left unencrypted when the blob is decapsulated) is called a red blob. When SEC is instructed to encapsulate data as a red blob, it assumes that the data to be encapsulated is unencrypted and it proceeds to encrypt the data with the blob key. Likewise, when SEC is instructed to decapsulate a red blob, it assumes that the data that is decapsulated is to be left in memory unencrypted. Other mechanisms, such as an operating system or hypervisor acting in conjunction with a memory management unit, may be used to protect the data before it is encapsulated into a blob and after it is decapsulated from a blob.

#### 12.6.5.2.2 Black blobs (for cryptographic keys)

SEC's black blob mechanism is a means for translating between black key encapsulation and blob encapsulation without exposing the key during the translation process. A black blob is simply a blob whose input during blob encapsulation is assumed to be a black key, and whose output during blob decapsulation is either a black key that is written into memory, or an unencrypted key that is placed directly into a Key Register.

SEC supports the protection of cryptographic session keys by encrypting these keys in a "black key" encapsulation format when storing them in memory via a FIFO STORE command, and then decapsulating them "on-the-fly" as they are referenced by a job descriptor with a descriptor KEY command. Black key encapsulation or decapsulation is very quick, but black keys are intended only for protection during the current SoC power-on session. Black keys encapsulated during one chip power-on session cannot be decapsulated on subsequent power-on sessions because the key encryption key (JDKEK or TDKEK) is erased during power-down and is replaced by a new randomly-generated key encryption key at power-up. To protect a key so that it can be recovered on subsequent power cycles, the key must be encapsulated as a blob. A key could be encapsulated as a red blob, but this would require exposing the key in memory in unencrypted form. To avoid exposing keys in unencrypted form, SEC supports the concept of black blobs. (Data that is not sensitive to disclosure, either because it is inherently nonsensitive or because it always remains encrypted, is sometimes referred to as "black data".)

#### 12.6.5.2.3 Enforcing blob content type

When SEC is instructed to encapsulate a black blob, it first decapsulates the black key that was specified as input and then encapsulates the resulting key as a Black blob. The black blob itself is exactly the same as a red blob, except that the BKEK derivation is different from red blobs. This prevents a black blob from being decapsulated as a red blob, which would leave the key exposed in memory. Because black keys can be encrypted under either the JDKEK or the TDKEK, and can be encrypted in either AES-ECB mode or AES-CCM mode, SEC first decrypts the black key data with the appropriate KEK using the appropriate mode and then re-encrypts the key data with the BK using AES-CCM. During this process the key that is temporarily unencrypted is safely protected within SEC's hardware storage. To prevent mixing up the different types of black blobs (JDKEK vs. TDKEK and ECB vs. CCM), the BKEK for each type is derived differently.

### 12.6.5.3 Blob types differentiated by security state

SEC also supports different types of blobs for use in different security states. All of the blob-format types and blob-content types are available in each of the following different security states:

- Trusted state
- Secure state
- Non-secure state

However, the BKEKs for the blobs are derived differently for each of these states. Therefore, a blob encapsulated while operating in a particular state cannot be decapsulated while SEC is operating in another of these states:

- During trusted and secure states, the BKEK is derived from the secret master key (but using different key derivation functions in the two states).
- While SEC is operating in non-secure state, the BKEK is derived from the known test key. This latter type of blob is intended to facilitate testing using known-answer tests.

## 12.6.6 Blob encapsulation

A data blob is encrypted using a blob key (BK), which is a random number used as an AES-CCM key. The NIST AES-CCM specification states that for any key, all invocations must use distinct nonces and counter blocks. Although SEC uses the same nonce and initial counter block values for all data blobs, SEC satisfies the AES-CCM requirement because each encryption operation uses a different key (that is, a random number generated by the RNG). The nonce is given as all zeros, and so the initial block B0 = 3B00_0000_0000_0000_0000_0000_0000_*xxxx*h, where *xxxx* is the number of bytes of plaintext (maximum length is 65535 bytes), while the initial counter value Ctr0= 0300_0000_0000_0000_0000_0000_0000_0000h. These values are automatically generated during the encapsulation operation.

Figure 12-3 shows the entire blob-encryption operation. B0 is generated internally and stored in the Class 1 Context DWords 0 and 1, while Ctr0, also generated internally, is stored in Class 1 Context DWords 2 and 3 (see Table 11-97). The random BK value is stored in the Class 1 Key Register, and the operation mode is set to AES-CCM.

At the blob pointer, the first 32 bytes contain the key blob, which is the encrypted value of the random blob key. Output ciphertext data (data blob) is stored at the blob pointer + 32. The generated message-authentication code (MAC, the signature over the data blob) is stored in the final 16 bytes of the blob.

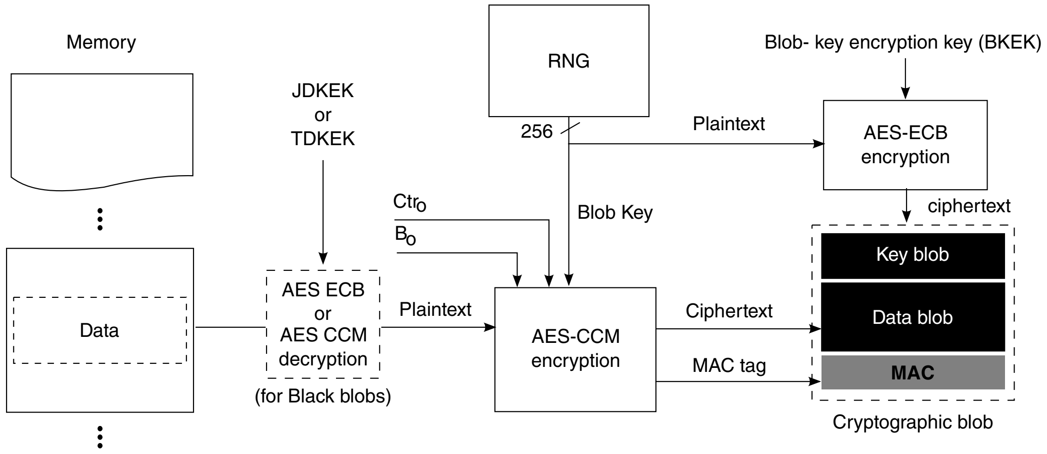


Figure 12-3. Encapsulating a blob

## 12.6.7 Blob decapsulation

Before decrypting a data blob, the associated key blob must be decrypted to obtain the blob key. The key blob resides at the blob pointer. AES-ECB mode is used to decrypt the key blob using the BKEK. Generation of the BKEK for blobs is described below.

Ctr0 and B0 are generated internally, and are stored in the Class 1 Context 1 and Context 2 registers, respectively (see Figure 12-4). AES-CCM mode is used to decrypt the data blob (starting at the blob pointer + 32), using the decrypted blob key.

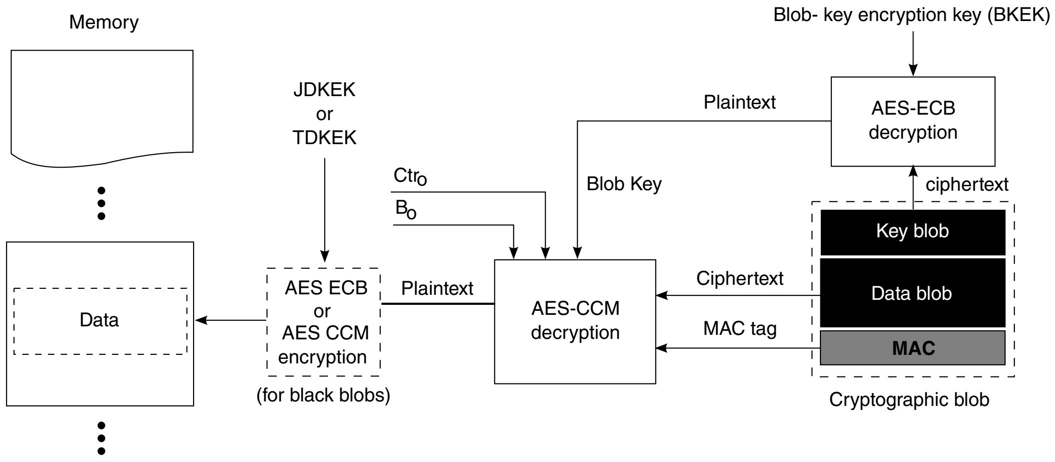


Figure 12-4. Decapsulating a blob

## 12.7 Critical security parameters

SEC contains several encryption and authentication keys that are identified as being critical security parameters (CSPs), as defined in FIPS140-2. Each of these CSPs are zeroized (cleared) upon the SecMon state machine entering the FAIL state. This FAIL state indicator is an input to SEC and can be observed via the SEC Status Register.

Upon receiving an indication that the security state machine has entered the FAIL state, all register-based CSPs are zeroized via the asynchronous hardware reset. SEC can be restarted after the chip has transitioned from FAIL to non-secure state; however, all critical security parameters are lost forever.

This table lists the critical security parameters included in SEC.

**Table 12-4. Critical security parameters**

| CSP | Notes | Related cross-reference |
|---|---|---|
| Zeroizable master key | Inside of security power island; loaded and locked once at provisioning time | — |
| CCB Class 1 Key Register | — | See register appendix |
| CCB Class 2 Key Register | — | See register appendix |
| PKHA E register | Exponent | See register appendix |
| Trusted descriptor signing key | Loaded at boot time from RNG | Trusted descriptors |
| Trusted descriptor key encryption key | | |
| Job descriptor key encryption key | Loaded at boot time from RNG | Keys available in different security modes |
| SNOW f8 internal state | — | SNOW 3G f8 accelerator functionality |
| SNOW f9 internal state | — | SNOW 3G f9 accelerator functionality |
| ZUCE internal state | — | ZUC encryption accelerator (ZUCE) functionality |
| ZUCA internal state | — | ZUC authentication accelerator (ZUCA) functionality |
| Crypto-engine internal datapath registers | — | See register appendix |
| Output data FIFO | — | — |

## 12.8 Manufacturing-protection chip-authentication process

The manufacturing-protection authentication process is used to authenticate the chip to the OEM's server. This authentication process can ensure that the chip:

- Is a genuine NXP part

- Is the correct part type
- Has been properly configured by means of fuses
- Is running authenticated OEM software
- Is currently in the secure or trusted mode

These are inputs to a key derivation function, to create a private ECC key that is available only to the crypto hardware. The public ECC key can be generated and used to later authenticate the chip and verify the security status of the chip. These properties are verified by digitally signing a message using this private ECC key. The message may be verified by a server using the public ECC key. Because only a genuine NXP part, configured correctly, and running in the proper security state can correctly sign the message, assurance of all of this is provided by the verification of the message signature. The message cannot be spoofed by untrustworthy software, because the private-key generation, public-key generation and signature functions are all implemented in hardware, and the chip-specific data is supplied by secure-boot firmware. After the signature over the message has been verified, the server can be assured that it is safe to download proprietary data to the chip over a secured connection.

The authentication process takes place in three stages, implemented via three functions built into the SEC hardware.

**Table 12-5. Manufacturing-protection chip-authentication functions**

| Function name | Abbreviation | Authentication steps implemented by function | Cross-reference |
|---|---|---|---|
| Manufacturing-protection private-key generation function | MPPrivK | • Takes input data to be authenticated and hashes that data with a secret value embedded in the silicon. The result is an ECDSA private key that is securely stored in the MPPKR. | MPPrivK-generation function |
| Manufacturing-protection public-key generation function | MPPubK | • Generates an ECDSA public key that matches the private key in MPPKR and outputs that public key.[1] | MPPubK-generation function |
| Manufacturing-protection sign function | MPSign | • Takes the value in the MPMsg register and concatenates any additional data supplied as ordinary input to the MPSign function<br>• Signs the concatenated message data using the private key that was stored in the MPPKR by | MPSign function |

**Table 12-5. Manufacturing-protection chip-authentication functions**

| Function name | Abbreviation | Authentication steps implemented by function | Cross-reference |
|---|---|---|---|
| | | the MPPrivK Generation function<br>• Outputs the signed message, along with the message representative. Software running on the chip sends this signed message to the OEM's server, which then verifies the signature by means of the public key output earlier by the MPPubK Generation function. | |

1. The MPPubK-generation function is run once on a single chip at the OEM's facility, and the OEM's server retains a copy of this public key to be used later in the authentication process.

## 12.8.1 Providing data to the manufacturing-protection authentication process

The purpose of the manufacturing-protection authentication process is to authenticate certain information, such as the chip's part number, serial number, and the super root key hash, by signing it with a private key that can be used only in a legitimate NXP chip of the correct type running in the secure or trusted states.

The following sections describe how data is input to the manufacturing-protection process.

### 12.8.1.1 Providing data to the MPPrivK-generation function

The MPPrivK-generation function is expected to be run only by the secure boot firmware.

#### 12.8.1.2 Providing data to the MPPubK-generation function

The only inputs to the MPPubK-generation function are the manufacturing protection private key from the MPPKR, and the elliptic curve selection from the CSEL field in the PDB. The hardware guarantees the correctness of both of these inputs because the MPPKR is accessible only to hardware, and the value in the CSEL field must match the value used in the MPPrivK-generation function.

#### 12.8.1.3 Providing data to the MPSign function

To provide data to the MPSign function, the secure boot firmware writes some or all of the data into the MPMR, then locks it by setting the MPMRL bit in the Security Configuration Register. Additional data can be provided as ordinary message input to the MPSign function. This additional data will be appended to the content of the MPMR before the data is hashed and signed. All this data is authenticated as having originated from a legitimate NXP chip of a specific type, because the data is signed with the manufacturing-protection private key when the MPSign function is invoked. But only the portion of the data that was written into the MPMR is guaranteed to have originated from trusted firmware. Since the MPSign function is intended to be invoked by software that has not yet been authenticated, the extra data supplied as ordinary message input to the MPSign function should be treated skeptically until the authentication process is complete.

#### 12.8.1.4 Role of the ROM-resident secure boot firmware

Because the ROM-resident secure boot firmware is the only software that is known to be trusted prior to authentication, it plays a crucial role in the manufacturing-protection authentication process. The ROM-resident boot firmware reads fuse-resident data that needs to be authenticated and either supplies some or all of it as data to the MPPrivK generation function or writes some or all of the data to the MPMR. Note that all of the data needed to authenticate the software image that will be booted must be supplied by the ROM-resident firmware using either the MPPrivK generation function or the MPMR.

The MPSign function is intended to be invoked by untrusted software that has just booted, which is why the data to be authenticated via the MPSign function must be supplied in advance by trusted ROM-resident secure boot software and then securely conveyed to the MPSign function via the MPMR. After the operating system has booted and is able to run a network-protocol stack, application software establishes a communication session with the OEM's server. The application software then runs a descriptor that invokes the MPSign protocol, which uses the ECDSA private key stored in MPPKR to sign a message composed of the content of MPMR followed by other optional data. Note that this additional optional data is supplied by potential untrustworthy software, so it can be relied on only if data authenticated via the MPPrivK-generation function or via the MPMR has demonstrated that the software that supplied the data was properly authenticated via the super root key hash.

## 12.8.2 MPPrivK-generation function

The MPPrivK-generation function uses supplied input data together with a secret value embedded in the silicon to generate an elliptic-curve DSA private key. The function stores the private key in the MPPKR and then the MPPKR is locked to prevent reading or writing from the register bus. The private key is later used in the MPPubK-generation function and the MPSign function. Note that an error is generated if the MPPrivK Generation function is run a second time in the same power-on session.

### 12.8.2.1 Differences between the MPPrivK-generation function and the DL KEY PAIR GEN function

The MPPrivK generation function is a specialized version of the DL KEY PAIR GEN function. The following list summarizes the key differences between the two functions.

- The MPPrivK generation function generates only ECDSA private keys, not DSA keypairs.
- The MPPrivK generation function generates the private key by applying a key generation function to the input message data and a secret value embedded in the silicon. The secret value is different in each chip type. The DL KEY PAIR GEN function cannot use the secret value embedded in the silicon.
- The MPPrivK generation function uses predefined ECC curves embedded in hardware. The choice of curve is specified by the CSEL field in the PDB. The DL KEY PAIR GEN function uses curve parameters supplied via the PDB.
- The MPPrivK generation function keeps the private key secret by storing it in the MPPrivK register. The DL KEY PAIR GEN function outputs the private key to memory (along with the public key).

### 12.8.2.2 MPPrivK-generation function parameters and operation

This table describes the MPPrivK-generation parameters.

Table 12-6. MPPrivK-generation function parameters

| Parameter | Source/Destination | Length | Definition |
|---|---|---|---|
| *q* | Built-in | L | Prime number or irreducible polynomial that creates the field |
| *r* | Built-in | N | Order of the field of private keys |
| *a,b* | Built-in | 2*L | ECC curve parameters. |
| $G_{x,y}$ | Built-in | 2*L | Generator point |
| m | Input | - | The message data to be input to the private-key generator function |
| *s* | Stored in MPPKR | N | Private key |

This table describes the inputs, outputs and operation of the MPPrivK function.

Table 12-7. MPPrivK-generation function inputs, outputs, and operation

| Property | Value |
|---|---|
| Inputs | • Message data to be input to the private key generation function.<br>• The Csel field in the PDB, selecting a predefined ECC curve. |
| Outputs | • The manufacturing protection private key *s*, which is stored in the MPPrivK register. |
| Operation | • Generate a private key *s*, in the range 1≤s<r. (Hash the supplied message data and the built-in secret value to yield *s*. If *s*=0, alter the input to the generation function by a constant and generate a new *s*.)<br>• Store *s* in MPKeyR as the private key. |

## 12.8.2.3 Protocol data block (PDB) for the MPPrivK-generation function

This figure shows the PDB for the MPPrivK-generation function.

Table 12-8. MPPrivK-generation PDB

| | | | |
|---|---|---|---|
| SGF<br>(1 bit) | Reserved<br>(10 bits) | Csel<br>(4 bits) | Reserved<br>(17 bits) |
| Pointer to m | | | |
| Message length | | | |

This figure shows the format of the SGF field.

Table 12-9. MPPrivK-generation function PDB-format of the SGF field

| |
|---|
| 31 |

*Table continues on the next page...*

**Table 12-9. MPPrivK-generation function PDB-format of the SGF field (continued)**

| |
|---|
| ref m SGF (Scatter Gather Flag) If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct-address pointer. |

This figure shows the format of the CSEL field.

**Table 12-10. MPPrivK-generation function PDB-format of the CSEL field**

| 20-17 |
|---|
| CSEL (Curve Select)<br>• 0011 = P256<br>• 0100 = P384<br>• 0101 = P521<br>All other values are reserved. |

## 12.8.3 MPPubK-generation function

The MPPubK-generation function uses the private key value stored in the MPPrivK register by the MPPrivK-generation function to generate a matching elliptic-curve DSA public key. The curve selected via the Csel field in the PDB must match the curve used by the MPPrivK-generation function, else an error is generated. The public key created by the MPPubK-generation function is written out to the specified results destination. Note that the MPPubK Generation function is intended to be run just once, at the OEM's facility, but no harm is done if it is run at other times.

### 12.8.3.1 Differences between the MPPubK-generation function and the DL KEY PAIR GEN function

The MPPubK generation function is a specialized version of the DL KEY PAIR GEN function. The following list summarizes the key differences between the two functions.

- The MPPubK generation function generates only an ECDSA public key, not DSA or ECDSA keypairs.
- The MPPubK generation function creates a public key to match the private key value that was stored in the MPPrivK register by the MPPrivK generation function.
- The MPPubK generation function uses predefined ECC curves embedded in hardware. The choice of curve is specified by the Csel field in the PDB.

- The MPPubK generation function outputs only the public key. Unlike the DL KEY PAIR GEN funciton it does not output the private key.
- The private key stored in the MPPrivK register by the MPPrivK generation function is not altered, and remains available for use in the MPSign function.

### 12.8.3.2 MPPubK-generation function parameters and operation

This table describes the MPPubK-generation parameters.

**Table 12-11. MPPubK-generation function parameters**

| Parameter | Source/Destination | Length | Definition |
|---|---|---|---|
| *q* | Built-in | L | Prime number or irreducible polynomial that creates the field |
| *r* | Built-in | N | Order of the field of private keys |
| *a,b* | Built-in | 2*L | ECC curve parameters. |
| $G_{x,y}$ | Built-in | 2*L | Generator point |
| *s* | Read from MPPKR | N | Private key |
| $W_{x,y}$ | Output | 2*L | Public key |

This table describes the inputs, outputs and operation of the MPPubK function.

**Table 12-12. MPPubK-generation function inputs, outputs, and operation**

| Property | Value |
|---|---|
| Inputs | • The Csel field in the PDB, selecting a predefined ECC curve.<br>• The manufacturing protection private key *s*, which is read from the MPPrivK register |
| Outputs | • The manufacturing protection public key $W_{x,y}$, which is output to memory. |
| Operation | • Compute $W_{x,y} = sG_{x,y}$<br>• Output $W_{x,y}$ as the public key. |

### 12.8.3.3 Protocol data block (PDB) for the MPPubK-generation function

This figure shows the PDB for the MPPubK-generation function.

**Table 12-13. MPPubK-generation PDB**

<table>
<tr><td>SGF<br>(1 bits)</td><td>Reserved<br>(10 bits)</td><td>Csel<br>(4 bits)</td><td>Reserved<br>(17 bits)</td></tr>
<tr><td colspan="4">Pointer to Wx,y</td></tr>
<tr><td colspan="4">Message length</td></tr>
</table>

This figure shows the format of the SGF field.

Table 12-14. MPPubK-generation function PDB-format of the SGF field

| 31 |
|---|
| ref Wx,y SGF (Scatter Gather Flag) - If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct-address pointer. |

This figure shows the format of the CSEL field.

Table 12-15. MPPubK-generation function PDB-format of the CSEL field

| 20-17 |
|---|
| CSEL (Curve Select)<br>• 0011 = P256<br>• 0100 = P384<br>• 0101 = P521<br>All other values are reserved. |

#### 12.8.3.4 Running the MPPubK generation function at the OEM's facility

When a chip is first adopted by an OEM, the OEM runs the MPPubK-generation function on a sample of the chip and saves the public key of the manufacturing protection keypair on the OEM's server. Running the MPPubK-generation function at the OEM's facility this one time guarantees that the public key is authentic, that is, that it matches the private key that is used to sign messages generated by properly configured NXP chips of this type. The OEM will first have programmed the trusted root public key into fuses, and then reboot the chip. The secure boot firmware will run the MPPrivK-generation function at POR and store the manufacturing protection private key in the MPPrivK register. The MPPubK-generation function will read the manufacturing protection private key from the MPPrivK register and generate a matching public key. Later when the identically configured chips are booted within the contract manufacturing facility, the identical private key will be generated by the MPPrivK generation function. The MPPrivK-generation function stores the private key in the MPPKR for use in the MPSign function. The message signed by the MPSign function can be authenticated against the manufacturing protection public key stored on the OEM's seerver.

### 12.8.4 MPSign function

MPSign is the elliptic-curve, digital-signature algorithm (ECDSA) signing function used in the manufacturing protection authentication process. See Manufacturing-protection chip-authentication process for a discussion of this process. MPSign supports only ECDSA in prime fields. This function takes message data as input, and outputs a signature over a message composed of the content of the MPMR, followed by the input-data message.

Note that the curve specified via the Csel field in the PDB must match the curve used in the MPPrivK-generation function. This table lists the MPSign protocol parameters.

#### 12.8.4.1 MPSign function parameters and operation

This table describes the MPSign function parameters.

**Table 12-16. MPSign function parameters**

| Parameter | Source/Destination | Length | Definition |
|---|---|---|---|
| *q* | Built-in | L | Prime number or irreducible polynomial that creates the field |
| *r* | Built-in | N | Order of the field of private keys |
| *a,b* | Built-in | 2*L | ECC curve parameters |
| $G_{x,y}$ | Built-in | 2*L | Generator point |
| *s* | Read from MPPKR | N | Private key |
| m | Input | - | The message data to be signed. |
| *C* | Output | N | First part of digital signature |
| *d* | Output | N | Second part of digital signature. The buffer for *d* must be a multiple of 16 bytes, as it is used to store an encrypted intermediate result, which may include padding. |
| *mes-rep* | Output | 256 | The hash of the MPMR concatenated with m. |

This table describes the inputs, outputs and operation of the MPSign function.

**Table 12-17. MPSign function inputs, outputs, and operation**

| Property | Value |
|---|---|
| Inputs | • m, the message data to be signed<br>• u, the private key (from the MPPrivK register)<br>• a,b, the curve parameters (selected via the Csel field in the PDB) |
| Outputs | • The signature over the signed message.<br>• mes-rep, the hash of MPMR concatentated with the message data |
| Operation | • Compute $V_{x,y}$ = u $G_{x,y}$, c = $V_x$ mod r. If c=0, try again with a new *u*.<br>• Compute d = $u^{-1}$(f+sc) mod r. If d=0, try again with a new *u*.<br>• Output (*C*, *d*) as the signature. |

### 12.8.4.2 Protocol data block (PDB) MPSign function

This figure shows the MPSign function PDB.

**Table 12-18. MPSign function PDB**

| SGF<br>(4 bits) | Reserved<br>(7 bits) | Csel<br>(4 bits) | Reserved<br>(17 bits) |
|---|---|---|---|
| Pointer to m | | | |
| Pointer to mes-rep | | | |
| Pointer to C | | | |
| Pointer to d | | | |
| Message length | | | |

This figure shows the format of the SGF field.

**Table 12-19. MPSign function PDB-format of the SGF field**

| 31 | 30 | 29 | 28 |
|---|---|---|---|
| ref m | ref mes-rep | ref C | ref d |
| **NOTE:** If the SGF bit is set, the argument is referenced via a scatter/gather table. If the SGF bit is not set, the argument is referenced via a direct-address pointer. | | | |

This figure shows the format of the CSEL field.

**Table 12-20. MPSign function function PDB-format of the CSEL field**

| 20-17 |
|---|
| CSEL (Curve Select)<br>• 0011 = P256<br>• 0100 = P384<br>• 0101 = P521<br>All other values are reserved. |

# Chapter 13
# SEC register descriptions

The SEC's memory map is divided into the following register address blocks, listed in the table below. Each block is aligned to a 64 KB page boundary.

Table 13-1. SEC Register Address Block Identifiers

| Block Identifier | Included registers |
|---|---|
| 0 | General registers (for example, configuration, control, debugging, and RNG) |
| 1-4 | Job Ring registers (JR0-3) |
| 6 | Real-time integrity check registers |
| 7 | Queue interface registers |
| 8-10 | Descriptor controller DECO 0-2 and CHA control block CCB 0-2 |

All reads of undefined and write-only addresses always return zero. Writes to undefined and read-only addresses are ignored. SEC will never generate a transfer error on the register bus. Although many of the SEC registers hold more than 32 bits, the register addresses shown in the Memory Map below represent how these registers are accessed over the register bus as 32-bit words.

**NOTE**

SEC performs certain actions automatically immediately after POR, and SEC may be used by the boot firmware at boot time. As a consequence, by the time software reads the SEC registers their reset values may already have been changed from the POR values.

**NOTE**

The SEC address space is divided into 16 64 KB pages to match the access granularity of the MMU. Registers that are intended to be accessed by a specific processor or process are grouped into one of these 16 pages so that access to these registers can be restricted via SMMUs or via the CPU's MMU. For instance, the general configuration and status registers are located within page 0 and are intended to be accessed only by privileged software. The registers that control each job ring are located in separate address blocks so that access to each job ring can be restricted to a particular process. Some registers, such as the version ID registers, are intended to be shared among processes. Rather than require each SEC driver process to have two MMU page entries, one page for its private registers and one for the shared registers, SEC "aliases" these shared registers into the upper section of each of the 16 address blocks. Reading any one of the address aliases for the same register returns the same information. Some of these aliased registers are writable, so access to these registers may require that software implement a concurrency control construct, as would be the case with any register that is read/write accessible by multiple processes.

**NOTE**

The reset value of some registers differs between different versions of SEC. To ensure driver compatibility across different versions of SEC, when updating fields within registers the registers should first be read, the required fields updated, and then the register should be written. This will avoid inadvertently changing the settings of other fields in the same register.

Most of SEC's configuration registers are accessible in block 0 of SEC's register space, as indicated in the following table. These registers are intended to be accessed by privileged software (e.g. boot software, hypervisor, secure operating system).

The format and fields in each SEC register are defined below. Some of the register format figures apply to several different registers. In such cases a different register name will be associated with each of the register offset addresses that appear at the top of the register format figure. Although these registers share the same format, they are independent registers. In addition, many registers can be accessed at multiple addresses. In these cases there will be a single register name and the list of addresses at which that register is accessible will be indicated as aliases. Unless noted in the individual register descriptions, registers are reset only at Power-On Reset (POR).

Although many of the SEC registers hold more than 32 bits, these registers are accessed over the register bus as 32-bit words. Note that all registers other than the CCB/DECO registers must be accessed only as full 32-bit words. Byte enables are permitted only for the CCB/DECO registers. All addresses not shown are reserved.

**NOTE**

In order to facilitate software compatibility across both big-endian and little-endian SoCs, SEC can be set to double-word swap registers that are indicated in the Memory Map as 64 bits. (See MCFGR[DWT].) These registers hold address pointers or integers larger than 32 bits, so setting the DWT bit appropriately will cause 64-bit read or write transactions to place the most-significant and least-significant words in the proper positions in 64-bit registers. A register whose width is shown as 32 bits in the Memory Map should be accessed as a single 32-bit bus transaction, even if there is an adjacent related 32-bit register (e.g. CHANUM_MS and CHANUM_LS). SEC does not double-word swap the addresses of such register pairs, so accessing them via 32-bit bus transactions will facilitate software portability across big-endian and little-endian SoCs.

Data read from and written to QI and DECO registers by software is treated as control data for the purpose of endianness conversion.

## 13.1 SEC Memory map

SEC base address: 170_0000h

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 4h | Master Configuration Register (MCFGR) | 32 | RW | 0000_2301h |
| Ch | Security Configuration Register (SCFGR) | 32 | RW | 0000_0000h |
| 10h | Job Ring 0 ICID Register - most significant half (JR0ICID_MS) | 32 | RW | 0000_0000h |
| 14h | Job Ring 0 ICID Register - least significant half (JR0ICID_LS) | 32 | RW | 0000_0000h |
| 18h | Job Ring 1 ICID Register - most significant half (JR1ICID_MS) | 32 | RW | 0000_0000h |
| 1Ch | Job Ring 1 ICID Register - least significant half (JR1ICID_LS) | 32 | RW | 0000_0000h |
| 20h | Job Ring 2 ICID Register - most significant half (JR2ICID_MS) | 32 | RW | 0000_0000h |
| 24h | Job Ring 2 ICID Register - least significant half (JR2ICID_LS) | 32 | RW | 0000_0000h |
| 28h | Job Ring 3 ICID Register - most significant half (JR3ICID_MS) | 32 | RW | 0000_0000h |
| 2Ch | Job Ring 3 ICID Register - least significant half (JR3ICID_LS) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 50h | Queue Manager Interface SDID Register (QISDID) | 32 | RW | 0000_0000h |
| 58h | Debug Control Register (DEBUGCTL) | 32 | RW | 0000_0000h |
| 5Ch | Job Ring Start Register (JRSTARTR) | 32 | RW | 0000_0000h |
| 60h | RTIC ICID Register for Block A - most significant half (RTICAICID_MS) | 32 | RW | 0000_0000h |
| 64h | RTIC ICID Register for Block A - least significant half (RTICAICID_LS) | 32 | RW | 0000_0000h |
| 68h | RTIC ICID Register for Block B - most significant half (RTICBICID_MS) | 32 | RW | 0000_0000h |
| 6Ch | RTIC ICID Register for Block B - least significant half (RTICBICID_LS) | 32 | RW | 0000_0000h |
| 70h | RTIC ICID Register for Block C - most significant half (RTICCICID_MS) | 32 | RW | 0000_0000h |
| 74h | RTIC ICID Register for Block C - least significant half (RTICCICID_LS) | 32 | RW | 0000_0000h |
| 78h | RTIC ICID Register for Block D - most significant half (RTICDICID_MS) | 32 | RW | 0000_0000h |
| 7Ch | RTIC ICID Register for Block D - least significant half (RTICDICID_LS) | 32 | RW | 0000_0000h |
| 94h | DECO Request Source Register (DECORSR) | 32 | RW | 0000_0000h |
| 9Ch | DECO Request Register (DECORR) | 32 | RW | 0000_0000h |
| A0h | DECO0 ICID Register - most significant half (DECO0ICID_MS) | 32 | RW | 0000_0000h |
| A4h | DECO0 ICID Register - least significant half (DECO0ICID_LS) | 32 | RW | 0000_0000h |
| A8h | DECO1 ICID Register - most significant half (DECO1ICID_MS) | 32 | RW | 0000_0000h |
| ACh | DECO1 ICID Register - least significant half (DECO1ICID_LS) | 32 | RW | 0000_0000h |
| B0h | DECO2 ICID Register - most significant half (DECO2ICID_MS) | 32 | RW | 0000_0000h |
| B4h | DECO2 ICID Register - least significant half (DECO2ICID_LS) | 32 | RW | 0000_0000h |
| 120h | DECO Availability Register (DAR) | 32 | RW | 0000_0000h |
| 124h | DECO Reset Register (DRR) | 32 | WO | 0000_0000h |
| 204h | DMA Control Register (DMAC) | 32 | RW | 0000_0003h |
| 220h | Peak Bandwidth Smoothing Limit Register (PBSL) | 32 | RW | 0000_0000h |
| 240h | DMA0_AIDL_MAP_MS (DMA0_AIDL_MAP_MS) | 32 | RO | See description. |
| 244h | DMA0_AIDL_MAP_LS (DMA0_AIDL_MAP_LS) | 32 | RO | See description. |
| 248h | DMA0_AIDM_MAP_MS (DMA0_AIDM_MAP_MS) | 32 | RO | See description. |
| 24Ch | DMA0_AIDM_MAP_LS (DMA0_AIDM_MAP_LS) | 32 | RO | See description. |
| 250h | DMA0 AXI ID Enable Register (DMA0_AID_ENB) | 32 | RO | See description. |
| 260h | DMA0 AXI Read Timing Check Register (DMA0_ARD_TC) | 64 | RW | 0000_0000_0000_0000h |
| 26Ch | DMA0 Read Timing Check Latency Register (DMA0_ARD_LAT) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 270h | DMA0 AXI Write Timing Check Register (DMA0_AWR_TC) | 64 | RW | 0000_0000_0000_0000h |
| 27Ch | DMA0 Write Timing Check Latency Register (DMA0_AWR_LAT) | 32 | RW | 0000_0000h |
| 300h - 33Fh | Manufacturing Protection Private Key Register (MPPKR0 - MPPKR63) | 8 | RW | 00h |
| 380h - 39Fh | Manufacturing Protection Message Register (MPMR0 - MPMR31) | 8 | RW | 00h |
| 3C0h - 3DFh | Manufacturing Protection Test Register (MPTESTR0 - MPTESTR31) | 8 | RO | 00h |
| 400h - 41Ch | Job Descriptor Key Encryption Key Register (JDKEKR0 - JDKEKR7) | 32 | RW | See description. |
| 420h - 43Ch | Trusted Descriptor Key Encryption Key Register (TDKEKR0 - TDKEKR7) | 32 | RW | See description. |
| 440h - 45Ch | Trusted Descriptor Signing Key Register (TDSKR0 - TDSKR7) | 32 | RW | See description. |
| 4E0h | Secure Key Nonce Register (SKNR) | 64 | RW | 0000_0000_0000_0000h |
| 504h | DMA Control Register (DMA_CTRL) | 32 | RW | 0000_0003h |
| 50Ch | DMA Status Register (DMA_STA) | 32 | RO | 0000_0000h |
| 510h | DMA_X_AID_7_4_MAP (DMA_X_AID_7_4_MAP) | 32 | RO | See description. |
| 514h | DMA_X_AID_3_0_MAP (DMA_X_AID_3_0_MAP) | 32 | RO | See description. |
| 518h | DMA_X_AID_15_12_MAP (DMA_X_AID_15_12_MAP) | 32 | RO | See description. |
| 51Ch | DMA_X_AID_11_8_MAP (DMA_X_AID_11_8_MAP) | 32 | RO | See description. |
| 524h | DMA_X AXI ID Map Enable Register (DMA_X_AID_15_0_EN) | 32 | RO | 0000_0000h |
| 530h | DMA_X AXI Read Timing Check Control Register (DMA_X_ARTC_CTL) | 32 | RW | 0000_0000h |
| 534h | DMA_X AXI Read Timing Check Late Count Register (DMA_X_ARTC_LC) | 32 | RW | 0000_0000h |
| 538h | DMA_X AXI Read Timing Check Sample Count Register (DMA_X_ARTC_SC) | 32 | RW | 0000_0000h |
| 53Ch | DMA_X Read Timing Check Latency Register (DMA_X_ARTC_LAT) | 32 | RW | 0000_0000h |
| 540h | DMA_X AXI Write Timing Check Control Register (DMA_X_AWTC_CTL) | 32 | RW | 0000_0000h |
| 544h | DMA_X AXI Write Timing Check Late Count Register (DMA_X_AWTC_LC) | 32 | RW | 0000_0000h |
| 548h | DMA_X AXI Write Timing Check Sample Count Register (DMA_X_AWTC_SC) | 32 | RW | 0000_0000h |
| 54Ch | DMA_X Write Timing Check Latency Register (DMA_X_AWTC_LAT) | 32 | RW | 0000_0000h |
| 600h | RNG TRNG Miscellaneous Control Register (RTMCTL) | 32 | RW | 0000_0001h |
| 604h | RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC) | 32 | RW | 0001_0022h |
| 608h | RNG TRNG Poker Range Register (RTPKRRNG) | 32 | RW | 0000_09A3h |
| 60Ch | RNG TRNG Poker Maximum Limit Register (RTPKRMAX) | 32 | RW | 0000_6920h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 60Ch | RNG TRNG Poker Square Calculation Result Register (RTPKRSQ) | 32 | RO | 0000_0000h |
| 610h | RNG TRNG Seed Control Register (RTSDCTL) | 32 | RW | 0C80_09C4h |
| 614h | RNG TRNG Sparse Bit Limit Register (RTSBLIM) | 32 | RW | 0000_003Fh |
| 614h | RNG TRNG Total Samples Register (RTTOTSAM) | 32 | RO | 0000_0000h |
| 618h | RNG TRNG Frequency Count Minimum Limit Register (RTFRQMIN) | 32 | RW | 0000_0190h |
| 61Ch | RNG TRNG Frequency Count Register (RTFRQCNT) | 32 | RO | 0000_0000h |
| 61Ch | RNG TRNG Frequency Count Maximum Limit Register (RTFRQMAX) | 32 | RW | 0000_1900h |
| 620h | RNG TRNG Statistical Check Monobit Count Register (RTSCMC) | 32 | RO | 0000_0000h |
| 620h | RNG TRNG Statistical Check Monobit Limit Register (RTSCML) | 32 | RW | 010C_0568h |
| 624h | RNG TRNG Statistical Check Run Length 1 Count Register (RTSC R1C) | 32 | RO | 0000_0000h |
| 624h | RNG TRNG Statistical Check Run Length 1 Limit Register (RTSC R1L) | 32 | RW | 00B2_0195h |
| 628h | RNG TRNG Statistical Check Run Length 2 Count Register (RTSC R2C) | 32 | RO | 0000_0000h |
| 628h | RNG TRNG Statistical Check Run Length 2 Limit Register (RTSC R2L) | 32 | RW | 007A_00DCh |
| 62Ch | RNG TRNG Statistical Check Run Length 3 Count Register (RTSC R3C) | 32 | RO | 0000_0000h |
| 62Ch | RNG TRNG Statistical Check Run Length 3 Limit Register (RTSC R3L) | 32 | RW | 0058_007Dh |
| 630h | RNG TRNG Statistical Check Run Length 4 Count Register (RTSC R4C) | 32 | RO | 0000_0000h |
| 630h | RNG TRNG Statistical Check Run Length 4 Limit Register (RTSC R4L) | 32 | RW | 0040_004Bh |
| 634h | RNG TRNG Statistical Check Run Length 5 Count Register (RTSC R5C) | 32 | RO | 0000_0000h |
| 634h | RNG TRNG Statistical Check Run Length 5 Limit Register (RTSC R5L) | 32 | RW | 002E_002Fh |
| 638h | RNG TRNG Statistical Check Run Length 6+ Count Register (RTSC R6PC) | 32 | RO | 0000_0000h |
| 638h | RNG TRNG Statistical Check Run Length 6+ Limit Register (RTSC R6PL) | 32 | RW | 002E_002Fh |
| 63Ch | RNG TRNG Status Register (RTSTATUS) | 32 | RO | 0000_0000h |
| 640h - 67Ch | RNG TRNG Entropy Read Register (RTENT0 - RTENT15) | 32 | RO | 0000_0000h |
| 680h | RNG TRNG Statistical Check Poker Count 1 and 0 Register (RTPK RCNT10) | 32 | RO | 0000_0000h |
| 684h | RNG TRNG Statistical Check Poker Count 3 and 2 Register (RTPK RCNT32) | 32 | RO | 0000_0000h |
| 688h | RNG TRNG Statistical Check Poker Count 5 and 4 Register (RTPK RCNT54) | 32 | RO | 0000_0000h |
| 68Ch | RNG TRNG Statistical Check Poker Count 7 and 6 Register (RTPK RCNT76) | 32 | RO | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 690h | RNG TRNG Statistical Check Poker Count 9 and 8 Register (RTPKRCNT98) | 32 | RO | 0000_0000h |
| 694h | RNG TRNG Statistical Check Poker Count B and A Register (RTPKRCNTBA) | 32 | RO | 0000_0000h |
| 698h | RNG TRNG Statistical Check Poker Count D and C Register (RTPKRCNTDC) | 32 | RO | 0000_0000h |
| 69Ch | RNG TRNG Statistical Check Poker Count F and E Register (RTPKRCNTFE) | 32 | RO | 0000_0000h |
| 6C0h | RNG DRNG Status Register (RDSTA) | 32 | RO | 0000_0000h |
| 6D0h | RNG DRNG State Handle 0 Reseed Interval Register (RDINT0) | 32 | RO | 0000_0000h |
| 6D4h | RNG DRNG State Handle 1 Reseed Interval Register (RDINT1) | 32 | RO | 0000_0000h |
| 6E0h | RNG DRNG Hash Control Register (RDHCNTL) | 32 | RW | 0000_0000h |
| 6E4h | RNG DRNG Hash Digest Register (RDHDIG) | 32 | RO | 0000_0000h |
| 6E8h | RNG DRNG Hash Buffer Register (RDHBUF) | 32 | WO | 0000_0000h |
| B00h | Recoverable Error Indication Status (REIS) | 32 | W1C | 0000_0000h |
| B0Ch | Recoverable Error Indication Halt (REIH) | 32 | RW | 0000_0000h |
| BF8h (alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| BFCh (alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |
| C00h | Holding Tank 0 Job Descriptor Address (HT0_JD_ADDR) | 64 | RO | 0000_0000_0000_0000h |
| C08h | Holding Tank 0 Shared Descriptor Address (HT0_SD_ADDR) | 64 | RO | 0000_0000_0000_0000h |
| C10h | Holding Tank 0 Job Queue Control, most-significant half (HT0_JQ_CTRL_MS) | 32 | RO | 0000_0000h |
| C14h | Holding Tank 0 Job Queue Control, least-significant half (HT0_JQ_CTRL_LS) | 32 | RO | 0000_0000h |
| C1Ch | Holding Tank Status (HT0_STATUS) | 32 | RO | 0000_0000h |
| C24h | Job Queue Debug Select Register (JQ_DEBUG_SEL) | 32 | RW | 0000_0000h |
| DBCh | Job Ring Job IDs in Use Register, least-significant half (JRJIDU_LS) | 32 | RO | 0000_0000h |
| DC0h | Job Ring Job-Done Job ID FIFO BC (JRJDJIFBC) | 32 | RO | 0000_0000h |
| DC4h | Job Ring Job-Done Job ID FIFO (JRJDJIF) | 32 | RO | 0000_0000h |
| DE4h | Job Ring Job-Done Source 1 (JRJDS1) | 32 | RO | 0000_0000h |
| E00h | Job Ring Job-Done Descriptor Address 0 Register (JRJDDA) | 64 | RO | 0000_0000_0000_0000h |
| F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |
| F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |
| FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |
| FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |
| FD4h (alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |
| FE0h (alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| FE4h (alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |
| FE8h (alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| FECh (alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |
| FF0h (alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| FF4h (alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |
| FF8h (alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| FFCh (alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |
| 1_0000h | Input Ring Base Address Register for Job Ring 0 (IRBAR_JR0) | 64 | RW | 0000_0000_0000_0000h |
| 1_000Ch | Input Ring Size Register for Job Ring 0 (IRSR_JR0) | 32 | RW | 0000_0000h |
| 1_0014h | Input Ring Slots Available Register for Job Ring 0 (IRSAR_JR0) | 32 | RW | 0000_0000h |
| 1_001Ch | Input Ring Jobs Added Register for Job Ring0 (IRJAR_JR0) | 32 | RW | 0000_0000h |
| 1_0020h | Output Ring Base Address Register for Job Ring 0 (ORBAR_JR0) | 64 | RW | 0000_0000_0000_0000h |
| 1_002Ch | Output Ring Size Register for Job Ring 0 (ORSR_JR0) | 32 | RW | 0000_0000h |
| 1_0034h | Output Ring Jobs Removed Register for Job Ring 0 (ORJRR_JR0) | 32 | RW | 0000_0000h |
| 1_003Ch | Output Ring Slots Full Register for Job Ring 0 (ORSFR_JR0) | 32 | RW | 0000_0000h |
| 1_0044h | Job Ring Output Status Register for Job Ring 0 (JRSTAR_JR0) | 32 | RO | 0000_0000h |
| 1_004Ch | Job Ring Interrupt Status Register for Job Ring 0 (JRINTR_JR0) | 32 | W1C | 0000_0000h |
| 1_0050h | Job Ring Configuration Register for Job Ring 0, most-significant half (JRCFGR_JR0_MS) | 32 | RW | 0000_0000h |
| 1_0054h | Job Ring Configuration Register for Job Ring 0, least-significant half (JRCFGR_JR0_LS) | 32 | RW | 0000_0000h |
| 1_005Ch | Input Ring Read Index Register for Job Ring 0 (IRRIR_JR0) | 32 | RW | 0000_0000h |
| 1_0064h | Output Ring Write Index Register for Job Ring 0 (ORWIR_JR0) | 32 | RW | 0000_0000h |
| 1_006Ch | Job Ring Command Register for Job Ring 0 (JRCR_JR0) | 32 | WO | 0000_0000h |
| 1_0704h | Job Ring 0 Address-Array Valid Register (JR0AAV) | 32 | RO | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 1_0800h | Job Ring 0 Address-Array Address 0 Register (JR0AAA0) | 64 | RO | 0000_0000_0000_0000h |
| 1_0808h | Job Ring 0 Address-Array Address 1 Register (JR0AAA1) | 64 | RO | 0000_0000_0000_0000h |
| 1_0810h | Job Ring 0 Address-Array Address 2 Register (JR0AAA2) | 64 | RO | 0000_0000_0000_0000h |
| 1_0818h | Job Ring 0 Address-Array Address 3 Register (JR0AAA3) | 64 | RO | 0000_0000_0000_0000h |
| 1_0820h | Job Ring 0 Address-Array Address 4 Register (JR0AAA4) | 64 | RO | 0000_0000_0000_0000h |
| 1_0828h | Job Ring 0 Address-Array Address 5 Register (JR0AAA5) | 64 | RO | 0000_0000_0000_0000h |
| 1_0830h | Job Ring 0 Address-Array Address 6 Register (JR0AAA6) | 64 | RO | 0000_0000_0000_0000h |
| 1_0838h | Job Ring 0 Address-Array Address 7 Register (JR0AAA7) | 64 | RO | 0000_0000_0000_0000h |
| 1_0E00h | Recoverable Error Indication Record 0 for Job Ring 0 (REIR0JR0) | 32 | RO | 0000_0000h |
| 1_0E08h | Recoverable Error Indication Record 2 for Job Ring 0 (REIR2JR0) | 64 | RO | 0000_0000_0000_0000h |
| 1_0E10h | Recoverable Error Indication Record 4 for Job Ring 0 (REIR4JR0) | 32 | RO | 0000_0000h |
| 1_0E14h | Recoverable Error Indication Record 5 for Job Ring 0 (REIR5JR0) | 32 | RO | 0000_0000h |
| 1_0F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |
| 1_0F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 1_0F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 1_0F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 1_0F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |
| 1_0F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 1_0F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| 1_0FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| 1_0FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| 1_0FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| 1_0FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| 1_0FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 1_0FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |
| 1_0FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |
| 1_0FD4h (alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |
| 1_0FE0h (alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| 1_0FE4h (alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |
| 1_0FE8h (alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| 1_0FECh (alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |
| 1_0FF0h (alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| 1_0FF4h (alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |
| 1_0FF8h (alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| 1_0FFCh (alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |
| 2_0000h | Input Ring Base Address Register for Job Ring 1 (IRBAR_JR1) | 64 | RW | 0000_0000_0000_0000h |
| 2_000Ch | Input Ring Size Register for Job Ring 1 (IRSR_JR1) | 32 | RW | 0000_0000h |
| 2_0014h | Input Ring Slots Available Register for Job Ring 1 (IRSAR_JR1) | 32 | RW | 0000_0000h |
| 2_001Ch | Input Ring Jobs Added Register for Job Ring1 (IRJAR_JR1) | 32 | RW | 0000_0000h |
| 2_0020h | Output Ring Base Address Register for Job Ring 1 (ORBAR_JR1) | 64 | RW | 0000_0000_0000_0000h |
| 2_002Ch | Output Ring Size Register for Job Ring 1 (ORSR_JR1) | 32 | RW | 0000_0000h |
| 2_0034h | Output Ring Jobs Removed Register for Job Ring 1 (ORJRR_JR1) | 32 | RW | 0000_0000h |
| 2_003Ch | Output Ring Slots Full Register for Job Ring 1 (ORSFR_JR1) | 32 | RW | 0000_0000h |
| 2_0044h | Job Ring Output Status Register for Job Ring 1 (JRSTAR_JR1) | 32 | RO | 0000_0000h |
| 2_004Ch | Job Ring Interrupt Status Register for Job Ring 1 (JRINTR_JR1) | 32 | W1C | 0000_0000h |
| 2_0050h | Job Ring Configuration Register for Job Ring 1, most-significant half (JRCFGR_JR1_MS) | 32 | RW | 0000_0000h |
| 2_0054h | Job Ring Configuration Register for Job Ring 1, least-significant half (JRCFGR_JR1_LS) | 32 | RW | 0000_0000h |
| 2_005Ch | Input Ring Read Index Register for Job Ring 1 (IRRIR_JR1) | 32 | RW | 0000_0000h |
| 2_0064h | Output Ring Write Index Register for Job Ring 1 (ORWIR_JR1) | 32 | RW | 0000_0000h |
| 2_006Ch | Job Ring Command Register for Job Ring 1 (JRCR_JR1) | 32 | WO | 0000_0000h |
| 2_0704h | Job Ring 1 Address-Array Valid Register (JR1AAV) | 32 | RO | 0000_0000h |
| 2_0800h | Job Ring 1 Address-Array Address 0 Register (JR1AAA0) | 64 | RO | 0000_0000_0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 2_0808h | Job Ring 1 Address-Array Address 1 Register (JR1AAA1) | 64 | RO | 0000_0000_0000_0000h |
| 2_0810h | Job Ring 1 Address-Array Address 2 Register (JR1AAA2) | 64 | RO | 0000_0000_0000_0000h |
| 2_0818h | Job Ring 1 Address-Array Address 3 Register (JR1AAA3) | 64 | RO | 0000_0000_0000_0000h |
| 2_0820h | Job Ring 1 Address-Array Address 4 Register (JR1AAA4) | 64 | RO | 0000_0000_0000_0000h |
| 2_0828h | Job Ring 1 Address-Array Address 5 Register (JR1AAA5) | 64 | RO | 0000_0000_0000_0000h |
| 2_0830h | Job Ring 1 Address-Array Address 6 Register (JR1AAA6) | 64 | RO | 0000_0000_0000_0000h |
| 2_0838h | Job Ring 1 Address-Array Address 7 Register (JR1AAA7) | 64 | RO | 0000_0000_0000_0000h |
| 2_0E00h | Recoverable Error Indication Record 0 for Job Ring 1 (REIR0JR1) | 32 | RO | 0000_0000h |
| 2_0E08h | Recoverable Error Indication Record 2 for Job Ring 1 (REIR2JR1) | 64 | RO | 0000_0000_0000_0000h |
| 2_0E10h | Recoverable Error Indication Record 4 for Job Ring 1 (REIR4JR1) | 32 | RO | 0000_0000h |
| 2_0E14h | Recoverable Error Indication Record 5 for Job Ring 1 (REIR5JR1) | 32 | RO | 0000_0000h |
| 2_0F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |
| 2_0F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 2_0F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 2_0F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 2_0F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |
| 2_0F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 2_0F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| 2_0FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| 2_0FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| 2_0FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| 2_0FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| 2_0FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |
| 2_0FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 2_0FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |
| 2_0FD4h (alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |
| 2_0FE0h (alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| 2_0FE4h (alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |
| 2_0FE8h (alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| 2_0FECh (alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |
| 2_0FF0h (alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| 2_0FF4h (alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |
| 2_0FF8h (alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| 2_0FFCh (alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |
| 3_0000h | Input Ring Base Address Register for Job Ring 2 (IRBAR_JR2) | 64 | RW | 0000_0000_0000_0000h |
| 3_000Ch | Input Ring Size Register for Job Ring 2 (IRSR_JR2) | 32 | RW | 0000_0000h |
| 3_0014h | Input Ring Slots Available Register for Job Ring 2 (IRSAR_JR2) | 32 | RW | 0000_0000h |
| 3_001Ch | Input Ring Jobs Added Register for Job Ring2 (IRJAR_JR2) | 32 | RW | 0000_0000h |
| 3_0020h | Output Ring Base Address Register for Job Ring 2 (ORBAR_JR2) | 64 | RW | 0000_0000_0000_0000h |
| 3_002Ch | Output Ring Size Register for Job Ring 2 (ORSR_JR2) | 32 | RW | 0000_0000h |
| 3_0034h | Output Ring Jobs Removed Register for Job Ring 2 (ORJRR_JR2) | 32 | RW | 0000_0000h |
| 3_003Ch | Output Ring Slots Full Register for Job Ring 2 (ORSFR_JR2) | 32 | RW | 0000_0000h |
| 3_0044h | Job Ring Output Status Register for Job Ring 2 (JRSTAR_JR2) | 32 | RO | 0000_0000h |
| 3_004Ch | Job Ring Interrupt Status Register for Job Ring 2 (JRINTR_JR2) | 32 | W1C | 0000_0000h |
| 3_0050h | Job Ring Configuration Register for Job Ring 2, most-significant half (JRCFGR_JR2_MS) | 32 | RW | 0000_0000h |
| 3_0054h | Job Ring Configuration Register for Job Ring 2, least-significant half (JRCFGR_JR2_LS) | 32 | RW | 0000_0000h |
| 3_005Ch | Input Ring Read Index Register for Job Ring 2 (IRRIR_JR2) | 32 | RW | 0000_0000h |
| 3_0064h | Output Ring Write Index Register for Job Ring 2 (ORWIR_JR2) | 32 | RW | 0000_0000h |
| 3_006Ch | Job Ring Command Register for Job Ring 2 (JRCR_JR2) | 32 | WO | 0000_0000h |
| 3_0704h | Job Ring 2 Address-Array Valid Register (JR2AAV) | 32 | RO | 0000_0000h |
| 3_0800h | Job Ring 2 Address-Array Address 0 Register (JR2AAA0) | 64 | RO | 0000_0000_0000_0000h |
| 3_0808h | Job Ring 2 Address-Array Address 1 Register (JR2AAA1) | 64 | RO | 0000_0000_0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 3_0810h | Job Ring 2 Address-Array Address 2 Register (JR2AAA2) | 64 | RO | 0000_0000_0000_0000h |
| 3_0818h | Job Ring 2 Address-Array Address 3 Register (JR2AAA3) | 64 | RO | 0000_0000_0000_0000h |
| 3_0820h | Job Ring 2 Address-Array Address 4 Register (JR2AAA4) | 64 | RO | 0000_0000_0000_0000h |
| 3_0828h | Job Ring 2 Address-Array Address 5 Register (JR2AAA5) | 64 | RO | 0000_0000_0000_0000h |
| 3_0830h | Job Ring 2 Address-Array Address 6 Register (JR2AAA6) | 64 | RO | 0000_0000_0000_0000h |
| 3_0838h | Job Ring 2 Address-Array Address 7 Register (JR2AAA7) | 64 | RO | 0000_0000_0000_0000h |
| 3_0E00h | Recoverable Error Indication Record 0 for Job Ring 2 (REIR0JR2) | 32 | RO | 0000_0000h |
| 3_0E08h | Recoverable Error Indication Record 2 for Job Ring 2 (REIR2JR2) | 64 | RO | 0000_0000_0000_0000h |
| 3_0E10h | Recoverable Error Indication Record 4 for Job Ring 2 (REIR4JR2) | 32 | RO | 0000_0000h |
| 3_0E14h | Recoverable Error Indication Record 5 for Job Ring 2 (REIR5JR2) | 32 | RO | 0000_0000h |
| 3_0F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |
| 3_0F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 3_0F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 3_0F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 3_0F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |
| 3_0F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 3_0F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| 3_0FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| 3_0FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| 3_0FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| 3_0FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| 3_0FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |
| 3_0FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |
| 3_0FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 3_0FD4h<br>(alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |
| 3_0FE0h<br>(alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| 3_0FE4h<br>(alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |
| 3_0FE8h<br>(alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| 3_0FECh<br>(alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |
| 3_0FF0h<br>(alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| 3_0FF4h<br>(alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |
| 3_0FF8h<br>(alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| 3_0FFCh<br>(alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |
| 4_0000h | Input Ring Base Address Register for Job Ring 3 (IRBAR_JR3) | 64 | RW | 0000_0000_0000_0000h |
| 4_000Ch | Input Ring Size Register for Job Ring 3 (IRSR_JR3) | 32 | RW | 0000_0000h |
| 4_0014h | Input Ring Slots Available Register for Job Ring 3 (IRSAR_JR3) | 32 | RW | 0000_0000h |
| 4_001Ch | Input Ring Jobs Added Register for Job Ring3 (IRJAR_JR3) | 32 | RW | 0000_0000h |
| 4_0020h | Output Ring Base Address Register for Job Ring 3 (ORBAR_JR3) | 64 | RW | 0000_0000_0000_0000h |
| 4_002Ch | Output Ring Size Register for Job Ring 3 (ORSR_JR3) | 32 | RW | 0000_0000h |
| 4_0034h | Output Ring Jobs Removed Register for Job Ring 3 (ORJRR_JR3) | 32 | RW | 0000_0000h |
| 4_003Ch | Output Ring Slots Full Register for Job Ring 3 (ORSFR_JR3) | 32 | RW | 0000_0000h |
| 4_0044h | Job Ring Output Status Register for Job Ring 3 (JRSTAR_JR3) | 32 | RO | 0000_0000h |
| 4_004Ch | Job Ring Interrupt Status Register for Job Ring 3 (JRINTR_JR3) | 32 | W1C | 0000_0000h |
| 4_0050h | Job Ring Configuration Register for Job Ring 3, most-significant half (JRCFGR_JR3_MS) | 32 | RW | 0000_0000h |
| 4_0054h | Job Ring Configuration Register for Job Ring 3, least-significant half (JRCFGR_JR3_LS) | 32 | RW | 0000_0000h |
| 4_005Ch | Input Ring Read Index Register for Job Ring 3 (IRRIR_JR3) | 32 | RW | 0000_0000h |
| 4_0064h | Output Ring Write Index Register for Job Ring 3 (ORWIR_JR3) | 32 | RW | 0000_0000h |
| 4_006Ch | Job Ring Command Register for Job Ring 3 (JRCR_JR3) | 32 | WO | 0000_0000h |
| 4_0704h | Job Ring 3 Address-Array Valid Register (JR3AAV) | 32 | RO | 0000_0000h |
| 4_0800h | Job Ring 3 Address-Array Address 0 Register (JR3AAA0) | 64 | RO | 0000_0000_0000_0000h |
| 4_0808h | Job Ring 3 Address-Array Address 1 Register (JR3AAA1) | 64 | RO | 0000_0000_0000_0000h |
| 4_0810h | Job Ring 3 Address-Array Address 2 Register (JR3AAA2) | 64 | RO | 0000_0000_0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 4_0818h | Job Ring 3 Address-Array Address 3 Register (JR3AAA3) | 64 | RO | 0000_0000_0000_0000h |
| 4_0820h | Job Ring 3 Address-Array Address 4 Register (JR3AAA4) | 64 | RO | 0000_0000_0000_0000h |
| 4_0828h | Job Ring 3 Address-Array Address 5 Register (JR3AAA5) | 64 | RO | 0000_0000_0000_0000h |
| 4_0830h | Job Ring 3 Address-Array Address 6 Register (JR3AAA6) | 64 | RO | 0000_0000_0000_0000h |
| 4_0838h | Job Ring 3 Address-Array Address 7 Register (JR3AAA7) | 64 | RO | 0000_0000_0000_0000h |
| 4_0E00h | Recoverable Error Indication Record 0 for Job Ring 3 (REIR0JR3) | 32 | RO | 0000_0000h |
| 4_0E08h | Recoverable Error Indication Record 2 for Job Ring 3 (REIR2JR3) | 64 | RO | 0000_0000_0000_0000h |
| 4_0E10h | Recoverable Error Indication Record 4 for Job Ring 3 (REIR4JR3) | 32 | RO | 0000_0000h |
| 4_0E14h | Recoverable Error Indication Record 5 for Job Ring 3 (REIR5JR3) | 32 | RO | 0000_0000h |
| 4_0F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |
| 4_0F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 4_0F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 4_0F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 4_0F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |
| 4_0F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 4_0F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| 4_0FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| 4_0FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| 4_0FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| 4_0FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| 4_0FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |
| 4_0FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |
| 4_0FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |
| 4_0FD4h (alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 4_0FE0h<br>(alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| 4_0FE4h<br>(alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |
| 4_0FE8h<br>(alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| 4_0FECh<br>(alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |
| 4_0FF0h<br>(alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| 4_0FF4h<br>(alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |
| 4_0FF8h<br>(alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| 4_0FFCh<br>(alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |
| 6_0004h | RTIC Status Register (RSTA) | 32 | RO | 0000_0000h |
| 6_000Ch | RTIC Command Register (RCMD) | 32 | RW | 0000_0000h |
| 6_0014h | RTIC Control Register (RCTL) | 32 | RW | 0000_0000h |
| 6_001Ch | RTIC Throttle Register (RTHR) | 32 | RW | 0000_0000h |
| 6_0028h | RTIC Watchdog Timer (RWDOG) | 64 | RW | 0000_0000_0000_0000h |
| 6_0034h | RTIC Endian Register (REND) | 32 | RW | 0000_0000h |
| 6_0100h | RTIC Memory Block A Address 0 Register (RMAA0) | 64 | RW | 0000_0000_0000_0000h |
| 6_010Ch | RTIC Memory Block A Length 0 Register (RMAL0) | 32 | RW | 0000_0000h |
| 6_0110h | RTIC Memory Block A Address 1 Register (RMAA1) | 64 | RW | 0000_0000_0000_0000h |
| 6_011Ch | RTIC Memory Block A Length 1 Register (RMAL1) | 32 | RW | 0000_0000h |
| 6_0120h | RTIC Memory Block B Address 0 Register (RMBA0) | 64 | RW | 0000_0000_0000_0000h |
| 6_012Ch | RTIC Memory Block B Length 0 Register (RMBL0) | 32 | RW | 0000_0000h |
| 6_0130h | RTIC Memory Block B Address 1 Register (RMBA1) | 64 | RW | 0000_0000_0000_0000h |
| 6_013Ch | RTIC Memory Block B Length 1 Register (RMBL1) | 32 | RW | 0000_0000h |
| 6_0140h | RTIC Memory Block C Address 0 Register (RMCA0) | 64 | RW | 0000_0000_0000_0000h |
| 6_014Ch | RTIC Memory Block C Length 0 Register (RMCL0) | 32 | RW | 0000_0000h |
| 6_0150h | RTIC Memory Block C Address 1 Register (RMCA1) | 64 | RW | 0000_0000_0000_0000h |
| 6_015Ch | RTIC Memory Block C Length 1 Register (RMCL1) | 32 | RW | 0000_0000h |
| 6_0160h | RTIC Memory Block D Address 0 Register (RMDA0) | 64 | RW | 0000_0000_0000_0000h |
| 6_016Ch | RTIC Memory Block D Length 0 Register (RMDL0) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_0170h | RTIC Memory Block D Address 1 Register (RMDA1) | 64 | RW | 0000_0000_0000_0000h |
| 6_017Ch | RTIC Memory Block D Length 1 Register (RMDL1) | 32 | RW | 0000_0000h |
| 6_0200h | RTIC Memory Block A Big Endian Hash Result Word 0 (RAMDB_0) | 32 | RW | 0000_0000h |
| 6_0204h | RTIC Memory Block A Big Endian Hash Result Word 1 (RAMDB_1) | 32 | RW | 0000_0000h |
| 6_0208h | RTIC Memory Block A Big Endian Hash Result Word 2 (RAMDB_2) | 32 | RW | 0000_0000h |
| 6_020Ch | RTIC Memory Block A Big Endian Hash Result Word 3 (RAMDB_3) | 32 | RW | 0000_0000h |
| 6_0210h | RTIC Memory Block A Big Endian Hash Result Word 4 (RAMDB_4) | 32 | RW | 0000_0000h |
| 6_0214h | RTIC Memory Block A Big Endian Hash Result Word 5 (RAMDB_5) | 32 | RW | 0000_0000h |
| 6_0218h | RTIC Memory Block A Big Endian Hash Result Word 6 (RAMDB_6) | 32 | RW | 0000_0000h |
| 6_021Ch | RTIC Memory Block A Big Endian Hash Result Word 7 (RAMDB_7) | 32 | RW | 0000_0000h |
| 6_0220h | RTIC Memory Block A Big Endian Hash Result Word 8 (RAMDB_8) | 32 | RW | 0000_0000h |
| 6_0224h | RTIC Memory Block A Big Endian Hash Result Word 9 (RAMDB_9) | 32 | RW | 0000_0000h |
| 6_0228h | RTIC Memory Block A Big Endian Hash Result Word 10 (RAMDB_10) | 32 | RW | 0000_0000h |
| 6_022Ch | RTIC Memory Block A Big Endian Hash Result Word 11 (RAMDB_11) | 32 | RW | 0000_0000h |
| 6_0230h | RTIC Memory Block A Big Endian Hash Result Word 12 (RAMDB_12) | 32 | RW | 0000_0000h |
| 6_0234h | RTIC Memory Block A Big Endian Hash Result Word 13 (RAMDB_13) | 32 | RW | 0000_0000h |
| 6_0238h | RTIC Memory Block A Big Endian Hash Result Word 14 (RAMDB_14) | 32 | RW | 0000_0000h |
| 6_023Ch | RTIC Memory Block A Big Endian Hash Result Word 15 (RAMDB_15) | 32 | RW | 0000_0000h |
| 6_0240h | RTIC Memory Block A Big Endian Hash Result Word 16 (RAMDB_16) | 32 | RW | 0000_0000h |
| 6_0244h | RTIC Memory Block A Big Endian Hash Result Word 17 (RAMDB_17) | 32 | RW | 0000_0000h |
| 6_0248h | RTIC Memory Block A Big Endian Hash Result Word 18 (RAMDB_18) | 32 | RW | 0000_0000h |
| 6_024Ch | RTIC Memory Block A Big Endian Hash Result Word 19 (RAMDB_19) | 32 | RW | 0000_0000h |
| 6_0250h | RTIC Memory Block A Big Endian Hash Result Word 20 (RAMDB_20) | 32 | RW | 0000_0000h |
| 6_0254h | RTIC Memory Block A Big Endian Hash Result Word 21 (RAMDB_21) | 32 | RW | 0000_0000h |
| 6_0258h | RTIC Memory Block A Big Endian Hash Result Word 22 (RAMDB_22) | 32 | RW | 0000_0000h |
| 6_025Ch | RTIC Memory Block A Big Endian Hash Result Word 23 (RAMDB_23) | 32 | RW | 0000_0000h |
| 6_0260h | RTIC Memory Block A Big Endian Hash Result Word 24 (RAMDB_24) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_0264h | RTIC Memory Block A Big Endian Hash Result Word 25 (RAMDB_25) | 32 | RW | 0000_0000h |
| 6_0268h | RTIC Memory Block A Big Endian Hash Result Word 26 (RAMDB_26) | 32 | RW | 0000_0000h |
| 6_026Ch | RTIC Memory Block A Big Endian Hash Result Word 27 (RAMDB_27) | 32 | RW | 0000_0000h |
| 6_0270h | RTIC Memory Block A Big Endian Hash Result Word 28 (RAMDB_28) | 32 | RW | 0000_0000h |
| 6_0274h | RTIC Memory Block A Big Endian Hash Result Word 29 (RAMDB_29) | 32 | RW | 0000_0000h |
| 6_0278h | RTIC Memory Block A Big Endian Hash Result Word 30 (RAMDB_30) | 32 | RW | 0000_0000h |
| 6_027Ch | RTIC Memory Block A Big Endian Hash Result Word 31 (RAMDB_31) | 32 | RW | 0000_0000h |
| 6_0280h | RTIC Memory Block A Little Endian Hash Result Word 0 (RAMDL_0) | 32 | RW | 0000_0000h |
| 6_0284h | RTIC Memory Block A Little Endian Hash Result Word 1 (RAMDL_1) | 32 | RW | 0000_0000h |
| 6_0288h | RTIC Memory Block A Little Endian Hash Result Word 2 (RAMDL_2) | 32 | RW | 0000_0000h |
| 6_028Ch | RTIC Memory Block A Little Endian Hash Result Word 3 (RAMDL_3) | 32 | RW | 0000_0000h |
| 6_0290h | RTIC Memory Block A Little Endian Hash Result Word 4 (RAMDL_4) | 32 | RW | 0000_0000h |
| 6_0294h | RTIC Memory Block A Little Endian Hash Result Word 5 (RAMDL_5) | 32 | RW | 0000_0000h |
| 6_0298h | RTIC Memory Block A Little Endian Hash Result Word 6 (RAMDL_6) | 32 | RW | 0000_0000h |
| 6_029Ch | RTIC Memory Block A Little Endian Hash Result Word 7 (RAMDL_7) | 32 | RW | 0000_0000h |
| 6_02A0h | RTIC Memory Block A Little Endian Hash Result Word 8 (RAMDL_8) | 32 | RW | 0000_0000h |
| 6_02A4h | RTIC Memory Block A Little Endian Hash Result Word 9 (RAMDL_9) | 32 | RW | 0000_0000h |
| 6_02A8h | RTIC Memory Block A Little Endian Hash Result Word 10 (RAMDL_10) | 32 | RW | 0000_0000h |
| 6_02ACh | RTIC Memory Block A Little Endian Hash Result Word 11 (RAMDL_11) | 32 | RW | 0000_0000h |
| 6_02B0h | RTIC Memory Block A Little Endian Hash Result Word 12 (RAMDL_12) | 32 | RW | 0000_0000h |
| 6_02B4h | RTIC Memory Block A Little Endian Hash Result Word 13 (RAMDL_13) | 32 | RW | 0000_0000h |
| 6_02B8h | RTIC Memory Block A Little Endian Hash Result Word 14 (RAMDL_14) | 32 | RW | 0000_0000h |
| 6_02BCh | RTIC Memory Block A Little Endian Hash Result Word 15 (RAMDL_15) | 32 | RW | 0000_0000h |
| 6_02C0h | RTIC Memory Block A Little Endian Hash Result Word 16 (RAMDL_16) | 32 | RW | 0000_0000h |
| 6_02C4h | RTIC Memory Block A Little Endian Hash Result Word 17 (RAMDL_17) | 32 | RW | 0000_0000h |
| 6_02C8h | RTIC Memory Block A Little Endian Hash Result Word 18 (RAMDL_18) | 32 | RW | 0000_0000h |
| 6_02CCh | RTIC Memory Block A Little Endian Hash Result Word 19 (RAMDL_19) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_02D0h | RTIC Memory Block A Little Endian Hash Result Word 20 (RAMDL_20) | 32 | RW | 0000_0000h |
| 6_02D4h | RTIC Memory Block A Little Endian Hash Result Word 21 (RAMDL_21) | 32 | RW | 0000_0000h |
| 6_02D8h | RTIC Memory Block A Little Endian Hash Result Word 22 (RAMDL_22) | 32 | RW | 0000_0000h |
| 6_02DCh | RTIC Memory Block A Little Endian Hash Result Word 23 (RAMDL_23) | 32 | RW | 0000_0000h |
| 6_02E0h | RTIC Memory Block A Little Endian Hash Result Word 24 (RAMDL_24) | 32 | RW | 0000_0000h |
| 6_02E4h | RTIC Memory Block A Little Endian Hash Result Word 25 (RAMDL_25) | 32 | RW | 0000_0000h |
| 6_02E8h | RTIC Memory Block A Little Endian Hash Result Word 26 (RAMDL_26) | 32 | RW | 0000_0000h |
| 6_02ECh | RTIC Memory Block A Little Endian Hash Result Word 27 (RAMDL_27) | 32 | RW | 0000_0000h |
| 6_02F0h | RTIC Memory Block A Little Endian Hash Result Word 28 (RAMDL_28) | 32 | RW | 0000_0000h |
| 6_02F4h | RTIC Memory Block A Little Endian Hash Result Word 29 (RAMDL_29) | 32 | RW | 0000_0000h |
| 6_02F8h | RTIC Memory Block A Little Endian Hash Result Word 30 (RAMDL_30) | 32 | RW | 0000_0000h |
| 6_02FCh | RTIC Memory Block A Little Endian Hash Result Word 31 (RAMDL_31) | 32 | RW | 0000_0000h |
| 6_0300h | RTIC Memory Block B Big Endian Hash Result Word 0 (RBMDB_0) | 32 | RW | 0000_0000h |
| 6_0304h | RTIC Memory Block B Big Endian Hash Result Word 1 (RBMDB_1) | 32 | RW | 0000_0000h |
| 6_0308h | RTIC Memory Block B Big Endian Hash Result Word 2 (RBMDB_2) | 32 | RW | 0000_0000h |
| 6_030Ch | RTIC Memory Block B Big Endian Hash Result Word 3 (RBMDB_3) | 32 | RW | 0000_0000h |
| 6_0310h | RTIC Memory Block B Big Endian Hash Result Word 4 (RBMDB_4) | 32 | RW | 0000_0000h |
| 6_0314h | RTIC Memory Block B Big Endian Hash Result Word 5 (RBMDB_5) | 32 | RW | 0000_0000h |
| 6_0318h | RTIC Memory Block B Big Endian Hash Result Word 6 (RBMDB_6) | 32 | RW | 0000_0000h |
| 6_031Ch | RTIC Memory Block B Big Endian Hash Result Word 7 (RBMDB_7) | 32 | RW | 0000_0000h |
| 6_0320h | RTIC Memory Block B Big Endian Hash Result Word 8 (RBMDB_8) | 32 | RW | 0000_0000h |
| 6_0324h | RTIC Memory Block B Big Endian Hash Result Word 9 (RBMDB_9) | 32 | RW | 0000_0000h |
| 6_0328h | RTIC Memory Block B Big Endian Hash Result Word 10 (RBMDB_10) | 32 | RW | 0000_0000h |
| 6_032Ch | RTIC Memory Block B Big Endian Hash Result Word 11 (RBMDB_11) | 32 | RW | 0000_0000h |
| 6_0330h | RTIC Memory Block B Big Endian Hash Result Word 12 (RBMDB_12) | 32 | RW | 0000_0000h |
| 6_0334h | RTIC Memory Block B Big Endian Hash Result Word 13 (RBMDB_13) | 32 | RW | 0000_0000h |
| 6_0338h | RTIC Memory Block B Big Endian Hash Result Word 14 (RBMDB_14) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_033Ch | RTIC Memory Block B Big Endian Hash Result Word 15 (RBMDB_15) | 32 | RW | 0000_0000h |
| 6_0340h | RTIC Memory Block B Big Endian Hash Result Word 16 (RBMDB_16) | 32 | RW | 0000_0000h |
| 6_0344h | RTIC Memory Block B Big Endian Hash Result Word 17 (RBMDB_17) | 32 | RW | 0000_0000h |
| 6_0348h | RTIC Memory Block B Big Endian Hash Result Word 18 (RBMDB_18) | 32 | RW | 0000_0000h |
| 6_034Ch | RTIC Memory Block B Big Endian Hash Result Word 19 (RBMDB_19) | 32 | RW | 0000_0000h |
| 6_0350h | RTIC Memory Block B Big Endian Hash Result Word 20 (RBMDB_20) | 32 | RW | 0000_0000h |
| 6_0354h | RTIC Memory Block B Big Endian Hash Result Word 21 (RBMDB_21) | 32 | RW | 0000_0000h |
| 6_0358h | RTIC Memory Block B Big Endian Hash Result Word 22 (RBMDB_22) | 32 | RW | 0000_0000h |
| 6_035Ch | RTIC Memory Block B Big Endian Hash Result Word 23 (RBMDB_23) | 32 | RW | 0000_0000h |
| 6_0360h | RTIC Memory Block B Big Endian Hash Result Word 24 (RBMDB_24) | 32 | RW | 0000_0000h |
| 6_0364h | RTIC Memory Block B Big Endian Hash Result Word 25 (RBMDB_25) | 32 | RW | 0000_0000h |
| 6_0368h | RTIC Memory Block B Big Endian Hash Result Word 26 (RBMDB_26) | 32 | RW | 0000_0000h |
| 6_036Ch | RTIC Memory Block B Big Endian Hash Result Word 27 (RBMDB_27) | 32 | RW | 0000_0000h |
| 6_0370h | RTIC Memory Block B Big Endian Hash Result Word 28 (RBMDB_28) | 32 | RW | 0000_0000h |
| 6_0374h | RTIC Memory Block B Big Endian Hash Result Word 29 (RBMDB_29) | 32 | RW | 0000_0000h |
| 6_0378h | RTIC Memory Block B Big Endian Hash Result Word 30 (RBMDB_30) | 32 | RW | 0000_0000h |
| 6_037Ch | RTIC Memory Block B Big Endian Hash Result Word 31 (RBMDB_31) | 32 | RW | 0000_0000h |
| 6_0380h | RTIC Memory Block B Little Endian Hash Result Word 0 (RBMDL_0) | 32 | RW | 0000_0000h |
| 6_0384h | RTIC Memory Block B Little Endian Hash Result Word 1 (RBMDL_1) | 32 | RW | 0000_0000h |
| 6_0388h | RTIC Memory Block B Little Endian Hash Result Word 2 (RBMDL_2) | 32 | RW | 0000_0000h |
| 6_038Ch | RTIC Memory Block B Little Endian Hash Result Word 3 (RBMDL_3) | 32 | RW | 0000_0000h |
| 6_0390h | RTIC Memory Block B Little Endian Hash Result Word 4 (RBMDL_4) | 32 | RW | 0000_0000h |
| 6_0394h | RTIC Memory Block B Little Endian Hash Result Word 5 (RBMDL_5) | 32 | RW | 0000_0000h |
| 6_0398h | RTIC Memory Block B Little Endian Hash Result Word 6 (RBMDL_6) | 32 | RW | 0000_0000h |
| 6_039Ch | RTIC Memory Block B Little Endian Hash Result Word 7 (RBMDL_7) | 32 | RW | 0000_0000h |
| 6_03A0h | RTIC Memory Block B Little Endian Hash Result Word 8 (RBMDL_8) | 32 | RW | 0000_0000h |
| 6_03A4h | RTIC Memory Block B Little Endian Hash Result Word 9 (RBMDL_9) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_03A8h | RTIC Memory Block B Little Endian Hash Result Word 10 (RBMDL_10) | 32 | RW | 0000_0000h |
| 6_03ACh | RTIC Memory Block B Little Endian Hash Result Word 11 (RBMDL_11) | 32 | RW | 0000_0000h |
| 6_03B0h | RTIC Memory Block B Little Endian Hash Result Word 12 (RBMDL_12) | 32 | RW | 0000_0000h |
| 6_03B4h | RTIC Memory Block B Little Endian Hash Result Word 13 (RBMDL_13) | 32 | RW | 0000_0000h |
| 6_03B8h | RTIC Memory Block B Little Endian Hash Result Word 14 (RBMDL_14) | 32 | RW | 0000_0000h |
| 6_03BCh | RTIC Memory Block B Little Endian Hash Result Word 15 (RBMDL_15) | 32 | RW | 0000_0000h |
| 6_03C0h | RTIC Memory Block B Little Endian Hash Result Word 16 (RBMDL_16) | 32 | RW | 0000_0000h |
| 6_03C4h | RTIC Memory Block B Little Endian Hash Result Word 17 (RBMDL_17) | 32 | RW | 0000_0000h |
| 6_03C8h | RTIC Memory Block B Little Endian Hash Result Word 18 (RBMDL_18) | 32 | RW | 0000_0000h |
| 6_03CCh | RTIC Memory Block B Little Endian Hash Result Word 19 (RBMDL_19) | 32 | RW | 0000_0000h |
| 6_03D0h | RTIC Memory Block B Little Endian Hash Result Word 20 (RBMDL_20) | 32 | RW | 0000_0000h |
| 6_03D4h | RTIC Memory Block B Little Endian Hash Result Word 21 (RBMDL_21) | 32 | RW | 0000_0000h |
| 6_03D8h | RTIC Memory Block B Little Endian Hash Result Word 22 (RBMDL_22) | 32 | RW | 0000_0000h |
| 6_03DCh | RTIC Memory Block B Little Endian Hash Result Word 23 (RBMDL_23) | 32 | RW | 0000_0000h |
| 6_03E0h | RTIC Memory Block B Little Endian Hash Result Word 24 (RBMDL_24) | 32 | RW | 0000_0000h |
| 6_03E4h | RTIC Memory Block B Little Endian Hash Result Word 25 (RBMDL_25) | 32 | RW | 0000_0000h |
| 6_03E8h | RTIC Memory Block B Little Endian Hash Result Word 26 (RBMDL_26) | 32 | RW | 0000_0000h |
| 6_03ECh | RTIC Memory Block B Little Endian Hash Result Word 27 (RBMDL_27) | 32 | RW | 0000_0000h |
| 6_03F0h | RTIC Memory Block B Little Endian Hash Result Word 28 (RBMDL_28) | 32 | RW | 0000_0000h |
| 6_03F4h | RTIC Memory Block B Little Endian Hash Result Word 29 (RBMDL_29) | 32 | RW | 0000_0000h |
| 6_03F8h | RTIC Memory Block B Little Endian Hash Result Word 30 (RBMDL_30) | 32 | RW | 0000_0000h |
| 6_03FCh | RTIC Memory Block B Little Endian Hash Result Word 31 (RBMDL_31) | 32 | RW | 0000_0000h |
| 6_0400h | RTIC Memory Block C Big Endian Hash Result Word 0 (RCMDB_0) | 32 | RW | 0000_0000h |
| 6_0404h | RTIC Memory Block C Big Endian Hash Result Word 1 (RCMDB_1) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_0408h | RTIC Memory Block C Big Endian Hash Result Word 2 (RCMDB_2) | 32 | RW | 0000_0000h |
| 6_040Ch | RTIC Memory Block C Big Endian Hash Result Word 3 (RCMDB_3) | 32 | RW | 0000_0000h |
| 6_0410h | RTIC Memory Block C Big Endian Hash Result Word 4 (RCMDB_4) | 32 | RW | 0000_0000h |
| 6_0414h | RTIC Memory Block C Big Endian Hash Result Word 5 (RCMDB_5) | 32 | RW | 0000_0000h |
| 6_0418h | RTIC Memory Block C Big Endian Hash Result Word 6 (RCMDB_6) | 32 | RW | 0000_0000h |
| 6_041Ch | RTIC Memory Block C Big Endian Hash Result Word 7 (RCMDB_7) | 32 | RW | 0000_0000h |
| 6_0420h | RTIC Memory Block C Big Endian Hash Result Word 8 (RCMDB_8) | 32 | RW | 0000_0000h |
| 6_0424h | RTIC Memory Block C Big Endian Hash Result Word 9 (RCMDB_9) | 32 | RW | 0000_0000h |
| 6_0428h | RTIC Memory Block C Big Endian Hash Result Word 10 (RCMDB_10) | 32 | RW | 0000_0000h |
| 6_042Ch | RTIC Memory Block C Big Endian Hash Result Word 11 (RCMDB_11) | 32 | RW | 0000_0000h |
| 6_0430h | RTIC Memory Block C Big Endian Hash Result Word 12 (RCMDB_12) | 32 | RW | 0000_0000h |
| 6_0434h | RTIC Memory Block C Big Endian Hash Result Word 13 (RCMDB_13) | 32 | RW | 0000_0000h |
| 6_0438h | RTIC Memory Block C Big Endian Hash Result Word 14 (RCMDB_14) | 32 | RW | 0000_0000h |
| 6_043Ch | RTIC Memory Block C Big Endian Hash Result Word 15 (RCMDB_15) | 32 | RW | 0000_0000h |
| 6_0440h | RTIC Memory Block C Big Endian Hash Result Word 16 (RCMDB_16) | 32 | RW | 0000_0000h |
| 6_0444h | RTIC Memory Block C Big Endian Hash Result Word 17 (RCMDB_17) | 32 | RW | 0000_0000h |
| 6_0448h | RTIC Memory Block C Big Endian Hash Result Word 18 (RCMDB_18) | 32 | RW | 0000_0000h |
| 6_044Ch | RTIC Memory Block C Big Endian Hash Result Word 19 (RCMDB_19) | 32 | RW | 0000_0000h |
| 6_0450h | RTIC Memory Block C Big Endian Hash Result Word 20 (RCMDB_20) | 32 | RW | 0000_0000h |
| 6_0454h | RTIC Memory Block C Big Endian Hash Result Word 21 (RCMDB_21) | 32 | RW | 0000_0000h |
| 6_0458h | RTIC Memory Block C Big Endian Hash Result Word 22 (RCMDB_22) | 32 | RW | 0000_0000h |
| 6_045Ch | RTIC Memory Block C Big Endian Hash Result Word 23 (RCMDB_23) | 32 | RW | 0000_0000h |
| 6_0460h | RTIC Memory Block C Big Endian Hash Result Word 24 (RCMDB_24) | 32 | RW | 0000_0000h |
| 6_0464h | RTIC Memory Block C Big Endian Hash Result Word 25 (RCMDB_25) | 32 | RW | 0000_0000h |
| 6_0468h | RTIC Memory Block C Big Endian Hash Result Word 26 (RCMDB_26) | 32 | RW | 0000_0000h |
| 6_046Ch | RTIC Memory Block C Big Endian Hash Result Word 27 (RCMDB_27) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_0470h | RTIC Memory Block C Big Endian Hash Result Word 28 (RCMDB_28) | 32 | RW | 0000_0000h |
| 6_0474h | RTIC Memory Block C Big Endian Hash Result Word 29 (RCMDB_29) | 32 | RW | 0000_0000h |
| 6_0478h | RTIC Memory Block C Big Endian Hash Result Word 30 (RCMDB_30) | 32 | RW | 0000_0000h |
| 6_047Ch | RTIC Memory Block C Big Endian Hash Result Word 31 (RCMDB_31) | 32 | RW | 0000_0000h |
| 6_0480h | RTIC Memory Block C Little Endian Hash Result Word 0 (RCMDL_0) | 32 | RW | 0000_0000h |
| 6_0484h | RTIC Memory Block C Little Endian Hash Result Word 1 (RCMDL_1) | 32 | RW | 0000_0000h |
| 6_0488h | RTIC Memory Block C Little Endian Hash Result Word 2 (RCMDL_2) | 32 | RW | 0000_0000h |
| 6_048Ch | RTIC Memory Block C Little Endian Hash Result Word 3 (RCMDL_3) | 32 | RW | 0000_0000h |
| 6_0490h | RTIC Memory Block C Little Endian Hash Result Word 4 (RCMDL_4) | 32 | RW | 0000_0000h |
| 6_0494h | RTIC Memory Block C Little Endian Hash Result Word 5 (RCMDL_5) | 32 | RW | 0000_0000h |
| 6_0498h | RTIC Memory Block C Little Endian Hash Result Word 6 (RCMDL_6) | 32 | RW | 0000_0000h |
| 6_049Ch | RTIC Memory Block C Little Endian Hash Result Word 7 (RCMDL_7) | 32 | RW | 0000_0000h |
| 6_04A0h | RTIC Memory Block C Little Endian Hash Result Word 8 (RCMDL_8) | 32 | RW | 0000_0000h |
| 6_04A4h | RTIC Memory Block C Little Endian Hash Result Word 9 (RCMDL_9) | 32 | RW | 0000_0000h |
| 6_04A8h | RTIC Memory Block C Little Endian Hash Result Word 10 (RCMDL_10) | 32 | RW | 0000_0000h |
| 6_04ACh | RTIC Memory Block C Little Endian Hash Result Word 11 (RCMDL_11) | 32 | RW | 0000_0000h |
| 6_04B0h | RTIC Memory Block C Little Endian Hash Result Word 12 (RCMDL_12) | 32 | RW | 0000_0000h |
| 6_04B4h | RTIC Memory Block C Little Endian Hash Result Word 13 (RCMDL_13) | 32 | RW | 0000_0000h |
| 6_04B8h | RTIC Memory Block C Little Endian Hash Result Word 14 (RCMDL_14) | 32 | RW | 0000_0000h |
| 6_04BCh | RTIC Memory Block C Little Endian Hash Result Word 15 (RCMDL_15) | 32 | RW | 0000_0000h |
| 6_04C0h | RTIC Memory Block C Little Endian Hash Result Word 16 (RCMDL_16) | 32 | RW | 0000_0000h |
| 6_04C4h | RTIC Memory Block C Little Endian Hash Result Word 17 (RCMDL_17) | 32 | RW | 0000_0000h |
| 6_04C8h | RTIC Memory Block C Little Endian Hash Result Word 18 (RCMDL_18) | 32 | RW | 0000_0000h |
| 6_04CCh | RTIC Memory Block C Little Endian Hash Result Word 19 (RCMDL_19) | 32 | RW | 0000_0000h |
| 6_04D0h | RTIC Memory Block C Little Endian Hash Result Word 20 (RCMDL_20) | 32 | RW | 0000_0000h |
| 6_04D4h | RTIC Memory Block C Little Endian Hash Result Word 21 (RCMDL_21) | 32 | RW | 0000_0000h |
| 6_04D8h | RTIC Memory Block C Little Endian Hash Result Word 22 (RCMDL_22) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_04DCh | RTIC Memory Block C Little Endian Hash Result Word 23 (RCMDL_23) | 32 | RW | 0000_0000h |
| 6_04E0h | RTIC Memory Block C Little Endian Hash Result Word 24 (RCMDL_24) | 32 | RW | 0000_0000h |
| 6_04E4h | RTIC Memory Block C Little Endian Hash Result Word 25 (RCMDL_25) | 32 | RW | 0000_0000h |
| 6_04E8h | RTIC Memory Block C Little Endian Hash Result Word 26 (RCMDL_26) | 32 | RW | 0000_0000h |
| 6_04ECh | RTIC Memory Block C Little Endian Hash Result Word 27 (RCMDL_27) | 32 | RW | 0000_0000h |
| 6_04F0h | RTIC Memory Block C Little Endian Hash Result Word 28 (RCMDL_28) | 32 | RW | 0000_0000h |
| 6_04F4h | RTIC Memory Block C Little Endian Hash Result Word 29 (RCMDL_29) | 32 | RW | 0000_0000h |
| 6_04F8h | RTIC Memory Block C Little Endian Hash Result Word 30 (RCMDL_30) | 32 | RW | 0000_0000h |
| 6_04FCh | RTIC Memory Block C Little Endian Hash Result Word 31 (RCMDL_31) | 32 | RW | 0000_0000h |
| 6_0500h | RTIC Memory Block D Big Endian Hash Result Word 0 (RDMDB_0) | 32 | RW | 0000_0000h |
| 6_0504h | RTIC Memory Block D Big Endian Hash Result Word 1 (RDMDB_1) | 32 | RW | 0000_0000h |
| 6_0508h | RTIC Memory Block D Big Endian Hash Result Word 2 (RDMDB_2) | 32 | RW | 0000_0000h |
| 6_050Ch | RTIC Memory Block D Big Endian Hash Result Word 3 (RDMDB_3) | 32 | RW | 0000_0000h |
| 6_0510h | RTIC Memory Block D Big Endian Hash Result Word 4 (RDMDB_4) | 32 | RW | 0000_0000h |
| 6_0514h | RTIC Memory Block D Big Endian Hash Result Word 5 (RDMDB_5) | 32 | RW | 0000_0000h |
| 6_0518h | RTIC Memory Block D Big Endian Hash Result Word 6 (RDMDB_6) | 32 | RW | 0000_0000h |
| 6_051Ch | RTIC Memory Block D Big Endian Hash Result Word 7 (RDMDB_7) | 32 | RW | 0000_0000h |
| 6_0520h | RTIC Memory Block D Big Endian Hash Result Word 8 (RDMDB_8) | 32 | RW | 0000_0000h |
| 6_0524h | RTIC Memory Block D Big Endian Hash Result Word 9 (RDMDB_9) | 32 | RW | 0000_0000h |
| 6_0528h | RTIC Memory Block D Big Endian Hash Result Word 10 (RDMDB_10) | 32 | RW | 0000_0000h |
| 6_052Ch | RTIC Memory Block D Big Endian Hash Result Word 11 (RDMDB_11) | 32 | RW | 0000_0000h |
| 6_0530h | RTIC Memory Block D Big Endian Hash Result Word 12 (RDMDB_12) | 32 | RW | 0000_0000h |
| 6_0534h | RTIC Memory Block D Big Endian Hash Result Word 13 (RDMDB_13) | 32 | RW | 0000_0000h |
| 6_0538h | RTIC Memory Block D Big Endian Hash Result Word 14 (RDMDB_14) | 32 | RW | 0000_0000h |
| 6_053Ch | RTIC Memory Block D Big Endian Hash Result Word 15 (RDMDB_15) | 32 | RW | 0000_0000h |
| 6_0540h | RTIC Memory Block D Big Endian Hash Result Word 16 (RDMDB_16) | 32 | RW | 0000_0000h |
| 6_0544h | RTIC Memory Block D Big Endian Hash Result Word 17 (RDMDB_17) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_0548h | RTIC Memory Block D Big Endian Hash Result Word 18 (RDMDB_18) | 32 | RW | 0000_0000h |
| 6_054Ch | RTIC Memory Block D Big Endian Hash Result Word 19 (RDMDB_19) | 32 | RW | 0000_0000h |
| 6_0550h | RTIC Memory Block D Big Endian Hash Result Word 20 (RDMDB_20) | 32 | RW | 0000_0000h |
| 6_0554h | RTIC Memory Block D Big Endian Hash Result Word 21 (RDMDB_21) | 32 | RW | 0000_0000h |
| 6_0558h | RTIC Memory Block D Big Endian Hash Result Word 22 (RDMDB_22) | 32 | RW | 0000_0000h |
| 6_055Ch | RTIC Memory Block D Big Endian Hash Result Word 23 (RDMDB_23) | 32 | RW | 0000_0000h |
| 6_0560h | RTIC Memory Block D Big Endian Hash Result Word 24 (RDMDB_24) | 32 | RW | 0000_0000h |
| 6_0564h | RTIC Memory Block D Big Endian Hash Result Word 25 (RDMDB_25) | 32 | RW | 0000_0000h |
| 6_0568h | RTIC Memory Block D Big Endian Hash Result Word 26 (RDMDB_26) | 32 | RW | 0000_0000h |
| 6_056Ch | RTIC Memory Block D Big Endian Hash Result Word 27 (RDMDB_27) | 32 | RW | 0000_0000h |
| 6_0570h | RTIC Memory Block D Big Endian Hash Result Word 28 (RDMDB_28) | 32 | RW | 0000_0000h |
| 6_0574h | RTIC Memory Block D Big Endian Hash Result Word 29 (RDMDB_29) | 32 | RW | 0000_0000h |
| 6_0578h | RTIC Memory Block D Big Endian Hash Result Word 30 (RDMDB_30) | 32 | RW | 0000_0000h |
| 6_057Ch | RTIC Memory Block D Big Endian Hash Result Word 31 (RDMDB_31) | 32 | RW | 0000_0000h |
| 6_0580h | RTIC Memory Block D Little Endian Hash Result Word 0 (RDMDL_0) | 32 | RW | 0000_0000h |
| 6_0584h | RTIC Memory Block D Little Endian Hash Result Word 1 (RDMDL_1) | 32 | RW | 0000_0000h |
| 6_0588h | RTIC Memory Block D Little Endian Hash Result Word 2 (RDMDL_2) | 32 | RW | 0000_0000h |
| 6_058Ch | RTIC Memory Block D Little Endian Hash Result Word 3 (RDMDL_3) | 32 | RW | 0000_0000h |
| 6_0590h | RTIC Memory Block D Little Endian Hash Result Word 4 (RDMDL_4) | 32 | RW | 0000_0000h |
| 6_0594h | RTIC Memory Block D Little Endian Hash Result Word 5 (RDMDL_5) | 32 | RW | 0000_0000h |
| 6_0598h | RTIC Memory Block D Little Endian Hash Result Word 6 (RDMDL_6) | 32 | RW | 0000_0000h |
| 6_059Ch | RTIC Memory Block D Little Endian Hash Result Word 7 (RDMDL_7) | 32 | RW | 0000_0000h |
| 6_05A0h | RTIC Memory Block D Little Endian Hash Result Word 8 (RDMDL_8) | 32 | RW | 0000_0000h |
| 6_05A4h | RTIC Memory Block D Little Endian Hash Result Word 9 (RDMDL_9) | 32 | RW | 0000_0000h |
| 6_05A8h | RTIC Memory Block D Little Endian Hash Result Word 10 (RDMDL_10) | 32 | RW | 0000_0000h |
| 6_05ACh | RTIC Memory Block D Little Endian Hash Result Word 11 (RDMDL_11) | 32 | RW | 0000_0000h |
| 6_05B0h | RTIC Memory Block D Little Endian Hash Result Word 12 (RDMDL_12) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width (In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_05B4h | RTIC Memory Block D Little Endian Hash Result Word 13 (RDMDL_13) | 32 | RW | 0000_0000h |
| 6_05B8h | RTIC Memory Block D Little Endian Hash Result Word 14 (RDMDL_14) | 32 | RW | 0000_0000h |
| 6_05BCh | RTIC Memory Block D Little Endian Hash Result Word 15 (RDMDL_15) | 32 | RW | 0000_0000h |
| 6_05C0h | RTIC Memory Block D Little Endian Hash Result Word 16 (RDMDL_16) | 32 | RW | 0000_0000h |
| 6_05C4h | RTIC Memory Block D Little Endian Hash Result Word 17 (RDMDL_17) | 32 | RW | 0000_0000h |
| 6_05C8h | RTIC Memory Block D Little Endian Hash Result Word 18 (RDMDL_18) | 32 | RW | 0000_0000h |
| 6_05CCh | RTIC Memory Block D Little Endian Hash Result Word 19 (RDMDL_19) | 32 | RW | 0000_0000h |
| 6_05D0h | RTIC Memory Block D Little Endian Hash Result Word 20 (RDMDL_20) | 32 | RW | 0000_0000h |
| 6_05D4h | RTIC Memory Block D Little Endian Hash Result Word 21 (RDMDL_21) | 32 | RW | 0000_0000h |
| 6_05D8h | RTIC Memory Block D Little Endian Hash Result Word 22 (RDMDL_22) | 32 | RW | 0000_0000h |
| 6_05DCh | RTIC Memory Block D Little Endian Hash Result Word 23 (RDMDL_23) | 32 | RW | 0000_0000h |
| 6_05E0h | RTIC Memory Block D Little Endian Hash Result Word 24 (RDMDL_24) | 32 | RW | 0000_0000h |
| 6_05E4h | RTIC Memory Block D Little Endian Hash Result Word 25 (RDMDL_25) | 32 | RW | 0000_0000h |
| 6_05E8h | RTIC Memory Block D Little Endian Hash Result Word 26 (RDMDL_26) | 32 | RW | 0000_0000h |
| 6_05ECh | RTIC Memory Block D Little Endian Hash Result Word 27 (RDMDL_27) | 32 | RW | 0000_0000h |
| 6_05F0h | RTIC Memory Block D Little Endian Hash Result Word 28 (RDMDL_28) | 32 | RW | 0000_0000h |
| 6_05F4h | RTIC Memory Block D Little Endian Hash Result Word 29 (RDMDL_29) | 32 | RW | 0000_0000h |
| 6_05F8h | RTIC Memory Block D Little Endian Hash Result Word 30 (RDMDL_30) | 32 | RW | 0000_0000h |
| 6_05FCh | RTIC Memory Block D Little Endian Hash Result Word 31 (RDMDL_31) | 32 | RW | 0000_0000h |
| 6_0E00h | Recoverable Error Indication Record 0 for RTIC (REIR0RTIC) | 32 | RO | 0000_0000h |
| 6_0E08h | Recoverable Error Indication Record 2 for RTIC (REIR2RTIC) | 64 | RO | 0000_0000_0000_0000h |
| 6_0E10h | Recoverable Error Indication Record 4 for RTIC (REIR4RTIC) | 32 | RO | 0000_0000h |
| 6_0E14h | Recoverable Error Indication Record 5 for RTIC (REIR5RTIC) | 32 | RO | 0000_0000h |
| 6_0F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 6_0F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 6_0F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 6_0F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 6_0F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |
| 6_0F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 6_0F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| 6_0FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| 6_0FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| 6_0FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| 6_0FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| 6_0FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |
| 6_0FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |
| 6_0FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |
| 6_0FD4h (alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |
| 6_0FE0h (alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| 6_0FE4h (alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |
| 6_0FE8h (alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| 6_0FECh (alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |
| 6_0FF0h (alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| 6_0FF4h (alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |
| 6_0FF8h (alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| 6_0FFCh (alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |
| 7_0000h | Queue Interface Control Register, most-significant (QICTL_MS) | 32 | RW | 0000_0000h |
| 7_0004h | Queue Interface Control Register, least-significant (QICTL_LS) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 7_000Ch | Queue Interface Status Register (QISTA) | 32 | W1C | 0000_0000h |
| 7_0010h | Queue Interface Dequeue Configuration Register, most-significant half (QIDQC_MS) | 32 | RW | 0400_FFFFh |
| 7_0014h | Queue Interface Dequeue Configuration Register, least-significant half (QIDQC_LS) | 32 | RW | 0000_0011h |
| 7_0018h | Queue Interface Enqueue Configuration Register, most-significant half (QIEQC_MS) | 32 | RW | 0000_0003h |
| 7_001Ch | Queue Interface Enqueue Configuration Register, least-significant half (QIEQC_LS) | 32 | RW | 0000_0000h |
| 7_0020h | Queue Interface ICID Configuration Register, most-significant half (QIIC_MS) | 32 | RW | 03FF_0000h |
| 7_0024h | Queue Interface ICID Configuration Register, least-significant half (QIIC_LS) | 32 | RW | 0000_0000h |
| 7_0100h | Queue Interface Descriptor Word 0 Register (QIDESC0) | 32 | RO | 0000_0000h |
| 7_0104h - 7_0130h | Queue Interface Descriptor Word a Registers (QIDESC1 - QIDE SC12) | 32 | RO | 0000_0000h |
| 7_0210h | Queue Interface Compound Frame Scatter/Gather Table Registers (QICFOFH_MS) | 32 | RO | 0000_0000h |
| 7_0214h | Queue Interface Compound Frame Scatter/Gather Table Registers (QICFOFH_LS) | 32 | RO | 0000_0000h |
| 7_0218h | Queue Interface Compound Frame Scatter/Gather Table Registers (QICFOFL_MS) | 32 | RO | 0000_0000h |
| 7_021Ch | Queue Interface Compound Frame Scatter/Gather Table Registers (QICFOFL_LS) | 32 | RO | 0000_0000h |
| 7_0220h | Queue Interface Compound Frame Scatter/Gather Table Registers (QICFIFH_MS) | 32 | RO | 0000_0000h |
| 7_0224h | Queue Interface Compound Frame Scatter/Gather Table Registers (QICFIFH_LS) | 32 | RO | 0000_0000h |
| 7_0228h | Queue Interface Compound Frame Scatter/Gather Table Registers (QICFIFL_MS) | 32 | RO | 0000_0000h |
| 7_022Ch | Queue Interface Compound Frame Scatter/Gather Table Registers (QICFIFL_LS) | 32 | RO | 0000_0000h |
| 7_0300h | Queue Interface Job ID Valid Register (QIJIDVALID) | 64 | RO | 0000_0000_0000_0000h |
| 7_0308h | Queue Interface Job ID Job Ready Register (QIJIDRDY) | 64 | RO | 0000_0000_0000_0000h |
| 7_0700h | Recoverable Error Indication Record 0 for the Queue Interface (REIR 0QI) | 32 | RO | 0000_0000h |
| 7_0704h | Recoverable Error Indication Record 1 for the Queue Interface (REIR 1QI) | 32 | RO | 0000_0000h |
| 7_0708h | Recoverable Error Indication Record 2 for the Queue Interface (REIR 2QI) | 64 | RO | 0000_0000_0000_0000h |
| 7_0710h | Recoverable Error Indication Record 4 for the Queue Interface (REIR 4QI) | 32 | RO | 0000_0000h |
| 7_0714h | Recoverable Error Indication Record 5 for the Queue Interface (REIR 5QI) | 32 | RO | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 7_0F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |
| 7_0F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 7_0F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 7_0F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 7_0F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |
| 7_0F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 7_0F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| 7_0FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| 7_0FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| 7_0FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| 7_0FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| 7_0FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |
| 7_0FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |
| 7_0FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |
| 7_0FD4h (alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |
| 7_0FE0h (alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| 7_0FE4h (alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |
| 7_0FE8h (alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| 7_0FECh (alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |
| 7_0FF0h (alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| 7_0FF4h (alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |
| 7_0FF8h (alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| 7_0FFCh (alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 8_0004h | CCB 0 Class 1 Mode Register Format for Non-Public Key Algorithms (C0C1MR_NPK) | 32 | RW | 0000_0000h |
| 8_0004h | CCB 0 Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK) | 32 | RW | 0000_0000h |
| 8_0004h | CCB 0 Class 1 Mode Register Format for RNG4 (C0C1MR_RNG) | 32 | RW | 0000_0000h |
| 8_000Ch | CCB 0 Class 1 Key Size Register (C0C1KSR) | 32 | RW | 0000_0000h |
| 8_0010h | CCB 0 Class 1 Data Size Register (C0C1DSR) | 64 | RW | 0000_0000_0000_0000h |
| 8_001Ch | CCB 0 Class 1 ICV Size Register (C0C1ICVSR) | 32 | RW | 0000_0000h |
| 8_0034h | CCB 0 CHA Control Register (C0CCTRL) | 32 | WO | 0000_0000h |
| 8_003Ch | CCB 0 Interrupt Control Register (C0ICTL) | 32 | W1C | 0000_0000h |
| 8_0044h | CCB 0 Clear Written Register (C0CWR) | 32 | WO | 0000_0000h |
| 8_0048h | CCB 0 Status and Error Register, most-significant half (C0CSTA_MS) | 32 | RO | 0000_0000h |
| 8_004Ch | CCB 0 Status and Error Register, least-significant half (C0CSTA_LS) | 32 | RO | 0000_0000h |
| 8_005Ch | CCB 0 AAD Size Register (C0AADSZR) | 32 | RW | 0000_0000h |
| 8_0064h | Class 1 IV Size Register (C0C1IVSZR) | 32 | RW | 0000_0000h |
| 8_0084h | PKHA A Size Register (C0PKASZR) | 32 | RW | 0000_0000h |
| 8_008Ch | PKHA B Size Register (C0PKBSZR) | 32 | RW | 0000_0000h |
| 8_0094h | PKHA N Size Register (C0PKNSZR) | 32 | RW | 0000_0000h |
| 8_009Ch | PKHA E Size Register (C0PKESZR) | 32 | RW | 0000_0000h |
| 8_0100h | CCB 0 Class 1 Context Register Word 0 (C0C1CTXR0) | 32 | RW | 0000_0000h |
| 8_0104h | CCB 0 Class 1 Context Register Word 1 (C0C1CTXR1) | 32 | RW | 0000_0000h |
| 8_0108h | CCB 0 Class 1 Context Register Word 2 (C0C1CTXR2) | 32 | RW | 0000_0000h |
| 8_010Ch | CCB 0 Class 1 Context Register Word 3 (C0C1CTXR3) | 32 | RW | 0000_0000h |
| 8_0110h | CCB 0 Class 1 Context Register Word 4 (C0C1CTXR4) | 32 | RW | 0000_0000h |
| 8_0114h | CCB 0 Class 1 Context Register Word 5 (C0C1CTXR5) | 32 | RW | 0000_0000h |
| 8_0118h | CCB 0 Class 1 Context Register Word 6 (C0C1CTXR6) | 32 | RW | 0000_0000h |
| 8_011Ch | CCB 0 Class 1 Context Register Word 7 (C0C1CTXR7) | 32 | RW | 0000_0000h |
| 8_0120h | CCB 0 Class 1 Context Register Word 8 (C0C1CTXR8) | 32 | RW | 0000_0000h |
| 8_0124h | CCB 0 Class 1 Context Register Word 9 (C0C1CTXR9) | 32 | RW | 0000_0000h |
| 8_0128h | CCB 0 Class 1 Context Register Word 10 (C0C1CTXR10) | 32 | RW | 0000_0000h |
| 8_012Ch | CCB 0 Class 1 Context Register Word 11 (C0C1CTXR11) | 32 | RW | 0000_0000h |
| 8_0130h | CCB 0 Class 1 Context Register Word 12 (C0C1CTXR12) | 32 | RW | 0000_0000h |
| 8_0134h | CCB 0 Class 1 Context Register Word 13 (C0C1CTXR13) | 32 | RW | 0000_0000h |
| 8_0138h | CCB 0 Class 1 Context Register Word 14 (C0C1CTXR14) | 32 | RW | 0000_0000h |
| 8_013Ch | CCB 0 Class 1 Context Register Word 15 (C0C1CTXR15) | 32 | RW | 0000_0000h |
| 8_0200h | CCB 0 Class 1 Key Registers Word 0 (C0C1KR0) | 32 | RW | 0000_0000h |
| 8_0204h | CCB 0 Class 1 Key Registers Word 1 (C0C1KR1) | 32 | RW | 0000_0000h |
| 8_0208h | CCB 0 Class 1 Key Registers Word 2 (C0C1KR2) | 32 | RW | 0000_0000h |
| 8_020Ch | CCB 0 Class 1 Key Registers Word 3 (C0C1KR3) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 8_0210h | CCB 0 Class 1 Key Registers Word 4 (C0C1KR4) | 32 | RW | 0000_0000h |
| 8_0214h | CCB 0 Class 1 Key Registers Word 5 (C0C1KR5) | 32 | RW | 0000_0000h |
| 8_0218h | CCB 0 Class 1 Key Registers Word 6 (C0C1KR6) | 32 | RW | 0000_0000h |
| 8_021Ch | CCB 0 Class 1 Key Registers Word 7 (C0C1KR7) | 32 | RW | 0000_0000h |
| 8_0404h | CCB 0 Class 2 Mode Register (C0C2MR) | 32 | RW | 0000_0000h |
| 8_040Ch | CCB 0 Class 2 Key Size Register (C0C2KSR) | 32 | RW | 0000_0000h |
| 8_0410h | CCB 0 Class 2 Data Size Register (C0C2DSR) | 64 | RW | 0000_0000_0000_0000h |
| 8_041Ch | CCB 0 Class 2 ICV Size Register (C0C2ICVSZR) | 32 | RW | 0000_0000h |
| 8_0500h | CCB 0 Class 2 Context Register Word 0 (C0C2CTXR0) | 32 | RW | 0000_0000h |
| 8_0504h | CCB 0 Class 2 Context Register Word 1 (C0C2CTXR1) | 32 | RW | 0000_0000h |
| 8_0508h | CCB 0 Class 2 Context Register Word 2 (C0C2CTXR2) | 32 | RW | 0000_0000h |
| 8_050Ch | CCB 0 Class 2 Context Register Word 3 (C0C2CTXR3) | 32 | RW | 0000_0000h |
| 8_0510h | CCB 0 Class 2 Context Register Word 4 (C0C2CTXR4) | 32 | RW | 0000_0000h |
| 8_0514h | CCB 0 Class 2 Context Register Word 5 (C0C2CTXR5) | 32 | RW | 0000_0000h |
| 8_0518h | CCB 0 Class 2 Context Register Word 6 (C0C2CTXR6) | 32 | RW | 0000_0000h |
| 8_051Ch | CCB 0 Class 2 Context Register Word 7 (C0C2CTXR7) | 32 | RW | 0000_0000h |
| 8_0520h | CCB 0 Class 2 Context Register Word 8 (C0C2CTXR8) | 32 | RW | 0000_0000h |
| 8_0524h | CCB 0 Class 2 Context Register Word 9 (C0C2CTXR9) | 32 | RW | 0000_0000h |
| 8_0528h | CCB 0 Class 2 Context Register Word 10 (C0C2CTXR10) | 32 | RW | 0000_0000h |
| 8_052Ch | CCB 0 Class 2 Context Register Word 11 (C0C2CTXR11) | 32 | RW | 0000_0000h |
| 8_0530h | CCB 0 Class 2 Context Register Word 12 (C0C2CTXR12) | 32 | RW | 0000_0000h |
| 8_0534h | CCB 0 Class 2 Context Register Word 13 (C0C2CTXR13) | 32 | RW | 0000_0000h |
| 8_0538h | CCB 0 Class 2 Context Register Word 14 (C0C2CTXR14) | 32 | RW | 0000_0000h |
| 8_053Ch | CCB 0 Class 2 Context Register Word 15 (C0C2CTXR15) | 32 | RW | 0000_0000h |
| 8_0540h | CCB 0 Class 2 Context Register Word 16 (C0C2CTXR16) | 32 | RW | 0000_0000h |
| 8_0544h | CCB 0 Class 2 Context Register Word 17 (C0C2CTXR17) | 32 | RW | 0000_0000h |
| 8_0600h | CCB 0 Class 2 Key Register Word 0 (C0C2KEYR0) | 32 | RW | 0000_0000h |
| 8_0604h | CCB 0 Class 2 Key Register Word 1 (C0C2KEYR1) | 32 | RW | 0000_0000h |
| 8_0608h | CCB 0 Class 2 Key Register Word 2 (C0C2KEYR2) | 32 | RW | 0000_0000h |
| 8_060Ch | CCB 0 Class 2 Key Register Word 3 (C0C2KEYR3) | 32 | RW | 0000_0000h |
| 8_0610h | CCB 0 Class 2 Key Register Word 4 (C0C2KEYR4) | 32 | RW | 0000_0000h |
| 8_0614h | CCB 0 Class 2 Key Register Word 5 (C0C2KEYR5) | 32 | RW | 0000_0000h |
| 8_0618h | CCB 0 Class 2 Key Register Word 6 (C0C2KEYR6) | 32 | RW | 0000_0000h |
| 8_061Ch | CCB 0 Class 2 Key Register Word 7 (C0C2KEYR7) | 32 | RW | 0000_0000h |
| 8_0620h | CCB 0 Class 2 Key Register Word 8 (C0C2KEYR8) | 32 | RW | 0000_0000h |
| 8_0624h | CCB 0 Class 2 Key Register Word 9 (C0C2KEYR9) | 32 | RW | 0000_0000h |
| 8_0628h | CCB 0 Class 2 Key Register Word 10 (C0C2KEYR10) | 32 | RW | 0000_0000h |
| 8_062Ch | CCB 0 Class 2 Key Register Word 11 (C0C2KEYR11) | 32 | RW | 0000_0000h |
| 8_0630h | CCB 0 Class 2 Key Register Word 12 (C0C2KEYR12) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 8_0634h | CCB 0 Class 2 Key Register Word 13 (C0C2KEYR13) | 32 | RW | 0000_0000h |
| 8_0638h | CCB 0 Class 2 Key Register Word 14 (C0C2KEYR14) | 32 | RW | 0000_0000h |
| 8_063Ch | CCB 0 Class 2 Key Register Word 15 (C0C2KEYR15) | 32 | RW | 0000_0000h |
| 8_0640h | CCB 0 Class 2 Key Register Word 16 (C0C2KEYR16) | 32 | RW | 0000_0000h |
| 8_0644h | CCB 0 Class 2 Key Register Word 17 (C0C2KEYR17) | 32 | RW | 0000_0000h |
| 8_0648h | CCB 0 Class 2 Key Register Word 18 (C0C2KEYR18) | 32 | RW | 0000_0000h |
| 8_064Ch | CCB 0 Class 2 Key Register Word 19 (C0C2KEYR19) | 32 | RW | 0000_0000h |
| 8_0650h | CCB 0 Class 2 Key Register Word 20 (C0C2KEYR20) | 32 | RW | 0000_0000h |
| 8_0654h | CCB 0 Class 2 Key Register Word 21 (C0C2KEYR21) | 32 | RW | 0000_0000h |
| 8_0658h | CCB 0 Class 2 Key Register Word 22 (C0C2KEYR22) | 32 | RW | 0000_0000h |
| 8_065Ch | CCB 0 Class 2 Key Register Word 23 (C0C2KEYR23) | 32 | RW | 0000_0000h |
| 8_0660h | CCB 0 Class 2 Key Register Word 24 (C0C2KEYR24) | 32 | RW | 0000_0000h |
| 8_0664h | CCB 0 Class 2 Key Register Word 25 (C0C2KEYR25) | 32 | RW | 0000_0000h |
| 8_0668h | CCB 0 Class 2 Key Register Word 26 (C0C2KEYR26) | 32 | RW | 0000_0000h |
| 8_066Ch | CCB 0 Class 2 Key Register Word 27 (C0C2KEYR27) | 32 | RW | 0000_0000h |
| 8_0670h | CCB 0 Class 2 Key Register Word 28 (C0C2KEYR28) | 32 | RW | 0000_0000h |
| 8_0674h | CCB 0 Class 2 Key Register Word 29 (C0C2KEYR29) | 32 | RW | 0000_0000h |
| 8_0678h | CCB 0 Class 2 Key Register Word 30 (C0C2KEYR30) | 32 | RW | 0000_0000h |
| 8_067Ch | CCB 0 Class 2 Key Register Word 31 (C0C2KEYR31) | 32 | RW | 0000_0000h |
| 8_07C0h | CCB 0 FIFO Status (C0FIFOSTA) | 32 | RO | 0000_0000h |
| 8_07D0h | CCB 0 iNformation FIFO When STYPE Is Not 10 (C0NFIFO) | 32 | WO | 0000_0000h |
| 8_07D0h | CCB 0 iNformation FIFO When STYPE Is 10 (C0NFIFO_2) | 32 | WO | 0000_0000h |
| 8_07E0h | CCB 0 Input Data FIFO (C0IFIFO) | 32 | WO | 0000_0000h |
| 8_07F0h | CCB 0 Output Data FIFO (C0OFIFO) | 64 | RO | 0000_0000_0000_0000h |
| 8_0800h | DECO0 Job Queue Control Register, most-significant half (D0JQCR_MS) | 32 | RW | 0000_0000h |
| 8_0804h | DECO0 Job Queue Control Register, least-significant half (D0JQCR_LS) | 32 | RO | 0000_0000h |
| 8_0808h | DECO0 Descriptor Address Register (D0DAR) | 64 | RO | 0000_0000_0000_0000h |
| 8_0810h | DECO0 Operation Status Register, most-significant half (D0OPSTA_MS) | 32 | RO | 0000_0000h |
| 8_0814h | DECO0 Operation Status Register, least-significant half (D0OPSTA_LS) | 32 | RO | 0000_0000h |
| 8_0818h | DECO0 Checksum Register (D0CKSUMR) | 32 | RW | 0000_0000h |
| 8_0820h | DECO0 ICID Status Register (D0ISR) | 32 | RO | 0000_0000h |
| 8_0820h | DECO0 SDID / Trusted ICID Status Register (D0SDIDSR) | 32 | RW | 0000_0000h |
| 8_0840h | DECO0 Math Register 0_MS (D0MTH0_MS) | 32 | RW | 0000_0000h |
| 8_0844h | DECO0 Math Register 0_LS (D0MTH0_LS) | 32 | RW | 0000_0000h |
| 8_0848h | DECO0 Math Register 1_MS (D0MTH1_MS) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 8_084Ch | DECO0 Math Register 1_LS (D0MTH1_LS) | 32 | RW | 0000_0000h |
| 8_0850h | DECO0 Math Register 2_MS (D0MTH2_MS) | 32 | RW | 0000_0000h |
| 8_0854h | DECO0 Math Register 2_LS (D0MTH2_LS) | 32 | RW | 0000_0000h |
| 8_0858h | DECO0 Math Register 3_MS (D0MTH3_MS) | 32 | RW | 0000_0000h |
| 8_085Ch | DECO0 Math Register 3_LS (D0MTH3_LS) | 32 | RW | 0000_0000h |
| 8_0860h | DECO0 Math Register 4_MS (D0MTH4_MS) | 32 | RW | 0000_0000h |
| 8_0864h | DECO0 Math Register 4_LS (D0MTH4_LS) | 32 | RW | 0000_0000h |
| 8_0868h | DECO0 Math Register 5_MS (D0MTH5_MS) | 32 | RW | 0000_0000h |
| 8_086Ch | DECO0 Math Register 5_LS (D0MTH5_LS) | 32 | RW | 0000_0000h |
| 8_0870h | DECO0 Math Register 6_MS (D0MTH6_MS) | 32 | RW | 0000_0000h |
| 8_0874h | DECO0 Math Register 6_LS (D0MTH6_LS) | 32 | RW | 0000_0000h |
| 8_0878h | DECO0 Math Register 7_MS (D0MTH7_MS) | 32 | RW | 0000_0000h |
| 8_087Ch | DECO0 Math Register 7_LS (D0MTH7_LS) | 32 | RW | 0000_0000h |
| 8_0880h | DECO0 Gather Table Register 0 Word 0 (D0GTR0_0) | 32 | RW | 0000_0000h |
| 8_0884h | DECO0 Gather Table Register 0 Word 1 (D0GTR0_1) | 32 | RW | 0000_0000h |
| 8_0888h | DECO0 Gather Table Register 0 Word 2 (D0GTR0_2) | 32 | RW | 0000_0000h |
| 8_088Ch | DECO0 Gather Table Register 0 Word 3 (D0GTR0_3) | 32 | RW | 0000_0000h |
| 8_0890h | DECO0 Gather Table Register 1 Word 0 (D0GTR1_0) | 32 | RW | 0000_0000h |
| 8_0894h | DECO0 Gather Table Register 1 Word 1 (D0GTR1_1) | 32 | RW | 0000_0000h |
| 8_0898h | DECO0 Gather Table Register 1 Word 2 (D0GTR1_2) | 32 | RW | 0000_0000h |
| 8_089Ch | DECO0 Gather Table Register 1 Word 3 (D0GTR1_3) | 32 | RW | 0000_0000h |
| 8_08A0h | DECO0 Gather Table Register 2 Word 0 (D0GTR2_0) | 32 | RW | 0000_0000h |
| 8_08A4h | DECO0 Gather Table Register 2 Word 1 (D0GTR2_1) | 32 | RW | 0000_0000h |
| 8_08A8h | DECO0 Gather Table Register 2 Word 2 (D0GTR2_2) | 32 | RW | 0000_0000h |
| 8_08ACh | DECO0 Gather Table Register 2 Word 3 (D0GTR2_3) | 32 | RW | 0000_0000h |
| 8_08B0h | DECO0 Gather Table Register 3 Word 0 (D0GTR3_0) | 32 | RW | 0000_0000h |
| 8_08B4h | DECO0 Gather Table Register 3 Word 1 (D0GTR3_1) | 32 | RW | 0000_0000h |
| 8_08B8h | DECO0 Gather Table Register 3 Word 2 (D0GTR3_2) | 32 | RW | 0000_0000h |
| 8_08BCh | DECO0 Gather Table Register 3 Word 3 (D0GTR3_3) | 32 | RW | 0000_0000h |
| 8_0900h | DECO0 Scatter Table Register 0 Word 0 (D0STR0_0) | 32 | RW | 0000_0000h |
| 8_0904h | DECO0 Scatter Table Register 0 Word 1 (D0STR0_1) | 32 | RW | 0000_0000h |
| 8_0908h | DECO0 Scatter Table Register 0 Word 2 (D0STR0_2) | 32 | RW | 0000_0000h |
| 8_090Ch | DECO0 Scatter Table Register 0 Word 3 (D0STR0_3) | 32 | RW | 0000_0000h |
| 8_0910h | DECO0 Scatter Table Register 1 Word 0 (D0STR1_0) | 32 | RW | 0000_0000h |
| 8_0914h | DECO0 Scatter Table Register 1 Word 1 (D0STR1_1) | 32 | RW | 0000_0000h |
| 8_0918h | DECO0 Scatter Table Register 1 Word 2 (D0STR1_2) | 32 | RW | 0000_0000h |
| 8_091Ch | DECO0 Scatter Table Register 1 Word 3 (D0STR1_3) | 32 | RW | 0000_0000h |
| 8_0920h | DECO0 Scatter Table Register 2 Word 0 (D0STR2_0) | 32 | RW | 0000_0000h |
| 8_0924h | DECO0 Scatter Table Register 2 Word 1 (D0STR2_1) | 32 | RW | 0000_0000h |
| 8_0928h | DECO0 Scatter Table Register 2 Word 2 (D0STR2_2) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 8_092Ch | DECO0 Scatter Table Register 2 Word 3 (D0STR2_3) | 32 | RW | 0000_0000h |
| 8_0930h | DECO0 Scatter Table Register 3 Word 0 (D0STR3_0) | 32 | RW | 0000_0000h |
| 8_0934h | DECO0 Scatter Table Register 3 Word 1 (D0STR3_1) | 32 | RW | 0000_0000h |
| 8_0938h | DECO0 Scatter Table Register 3 Word 2 (D0STR3_2) | 32 | RW | 0000_0000h |
| 8_093Ch | DECO0 Scatter Table Register 3 Word 3 (D0STR3_3) | 32 | RW | 0000_0000h |
| 8_0A00h | DECO0 Descriptor Buffer Word 0 (D0DESB0) | 32 | RW | 0000_0000h |
| 8_0A04h | DECO0 Descriptor Buffer Word 1 (D0DESB1) | 32 | RW | 0000_0000h |
| 8_0A08h | DECO0 Descriptor Buffer Word 2 (D0DESB2) | 32 | RW | 0000_0000h |
| 8_0A0Ch | DECO0 Descriptor Buffer Word 3 (D0DESB3) | 32 | RW | 0000_0000h |
| 8_0A10h | DECO0 Descriptor Buffer Word 4 (D0DESB4) | 32 | RW | 0000_0000h |
| 8_0A14h | DECO0 Descriptor Buffer Word 5 (D0DESB5) | 32 | RW | 0000_0000h |
| 8_0A18h | DECO0 Descriptor Buffer Word 6 (D0DESB6) | 32 | RW | 0000_0000h |
| 8_0A1Ch | DECO0 Descriptor Buffer Word 7 (D0DESB7) | 32 | RW | 0000_0000h |
| 8_0A20h | DECO0 Descriptor Buffer Word 8 (D0DESB8) | 32 | RW | 0000_0000h |
| 8_0A24h | DECO0 Descriptor Buffer Word 9 (D0DESB9) | 32 | RW | 0000_0000h |
| 8_0A28h | DECO0 Descriptor Buffer Word 10 (D0DESB10) | 32 | RW | 0000_0000h |
| 8_0A2Ch | DECO0 Descriptor Buffer Word 11 (D0DESB11) | 32 | RW | 0000_0000h |
| 8_0A30h | DECO0 Descriptor Buffer Word 12 (D0DESB12) | 32 | RW | 0000_0000h |
| 8_0A34h | DECO0 Descriptor Buffer Word 13 (D0DESB13) | 32 | RW | 0000_0000h |
| 8_0A38h | DECO0 Descriptor Buffer Word 14 (D0DESB14) | 32 | RW | 0000_0000h |
| 8_0A3Ch | DECO0 Descriptor Buffer Word 15 (D0DESB15) | 32 | RW | 0000_0000h |
| 8_0A40h | DECO0 Descriptor Buffer Word 16 (D0DESB16) | 32 | RW | 0000_0000h |
| 8_0A44h | DECO0 Descriptor Buffer Word 17 (D0DESB17) | 32 | RW | 0000_0000h |
| 8_0A48h | DECO0 Descriptor Buffer Word 18 (D0DESB18) | 32 | RW | 0000_0000h |
| 8_0A4Ch | DECO0 Descriptor Buffer Word 19 (D0DESB19) | 32 | RW | 0000_0000h |
| 8_0A50h | DECO0 Descriptor Buffer Word 20 (D0DESB20) | 32 | RW | 0000_0000h |
| 8_0A54h | DECO0 Descriptor Buffer Word 21 (D0DESB21) | 32 | RW | 0000_0000h |
| 8_0A58h | DECO0 Descriptor Buffer Word 22 (D0DESB22) | 32 | RW | 0000_0000h |
| 8_0A5Ch | DECO0 Descriptor Buffer Word 23 (D0DESB23) | 32 | RW | 0000_0000h |
| 8_0A60h | DECO0 Descriptor Buffer Word 24 (D0DESB24) | 32 | RW | 0000_0000h |
| 8_0A64h | DECO0 Descriptor Buffer Word 25 (D0DESB25) | 32 | RW | 0000_0000h |
| 8_0A68h | DECO0 Descriptor Buffer Word 26 (D0DESB26) | 32 | RW | 0000_0000h |
| 8_0A6Ch | DECO0 Descriptor Buffer Word 27 (D0DESB27) | 32 | RW | 0000_0000h |
| 8_0A70h | DECO0 Descriptor Buffer Word 28 (D0DESB28) | 32 | RW | 0000_0000h |
| 8_0A74h | DECO0 Descriptor Buffer Word 29 (D0DESB29) | 32 | RW | 0000_0000h |
| 8_0A78h | DECO0 Descriptor Buffer Word 30 (D0DESB30) | 32 | RW | 0000_0000h |
| 8_0A7Ch | DECO0 Descriptor Buffer Word 31 (D0DESB31) | 32 | RW | 0000_0000h |
| 8_0A80h | DECO0 Descriptor Buffer Word 32 (D0DESB32) | 32 | RW | 0000_0000h |
| 8_0A84h | DECO0 Descriptor Buffer Word 33 (D0DESB33) | 32 | RW | 0000_0000h |
| 8_0A88h | DECO0 Descriptor Buffer Word 34 (D0DESB34) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 8_0A8Ch | DECO0 Descriptor Buffer Word 35 (D0DESB35) | 32 | RW | 0000_0000h |
| 8_0A90h | DECO0 Descriptor Buffer Word 36 (D0DESB36) | 32 | RW | 0000_0000h |
| 8_0A94h | DECO0 Descriptor Buffer Word 37 (D0DESB37) | 32 | RW | 0000_0000h |
| 8_0A98h | DECO0 Descriptor Buffer Word 38 (D0DESB38) | 32 | RW | 0000_0000h |
| 8_0A9Ch | DECO0 Descriptor Buffer Word 39 (D0DESB39) | 32 | RW | 0000_0000h |
| 8_0AA0h | DECO0 Descriptor Buffer Word 40 (D0DESB40) | 32 | RW | 0000_0000h |
| 8_0AA4h | DECO0 Descriptor Buffer Word 41 (D0DESB41) | 32 | RW | 0000_0000h |
| 8_0AA8h | DECO0 Descriptor Buffer Word 42 (D0DESB42) | 32 | RW | 0000_0000h |
| 8_0AACh | DECO0 Descriptor Buffer Word 43 (D0DESB43) | 32 | RW | 0000_0000h |
| 8_0AB0h | DECO0 Descriptor Buffer Word 44 (D0DESB44) | 32 | RW | 0000_0000h |
| 8_0AB4h | DECO0 Descriptor Buffer Word 45 (D0DESB45) | 32 | RW | 0000_0000h |
| 8_0AB8h | DECO0 Descriptor Buffer Word 46 (D0DESB46) | 32 | RW | 0000_0000h |
| 8_0ABCh | DECO0 Descriptor Buffer Word 47 (D0DESB47) | 32 | RW | 0000_0000h |
| 8_0AC0h | DECO0 Descriptor Buffer Word 48 (D0DESB48) | 32 | RW | 0000_0000h |
| 8_0AC4h | DECO0 Descriptor Buffer Word 49 (D0DESB49) | 32 | RW | 0000_0000h |
| 8_0AC8h | DECO0 Descriptor Buffer Word 50 (D0DESB50) | 32 | RW | 0000_0000h |
| 8_0ACCh | DECO0 Descriptor Buffer Word 51 (D0DESB51) | 32 | RW | 0000_0000h |
| 8_0AD0h | DECO0 Descriptor Buffer Word 52 (D0DESB52) | 32 | RW | 0000_0000h |
| 8_0AD4h | DECO0 Descriptor Buffer Word 53 (D0DESB53) | 32 | RW | 0000_0000h |
| 8_0AD8h | DECO0 Descriptor Buffer Word 54 (D0DESB54) | 32 | RW | 0000_0000h |
| 8_0ADCh | DECO0 Descriptor Buffer Word 55 (D0DESB55) | 32 | RW | 0000_0000h |
| 8_0AE0h | DECO0 Descriptor Buffer Word 56 (D0DESB56) | 32 | RW | 0000_0000h |
| 8_0AE4h | DECO0 Descriptor Buffer Word 57 (D0DESB57) | 32 | RW | 0000_0000h |
| 8_0AE8h | DECO0 Descriptor Buffer Word 58 (D0DESB58) | 32 | RW | 0000_0000h |
| 8_0AECh | DECO0 Descriptor Buffer Word 59 (D0DESB59) | 32 | RW | 0000_0000h |
| 8_0AF0h | DECO0 Descriptor Buffer Word 60 (D0DESB60) | 32 | RW | 0000_0000h |
| 8_0AF4h | DECO0 Descriptor Buffer Word 61 (D0DESB61) | 32 | RW | 0000_0000h |
| 8_0AF8h | DECO0 Descriptor Buffer Word 62 (D0DESB62) | 32 | RW | 0000_0000h |
| 8_0AFCh | DECO0 Descriptor Buffer Word 63 (D0DESB63) | 32 | RW | 0000_0000h |
| 8_0E00h | DECO0 Debug Job (D0DJR) | 32 | RO | 0000_0000h |
| 8_0E04h | DECO0 Debug DECO (D0DDR) | 32 | RO | 0000_0000h |
| 8_0E08h | DECO0 Debug Job Pointer (D0DJP) | 64 | RO | 0000_0000_0000_0000h |
| 8_0E10h | DECO0 Debug Shared Pointer (D0SDP) | 64 | RO | 0000_0000_0000_0000h |
| 8_0E18h | DECO0 Debug_ICID, most-significant half (D0DIR_MS) | 32 | RO | 0000_0000h |
| 8_0E20h | Sequence Output Length Register (SOL0) | 32 | RW | 0000_0000h |
| 8_0E24h | Variable Sequence Output Length Register (VSOL0) | 32 | RW | 0000_0000h |
| 8_0E28h | Sequence Input Length Register (SIL0) | 32 | RW | 0000_0000h |
| 8_0E2Ch | Variable Sequence Input Length Register (VSIL0) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 8_0E30h | Protocol Override Register (D0POVRD) | 32 | RW | 0000_0000h |
| 8_0E34h | Variable Sequence Output Length Register; Upper 32 bits (UVSOL0) | 32 | RW | 0000_0000h |
| 8_0E38h | Variable Sequence Input Length Register; Upper 32 bits (UVSIL0) | 32 | RW | 0000_0000h |
| 8_0F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |
| 8_0F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 8_0F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 8_0F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 8_0F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |
| 8_0F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 8_0F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| 8_0FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| 8_0FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| 8_0FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| 8_0FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| 8_0FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |
| 8_0FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |
| 8_0FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |
| 8_0FD4h (alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |
| 8_0FE0h (alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| 8_0FE4h (alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |
| 8_0FE8h (alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| 8_0FECh (alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |
| 8_0FF0h (alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| 8_0FF4h (alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 8_0FF8h (alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| 8_0FFCh (alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |
| 9_0004h | CCB 1 Class 1 Mode Register Format for Non-Public Key Algorithms (C1C1MR_NPK) | 32 | RW | 0000_0000h |
| 9_0004h | CCB 1 Class 1 Mode Register Format for Public Key Algorithms (C1C1MR_PK) | 32 | RW | 0000_0000h |
| 9_0004h | CCB 1 Class 1 Mode Register Format for RNG4 (C1C1MR_RNG) | 32 | RW | 0000_0000h |
| 9_000Ch | CCB 1 Class 1 Key Size Register (C1C1KSR) | 32 | RW | 0000_0000h |
| 9_0010h | CCB 1 Class 1 Data Size Register (C1C1DSR) | 64 | RW | 0000_0000_0000_0000h |
| 9_001Ch | CCB 1 Class 1 ICV Size Register (C1C1ICVSR) | 32 | RW | 0000_0000h |
| 9_0034h | CCB 1 CHA Control Register (C1CCTRL) | 32 | WO | 0000_0000h |
| 9_003Ch | CCB 1 Interrupt Control Register (C1ICTL) | 32 | W1C | 0000_0000h |
| 9_0044h | CCB 1 Clear Written Register (C1CWR) | 32 | WO | 0000_0000h |
| 9_0048h | CCB 1 Status and Error Register, most-significant half (C1CSTA_MS) | 32 | RO | 0000_0000h |
| 9_004Ch | CCB 1 Status and Error Register, least-significant half (C1CSTA_LS) | 32 | RO | 0000_0000h |
| 9_005Ch | CCB 1 AAD Size Register (C1AADSZR) | 32 | RW | 0000_0000h |
| 9_0064h | Class 1 IV Size Register (C1C1IVSZR) | 32 | RW | 0000_0000h |
| 9_0084h | PKHA A Size Register (C1PKASZR) | 32 | RW | 0000_0000h |
| 9_008Ch | PKHA B Size Register (C1PKBSZR) | 32 | RW | 0000_0000h |
| 9_0094h | PKHA N Size Register (C1PKNSZR) | 32 | RW | 0000_0000h |
| 9_009Ch | PKHA E Size Register (C1PKESZR) | 32 | RW | 0000_0000h |
| 9_0100h | CCB 1 Class 1 Context Register Word 0 (C1C1CTXR0) | 32 | RW | 0000_0000h |
| 9_0104h | CCB 1 Class 1 Context Register Word 1 (C1C1CTXR1) | 32 | RW | 0000_0000h |
| 9_0108h | CCB 1 Class 1 Context Register Word 2 (C1C1CTXR2) | 32 | RW | 0000_0000h |
| 9_010Ch | CCB 1 Class 1 Context Register Word 3 (C1C1CTXR3) | 32 | RW | 0000_0000h |
| 9_0110h | CCB 1 Class 1 Context Register Word 4 (C1C1CTXR4) | 32 | RW | 0000_0000h |
| 9_0114h | CCB 1 Class 1 Context Register Word 5 (C1C1CTXR5) | 32 | RW | 0000_0000h |
| 9_0118h | CCB 1 Class 1 Context Register Word 6 (C1C1CTXR6) | 32 | RW | 0000_0000h |
| 9_011Ch | CCB 1 Class 1 Context Register Word 7 (C1C1CTXR7) | 32 | RW | 0000_0000h |
| 9_0120h | CCB 1 Class 1 Context Register Word 8 (C1C1CTXR8) | 32 | RW | 0000_0000h |
| 9_0124h | CCB 1 Class 1 Context Register Word 9 (C1C1CTXR9) | 32 | RW | 0000_0000h |
| 9_0128h | CCB 1 Class 1 Context Register Word 10 (C1C1CTXR10) | 32 | RW | 0000_0000h |
| 9_012Ch | CCB 1 Class 1 Context Register Word 11 (C1C1CTXR11) | 32 | RW | 0000_0000h |
| 9_0130h | CCB 1 Class 1 Context Register Word 12 (C1C1CTXR12) | 32 | RW | 0000_0000h |
| 9_0134h | CCB 1 Class 1 Context Register Word 13 (C1C1CTXR13) | 32 | RW | 0000_0000h |
| 9_0138h | CCB 1 Class 1 Context Register Word 14 (C1C1CTXR14) | 32 | RW | 0000_0000h |
| 9_013Ch | CCB 1 Class 1 Context Register Word 15 (C1C1CTXR15) | 32 | RW | 0000_0000h |
| 9_0200h | CCB 1 Class 1 Key Registers Word 0 (C1C1KR0) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 9_0204h | CCB 1 Class 1 Key Registers Word 1 (C1C1KR1) | 32 | RW | 0000_0000h |
| 9_0208h | CCB 1 Class 1 Key Registers Word 2 (C1C1KR2) | 32 | RW | 0000_0000h |
| 9_020Ch | CCB 1 Class 1 Key Registers Word 3 (C1C1KR3) | 32 | RW | 0000_0000h |
| 9_0210h | CCB 1 Class 1 Key Registers Word 4 (C1C1KR4) | 32 | RW | 0000_0000h |
| 9_0214h | CCB 1 Class 1 Key Registers Word 5 (C1C1KR5) | 32 | RW | 0000_0000h |
| 9_0218h | CCB 1 Class 1 Key Registers Word 6 (C1C1KR6) | 32 | RW | 0000_0000h |
| 9_021Ch | CCB 1 Class 1 Key Registers Word 7 (C1C1KR7) | 32 | RW | 0000_0000h |
| 9_0404h | CCB 1 Class 2 Mode Register (C1C2MR) | 32 | RW | 0000_0000h |
| 9_040Ch | CCB 1 Class 2 Key Size Register (C1C2KSR) | 32 | RW | 0000_0000h |
| 9_0410h | CCB 1 Class 2 Data Size Register (C1C2DSR) | 64 | RW | 0000_0000_0000_0000h |
| 9_041Ch | CCB 1 Class 2 ICV Size Register (C1C2ICVSZR) | 32 | RW | 0000_0000h |
| 9_0500h | CCB 1 Class 2 Context Register Word 0 (C1C2CTXR0) | 32 | RW | 0000_0000h |
| 9_0504h | CCB 1 Class 2 Context Register Word 1 (C1C2CTXR1) | 32 | RW | 0000_0000h |
| 9_0508h | CCB 1 Class 2 Context Register Word 2 (C1C2CTXR2) | 32 | RW | 0000_0000h |
| 9_050Ch | CCB 1 Class 2 Context Register Word 3 (C1C2CTXR3) | 32 | RW | 0000_0000h |
| 9_0510h | CCB 1 Class 2 Context Register Word 4 (C1C2CTXR4) | 32 | RW | 0000_0000h |
| 9_0514h | CCB 1 Class 2 Context Register Word 5 (C1C2CTXR5) | 32 | RW | 0000_0000h |
| 9_0518h | CCB 1 Class 2 Context Register Word 6 (C1C2CTXR6) | 32 | RW | 0000_0000h |
| 9_051Ch | CCB 1 Class 2 Context Register Word 7 (C1C2CTXR7) | 32 | RW | 0000_0000h |
| 9_0520h | CCB 1 Class 2 Context Register Word 8 (C1C2CTXR8) | 32 | RW | 0000_0000h |
| 9_0524h | CCB 1 Class 2 Context Register Word 9 (C1C2CTXR9) | 32 | RW | 0000_0000h |
| 9_0528h | CCB 1 Class 2 Context Register Word 10 (C1C2CTXR10) | 32 | RW | 0000_0000h |
| 9_052Ch | CCB 1 Class 2 Context Register Word 11 (C1C2CTXR11) | 32 | RW | 0000_0000h |
| 9_0530h | CCB 1 Class 2 Context Register Word 12 (C1C2CTXR12) | 32 | RW | 0000_0000h |
| 9_0534h | CCB 1 Class 2 Context Register Word 13 (C1C2CTXR13) | 32 | RW | 0000_0000h |
| 9_0538h | CCB 1 Class 2 Context Register Word 14 (C1C2CTXR14) | 32 | RW | 0000_0000h |
| 9_053Ch | CCB 1 Class 2 Context Register Word 15 (C1C2CTXR15) | 32 | RW | 0000_0000h |
| 9_0540h | CCB 1 Class 2 Context Register Word 16 (C1C2CTXR16) | 32 | RW | 0000_0000h |
| 9_0544h | CCB 1 Class 2 Context Register Word 17 (C1C2CTXR17) | 32 | RW | 0000_0000h |
| 9_0600h | CCB 1 Class 2 Key Register Word 0 (C1C2KEYR0) | 32 | RW | 0000_0000h |
| 9_0604h | CCB 1 Class 2 Key Register Word 1 (C1C2KEYR1) | 32 | RW | 0000_0000h |
| 9_0608h | CCB 1 Class 2 Key Register Word 2 (C1C2KEYR2) | 32 | RW | 0000_0000h |
| 9_060Ch | CCB 1 Class 2 Key Register Word 3 (C1C2KEYR3) | 32 | RW | 0000_0000h |
| 9_0610h | CCB 1 Class 2 Key Register Word 4 (C1C2KEYR4) | 32 | RW | 0000_0000h |
| 9_0614h | CCB 1 Class 2 Key Register Word 5 (C1C2KEYR5) | 32 | RW | 0000_0000h |
| 9_0618h | CCB 1 Class 2 Key Register Word 6 (C1C2KEYR6) | 32 | RW | 0000_0000h |
| 9_061Ch | CCB 1 Class 2 Key Register Word 7 (C1C2KEYR7) | 32 | RW | 0000_0000h |
| 9_0620h | CCB 1 Class 2 Key Register Word 8 (C1C2KEYR8) | 32 | RW | 0000_0000h |
| 9_0624h | CCB 1 Class 2 Key Register Word 9 (C1C2KEYR9) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 9_0628h | CCB 1 Class 2 Key Register Word 10 (C1C2KEYR10) | 32 | RW | 0000_0000h |
| 9_062Ch | CCB 1 Class 2 Key Register Word 11 (C1C2KEYR11) | 32 | RW | 0000_0000h |
| 9_0630h | CCB 1 Class 2 Key Register Word 12 (C1C2KEYR12) | 32 | RW | 0000_0000h |
| 9_0634h | CCB 1 Class 2 Key Register Word 13 (C1C2KEYR13) | 32 | RW | 0000_0000h |
| 9_0638h | CCB 1 Class 2 Key Register Word 14 (C1C2KEYR14) | 32 | RW | 0000_0000h |
| 9_063Ch | CCB 1 Class 2 Key Register Word 15 (C1C2KEYR15) | 32 | RW | 0000_0000h |
| 9_0640h | CCB 1 Class 2 Key Register Word 16 (C1C2KEYR16) | 32 | RW | 0000_0000h |
| 9_0644h | CCB 1 Class 2 Key Register Word 17 (C1C2KEYR17) | 32 | RW | 0000_0000h |
| 9_0648h | CCB 1 Class 2 Key Register Word 18 (C1C2KEYR18) | 32 | RW | 0000_0000h |
| 9_064Ch | CCB 1 Class 2 Key Register Word 19 (C1C2KEYR19) | 32 | RW | 0000_0000h |
| 9_0650h | CCB 1 Class 2 Key Register Word 20 (C1C2KEYR20) | 32 | RW | 0000_0000h |
| 9_0654h | CCB 1 Class 2 Key Register Word 21 (C1C2KEYR21) | 32 | RW | 0000_0000h |
| 9_0658h | CCB 1 Class 2 Key Register Word 22 (C1C2KEYR22) | 32 | RW | 0000_0000h |
| 9_065Ch | CCB 1 Class 2 Key Register Word 23 (C1C2KEYR23) | 32 | RW | 0000_0000h |
| 9_0660h | CCB 1 Class 2 Key Register Word 24 (C1C2KEYR24) | 32 | RW | 0000_0000h |
| 9_0664h | CCB 1 Class 2 Key Register Word 25 (C1C2KEYR25) | 32 | RW | 0000_0000h |
| 9_0668h | CCB 1 Class 2 Key Register Word 26 (C1C2KEYR26) | 32 | RW | 0000_0000h |
| 9_066Ch | CCB 1 Class 2 Key Register Word 27 (C1C2KEYR27) | 32 | RW | 0000_0000h |
| 9_0670h | CCB 1 Class 2 Key Register Word 28 (C1C2KEYR28) | 32 | RW | 0000_0000h |
| 9_0674h | CCB 1 Class 2 Key Register Word 29 (C1C2KEYR29) | 32 | RW | 0000_0000h |
| 9_0678h | CCB 1 Class 2 Key Register Word 30 (C1C2KEYR30) | 32 | RW | 0000_0000h |
| 9_067Ch | CCB 1 Class 2 Key Register Word 31 (C1C2KEYR31) | 32 | RW | 0000_0000h |
| 9_07C0h | CCB 1 FIFO Status (C1FIFOSTA) | 32 | RO | 0000_0000h |
| 9_07D0h | CCB 1 iNformation FIFO When STYPE Is Not 10 (C1NFIFO) | 32 | WO | 0000_0000h |
| 9_07D0h | CCB 1 iNformation FIFO When STYPE Is 10 (C1NFIFO_2) | 32 | WO | 0000_0000h |
| 9_07E0h | CCB 1 Input Data FIFO (C1IFIFO) | 32 | WO | 0000_0000h |
| 9_07F0h | CCB 1 Output Data FIFO (C1OFIFO) | 64 | RO | 0000_0000_0000_0000h |
| 9_0800h | DECO1 Job Queue Control Register, most-significant half (D1JQCR_MS) | 32 | RW | 0000_0000h |
| 9_0804h | DECO1 Job Queue Control Register, least-significant half (D1JQCR_LS) | 32 | RO | 0000_0000h |
| 9_0808h | DECO1 Descriptor Address Register (D1DAR) | 64 | RO | 0000_0000_0000_0000h |
| 9_0810h | DECO1 Operation Status Register, most-significant half (D1OPSTA_MS) | 32 | RO | 0000_0000h |
| 9_0814h | DECO1 Operation Status Register, least-significant half (D1OPSTA_LS) | 32 | RO | 0000_0000h |
| 9_0818h | DECO1 Checksum Register (D1CKSUMR) | 32 | RW | 0000_0000h |
| 9_0820h | DECO1 ICID Status Register (D1ISR) | 32 | RO | 0000_0000h |
| 9_0820h | DECO1 SDID / Trusted ICID Status Register (D1SDIDSR) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 9_0840h | DECO1 Math Register 0_MS (D1MTH0_MS) | 32 | RW | 0000_0000h |
| 9_0844h | DECO1 Math Register 0_LS (D1MTH0_LS) | 32 | RW | 0000_0000h |
| 9_0848h | DECO1 Math Register 1_MS (D1MTH1_MS) | 32 | RW | 0000_0000h |
| 9_084Ch | DECO1 Math Register 1_LS (D1MTH1_LS) | 32 | RW | 0000_0000h |
| 9_0850h | DECO1 Math Register 2_MS (D1MTH2_MS) | 32 | RW | 0000_0000h |
| 9_0854h | DECO1 Math Register 2_LS (D1MTH2_LS) | 32 | RW | 0000_0000h |
| 9_0858h | DECO1 Math Register 3_MS (D1MTH3_MS) | 32 | RW | 0000_0000h |
| 9_085Ch | DECO1 Math Register 3_LS (D1MTH3_LS) | 32 | RW | 0000_0000h |
| 9_0860h | DECO1 Math Register 4_MS (D1MTH4_MS) | 32 | RW | 0000_0000h |
| 9_0864h | DECO1 Math Register 4_LS (D1MTH4_LS) | 32 | RW | 0000_0000h |
| 9_0868h | DECO1 Math Register 5_MS (D1MTH5_MS) | 32 | RW | 0000_0000h |
| 9_086Ch | DECO1 Math Register 5_LS (D1MTH5_LS) | 32 | RW | 0000_0000h |
| 9_0870h | DECO1 Math Register 6_MS (D1MTH6_MS) | 32 | RW | 0000_0000h |
| 9_0874h | DECO1 Math Register 6_LS (D1MTH6_LS) | 32 | RW | 0000_0000h |
| 9_0878h | DECO1 Math Register 7_MS (D1MTH7_MS) | 32 | RW | 0000_0000h |
| 9_087Ch | DECO1 Math Register 7_LS (D1MTH7_LS) | 32 | RW | 0000_0000h |
| 9_0880h | DECO1 Gather Table Register 0 Word 0 (D1GTR0_0) | 32 | RW | 0000_0000h |
| 9_0884h | DECO1 Gather Table Register 0 Word 1 (D1GTR0_1) | 32 | RW | 0000_0000h |
| 9_0888h | DECO1 Gather Table Register 0 Word 2 (D1GTR0_2) | 32 | RW | 0000_0000h |
| 9_088Ch | DECO1 Gather Table Register 0 Word 3 (D1GTR0_3) | 32 | RW | 0000_0000h |
| 9_0890h | DECO1 Gather Table Register 1 Word 0 (D1GTR1_0) | 32 | RW | 0000_0000h |
| 9_0894h | DECO1 Gather Table Register 1 Word 1 (D1GTR1_1) | 32 | RW | 0000_0000h |
| 9_0898h | DECO1 Gather Table Register 1 Word 2 (D1GTR1_2) | 32 | RW | 0000_0000h |
| 9_089Ch | DECO1 Gather Table Register 1 Word 3 (D1GTR1_3) | 32 | RW | 0000_0000h |
| 9_08A0h | DECO1 Gather Table Register 2 Word 0 (D1GTR2_0) | 32 | RW | 0000_0000h |
| 9_08A4h | DECO1 Gather Table Register 2 Word 1 (D1GTR2_1) | 32 | RW | 0000_0000h |
| 9_08A8h | DECO1 Gather Table Register 2 Word 2 (D1GTR2_2) | 32 | RW | 0000_0000h |
| 9_08ACh | DECO1 Gather Table Register 2 Word 3 (D1GTR2_3) | 32 | RW | 0000_0000h |
| 9_08B0h | DECO1 Gather Table Register 3 Word 0 (D1GTR3_0) | 32 | RW | 0000_0000h |
| 9_08B4h | DECO1 Gather Table Register 3 Word 1 (D1GTR3_1) | 32 | RW | 0000_0000h |
| 9_08B8h | DECO1 Gather Table Register 3 Word 2 (D1GTR3_2) | 32 | RW | 0000_0000h |
| 9_08BCh | DECO1 Gather Table Register 3 Word 3 (D1GTR3_3) | 32 | RW | 0000_0000h |
| 9_0900h | DECO1 Scatter Table Register 0 Word 0 (D1STR0_0) | 32 | RW | 0000_0000h |
| 9_0904h | DECO1 Scatter Table Register 0 Word 1 (D1STR0_1) | 32 | RW | 0000_0000h |
| 9_0908h | DECO1 Scatter Table Register 0 Word 2 (D1STR0_2) | 32 | RW | 0000_0000h |
| 9_090Ch | DECO1 Scatter Table Register 0 Word 3 (D1STR0_3) | 32 | RW | 0000_0000h |
| 9_0910h | DECO1 Scatter Table Register 1 Word 0 (D1STR1_0) | 32 | RW | 0000_0000h |
| 9_0914h | DECO1 Scatter Table Register 1 Word 1 (D1STR1_1) | 32 | RW | 0000_0000h |
| 9_0918h | DECO1 Scatter Table Register 1 Word 2 (D1STR1_2) | 32 | RW | 0000_0000h |
| 9_091Ch | DECO1 Scatter Table Register 1 Word 3 (D1STR1_3) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 9_0920h | DECO1 Scatter Table Register 2 Word 0 (D1STR2_0) | 32 | RW | 0000_0000h |
| 9_0924h | DECO1 Scatter Table Register 2 Word 1 (D1STR2_1) | 32 | RW | 0000_0000h |
| 9_0928h | DECO1 Scatter Table Register 2 Word 2 (D1STR2_2) | 32 | RW | 0000_0000h |
| 9_092Ch | DECO1 Scatter Table Register 2 Word 3 (D1STR2_3) | 32 | RW | 0000_0000h |
| 9_0930h | DECO1 Scatter Table Register 3 Word 0 (D1STR3_0) | 32 | RW | 0000_0000h |
| 9_0934h | DECO1 Scatter Table Register 3 Word 1 (D1STR3_1) | 32 | RW | 0000_0000h |
| 9_0938h | DECO1 Scatter Table Register 3 Word 2 (D1STR3_2) | 32 | RW | 0000_0000h |
| 9_093Ch | DECO1 Scatter Table Register 3 Word 3 (D1STR3_3) | 32 | RW | 0000_0000h |
| 9_0A00h | DECO1 Descriptor Buffer Word 0 (D1DESB0) | 32 | RW | 0000_0000h |
| 9_0A04h | DECO1 Descriptor Buffer Word 1 (D1DESB1) | 32 | RW | 0000_0000h |
| 9_0A08h | DECO1 Descriptor Buffer Word 2 (D1DESB2) | 32 | RW | 0000_0000h |
| 9_0A0Ch | DECO1 Descriptor Buffer Word 3 (D1DESB3) | 32 | RW | 0000_0000h |
| 9_0A10h | DECO1 Descriptor Buffer Word 4 (D1DESB4) | 32 | RW | 0000_0000h |
| 9_0A14h | DECO1 Descriptor Buffer Word 5 (D1DESB5) | 32 | RW | 0000_0000h |
| 9_0A18h | DECO1 Descriptor Buffer Word 6 (D1DESB6) | 32 | RW | 0000_0000h |
| 9_0A1Ch | DECO1 Descriptor Buffer Word 7 (D1DESB7) | 32 | RW | 0000_0000h |
| 9_0A20h | DECO1 Descriptor Buffer Word 8 (D1DESB8) | 32 | RW | 0000_0000h |
| 9_0A24h | DECO1 Descriptor Buffer Word 9 (D1DESB9) | 32 | RW | 0000_0000h |
| 9_0A28h | DECO1 Descriptor Buffer Word 10 (D1DESB10) | 32 | RW | 0000_0000h |
| 9_0A2Ch | DECO1 Descriptor Buffer Word 11 (D1DESB11) | 32 | RW | 0000_0000h |
| 9_0A30h | DECO1 Descriptor Buffer Word 12 (D1DESB12) | 32 | RW | 0000_0000h |
| 9_0A34h | DECO1 Descriptor Buffer Word 13 (D1DESB13) | 32 | RW | 0000_0000h |
| 9_0A38h | DECO1 Descriptor Buffer Word 14 (D1DESB14) | 32 | RW | 0000_0000h |
| 9_0A3Ch | DECO1 Descriptor Buffer Word 15 (D1DESB15) | 32 | RW | 0000_0000h |
| 9_0A40h | DECO1 Descriptor Buffer Word 16 (D1DESB16) | 32 | RW | 0000_0000h |
| 9_0A44h | DECO1 Descriptor Buffer Word 17 (D1DESB17) | 32 | RW | 0000_0000h |
| 9_0A48h | DECO1 Descriptor Buffer Word 18 (D1DESB18) | 32 | RW | 0000_0000h |
| 9_0A4Ch | DECO1 Descriptor Buffer Word 19 (D1DESB19) | 32 | RW | 0000_0000h |
| 9_0A50h | DECO1 Descriptor Buffer Word 20 (D1DESB20) | 32 | RW | 0000_0000h |
| 9_0A54h | DECO1 Descriptor Buffer Word 21 (D1DESB21) | 32 | RW | 0000_0000h |
| 9_0A58h | DECO1 Descriptor Buffer Word 22 (D1DESB22) | 32 | RW | 0000_0000h |
| 9_0A5Ch | DECO1 Descriptor Buffer Word 23 (D1DESB23) | 32 | RW | 0000_0000h |
| 9_0A60h | DECO1 Descriptor Buffer Word 24 (D1DESB24) | 32 | RW | 0000_0000h |
| 9_0A64h | DECO1 Descriptor Buffer Word 25 (D1DESB25) | 32 | RW | 0000_0000h |
| 9_0A68h | DECO1 Descriptor Buffer Word 26 (D1DESB26) | 32 | RW | 0000_0000h |
| 9_0A6Ch | DECO1 Descriptor Buffer Word 27 (D1DESB27) | 32 | RW | 0000_0000h |
| 9_0A70h | DECO1 Descriptor Buffer Word 28 (D1DESB28) | 32 | RW | 0000_0000h |
| 9_0A74h | DECO1 Descriptor Buffer Word 29 (D1DESB29) | 32 | RW | 0000_0000h |
| 9_0A78h | DECO1 Descriptor Buffer Word 30 (D1DESB30) | 32 | RW | 0000_0000h |
| 9_0A7Ch | DECO1 Descriptor Buffer Word 31 (D1DESB31) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 9_0A80h | DECO1 Descriptor Buffer Word 32 (D1DESB32) | 32 | RW | 0000_0000h |
| 9_0A84h | DECO1 Descriptor Buffer Word 33 (D1DESB33) | 32 | RW | 0000_0000h |
| 9_0A88h | DECO1 Descriptor Buffer Word 34 (D1DESB34) | 32 | RW | 0000_0000h |
| 9_0A8Ch | DECO1 Descriptor Buffer Word 35 (D1DESB35) | 32 | RW | 0000_0000h |
| 9_0A90h | DECO1 Descriptor Buffer Word 36 (D1DESB36) | 32 | RW | 0000_0000h |
| 9_0A94h | DECO1 Descriptor Buffer Word 37 (D1DESB37) | 32 | RW | 0000_0000h |
| 9_0A98h | DECO1 Descriptor Buffer Word 38 (D1DESB38) | 32 | RW | 0000_0000h |
| 9_0A9Ch | DECO1 Descriptor Buffer Word 39 (D1DESB39) | 32 | RW | 0000_0000h |
| 9_0AA0h | DECO1 Descriptor Buffer Word 40 (D1DESB40) | 32 | RW | 0000_0000h |
| 9_0AA4h | DECO1 Descriptor Buffer Word 41 (D1DESB41) | 32 | RW | 0000_0000h |
| 9_0AA8h | DECO1 Descriptor Buffer Word 42 (D1DESB42) | 32 | RW | 0000_0000h |
| 9_0AACh | DECO1 Descriptor Buffer Word 43 (D1DESB43) | 32 | RW | 0000_0000h |
| 9_0AB0h | DECO1 Descriptor Buffer Word 44 (D1DESB44) | 32 | RW | 0000_0000h |
| 9_0AB4h | DECO1 Descriptor Buffer Word 45 (D1DESB45) | 32 | RW | 0000_0000h |
| 9_0AB8h | DECO1 Descriptor Buffer Word 46 (D1DESB46) | 32 | RW | 0000_0000h |
| 9_0ABCh | DECO1 Descriptor Buffer Word 47 (D1DESB47) | 32 | RW | 0000_0000h |
| 9_0AC0h | DECO1 Descriptor Buffer Word 48 (D1DESB48) | 32 | RW | 0000_0000h |
| 9_0AC4h | DECO1 Descriptor Buffer Word 49 (D1DESB49) | 32 | RW | 0000_0000h |
| 9_0AC8h | DECO1 Descriptor Buffer Word 50 (D1DESB50) | 32 | RW | 0000_0000h |
| 9_0ACCh | DECO1 Descriptor Buffer Word 51 (D1DESB51) | 32 | RW | 0000_0000h |
| 9_0AD0h | DECO1 Descriptor Buffer Word 52 (D1DESB52) | 32 | RW | 0000_0000h |
| 9_0AD4h | DECO1 Descriptor Buffer Word 53 (D1DESB53) | 32 | RW | 0000_0000h |
| 9_0AD8h | DECO1 Descriptor Buffer Word 54 (D1DESB54) | 32 | RW | 0000_0000h |
| 9_0ADCh | DECO1 Descriptor Buffer Word 55 (D1DESB55) | 32 | RW | 0000_0000h |
| 9_0AE0h | DECO1 Descriptor Buffer Word 56 (D1DESB56) | 32 | RW | 0000_0000h |
| 9_0AE4h | DECO1 Descriptor Buffer Word 57 (D1DESB57) | 32 | RW | 0000_0000h |
| 9_0AE8h | DECO1 Descriptor Buffer Word 58 (D1DESB58) | 32 | RW | 0000_0000h |
| 9_0AECh | DECO1 Descriptor Buffer Word 59 (D1DESB59) | 32 | RW | 0000_0000h |
| 9_0AF0h | DECO1 Descriptor Buffer Word 60 (D1DESB60) | 32 | RW | 0000_0000h |
| 9_0AF4h | DECO1 Descriptor Buffer Word 61 (D1DESB61) | 32 | RW | 0000_0000h |
| 9_0AF8h | DECO1 Descriptor Buffer Word 62 (D1DESB62) | 32 | RW | 0000_0000h |
| 9_0AFCh | DECO1 Descriptor Buffer Word 63 (D1DESB63) | 32 | RW | 0000_0000h |
| 9_0E00h | DECO1 Debug Job (D1DJR) | 32 | RO | 0000_0000h |
| 9_0E04h | DECO1 Debug DECO (D1DDR) | 32 | RO | 0000_0000h |
| 9_0E08h | DECO1 Debug Job Pointer (D1DJP) | 64 | RO | 0000_0000_0000_0000h |
| 9_0E10h | DECO1 Debug Shared Pointer (D1SDP) | 64 | RO | 0000_0000_0000_0000h |
| 9_0E18h | DECO1 Debug_ICID, most-significant half (D1DIR_MS) | 32 | RO | 0000_0000h |
| 9_0E20h | Sequence Output Length Register (SOL1) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width (In bits) | Access | Reset value |
|---|---|---|---|---|
| 9_0E24h | Variable Sequence Output Length Register (VSOL1) | 32 | RW | 0000_0000h |
| 9_0E28h | Sequence Input Length Register (SIL1) | 32 | RW | 0000_0000h |
| 9_0E2Ch | Variable Sequence Input Length Register (VSIL1) | 32 | RW | 0000_0000h |
| 9_0E30h | Protocol Override Register (D1POVRD) | 32 | RW | 0000_0000h |
| 9_0E34h | Variable Sequence Output Length Register; Upper 32 bits (UVSOL1) | 32 | RW | 0000_0000h |
| 9_0E38h | Variable Sequence Input Length Register; Upper 32 bits (UVSIL1) | 32 | RW | 0000_0000h |
| 9_0F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |
| 9_0F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 9_0F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| 9_0F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 9_0F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |
| 9_0F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| 9_0F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| 9_0FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| 9_0FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| 9_0FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| 9_0FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| 9_0FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |
| 9_0FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |
| 9_0FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |
| 9_0FD4h (alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |
| 9_0FE0h (alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| 9_0FE4h (alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |
| 9_0FE8h (alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| 9_0FECh (alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| 9_0FF0h (alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| 9_0FF4h (alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |
| 9_0FF8h (alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| 9_0FFCh (alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |
| A_0004h | CCB 2 Class 1 Mode Register Format for Non-Public Key Algorithms (C2C1MR_NPK) | 32 | RW | 0000_0000h |
| A_0004h | CCB 2 Class 1 Mode Register Format for Public Key Algorithms (C2C1MR_PK) | 32 | RW | 0000_0000h |
| A_0004h | CCB 2 Class 1 Mode Register Format for RNG4 (C2C1MR_RNG) | 32 | RW | 0000_0000h |
| A_000Ch | CCB 2 Class 1 Key Size Register (C2C1KSR) | 32 | RW | 0000_0000h |
| A_0010h | CCB 2 Class 1 Data Size Register (C2C1DSR) | 64 | RW | 0000_0000_0000_0000h |
| A_001Ch | CCB 2 Class 1 ICV Size Register (C2C1ICVSR) | 32 | RW | 0000_0000h |
| A_0034h | CCB 2 CHA Control Register (C2CCTRL) | 32 | WO | 0000_0000h |
| A_003Ch | CCB 2 Interrupt Control Register (C2ICTL) | 32 | W1C | 0000_0000h |
| A_0044h | CCB 2 Clear Written Register (C2CWR) | 32 | WO | 0000_0000h |
| A_0048h | CCB 2 Status and Error Register, most-significant half (C2CSTA_MS) | 32 | RO | 0000_0000h |
| A_004Ch | CCB 2 Status and Error Register, least-significant half (C2CSTA_LS) | 32 | RO | 0000_0000h |
| A_005Ch | CCB 2 AAD Size Register (C2AADSZR) | 32 | RW | 0000_0000h |
| A_0064h | Class 1 IV Size Register (C2C1IVSZR) | 32 | RW | 0000_0000h |
| A_0084h | PKHA A Size Register (C2PKASZR) | 32 | RW | 0000_0000h |
| A_008Ch | PKHA B Size Register (C2PKBSZR) | 32 | RW | 0000_0000h |
| A_0094h | PKHA N Size Register (C2PKNSZR) | 32 | RW | 0000_0000h |
| A_009Ch | PKHA E Size Register (C2PKESZR) | 32 | RW | 0000_0000h |
| A_0100h | CCB 2 Class 1 Context Register Word 0 (C2C1CTXR0) | 32 | RW | 0000_0000h |
| A_0104h | CCB 2 Class 1 Context Register Word 1 (C2C1CTXR1) | 32 | RW | 0000_0000h |
| A_0108h | CCB 2 Class 1 Context Register Word 2 (C2C1CTXR2) | 32 | RW | 0000_0000h |
| A_010Ch | CCB 2 Class 1 Context Register Word 3 (C2C1CTXR3) | 32 | RW | 0000_0000h |
| A_0110h | CCB 2 Class 1 Context Register Word 4 (C2C1CTXR4) | 32 | RW | 0000_0000h |
| A_0114h | CCB 2 Class 1 Context Register Word 5 (C2C1CTXR5) | 32 | RW | 0000_0000h |
| A_0118h | CCB 2 Class 1 Context Register Word 6 (C2C1CTXR6) | 32 | RW | 0000_0000h |
| A_011Ch | CCB 2 Class 1 Context Register Word 7 (C2C1CTXR7) | 32 | RW | 0000_0000h |
| A_0120h | CCB 2 Class 1 Context Register Word 8 (C2C1CTXR8) | 32 | RW | 0000_0000h |
| A_0124h | CCB 2 Class 1 Context Register Word 9 (C2C1CTXR9) | 32 | RW | 0000_0000h |
| A_0128h | CCB 2 Class 1 Context Register Word 10 (C2C1CTXR10) | 32 | RW | 0000_0000h |
| A_012Ch | CCB 2 Class 1 Context Register Word 11 (C2C1CTXR11) | 32 | RW | 0000_0000h |
| A_0130h | CCB 2 Class 1 Context Register Word 12 (C2C1CTXR12) | 32 | RW | 0000_0000h |
| A_0134h | CCB 2 Class 1 Context Register Word 13 (C2C1CTXR13) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| A_0138h | CCB 2 Class 1 Context Register Word 14 (C2C1CTXR14) | 32 | RW | 0000_0000h |
| A_013Ch | CCB 2 Class 1 Context Register Word 15 (C2C1CTXR15) | 32 | RW | 0000_0000h |
| A_0200h | CCB 2 Class 1 Key Registers Word 0 (C2C1KR0) | 32 | RW | 0000_0000h |
| A_0204h | CCB 2 Class 1 Key Registers Word 1 (C2C1KR1) | 32 | RW | 0000_0000h |
| A_0208h | CCB 2 Class 1 Key Registers Word 2 (C2C1KR2) | 32 | RW | 0000_0000h |
| A_020Ch | CCB 2 Class 1 Key Registers Word 3 (C2C1KR3) | 32 | RW | 0000_0000h |
| A_0210h | CCB 2 Class 1 Key Registers Word 4 (C2C1KR4) | 32 | RW | 0000_0000h |
| A_0214h | CCB 2 Class 1 Key Registers Word 5 (C2C1KR5) | 32 | RW | 0000_0000h |
| A_0218h | CCB 2 Class 1 Key Registers Word 6 (C2C1KR6) | 32 | RW | 0000_0000h |
| A_021Ch | CCB 2 Class 1 Key Registers Word 7 (C2C1KR7) | 32 | RW | 0000_0000h |
| A_0404h | CCB 2 Class 2 Mode Register (C2C2MR) | 32 | RW | 0000_0000h |
| A_040Ch | CCB 2 Class 2 Key Size Register (C2C2KSR) | 32 | RW | 0000_0000h |
| A_0410h | CCB 2 Class 2 Data Size Register (C2C2DSR) | 64 | RW | 0000_0000_0000_0000h |
| A_041Ch | CCB 2 Class 2 ICV Size Register (C2C2ICVSZR) | 32 | RW | 0000_0000h |
| A_0500h | CCB 2 Class 2 Context Register Word 0 (C2C2CTXR0) | 32 | RW | 0000_0000h |
| A_0504h | CCB 2 Class 2 Context Register Word 1 (C2C2CTXR1) | 32 | RW | 0000_0000h |
| A_0508h | CCB 2 Class 2 Context Register Word 2 (C2C2CTXR2) | 32 | RW | 0000_0000h |
| A_050Ch | CCB 2 Class 2 Context Register Word 3 (C2C2CTXR3) | 32 | RW | 0000_0000h |
| A_0510h | CCB 2 Class 2 Context Register Word 4 (C2C2CTXR4) | 32 | RW | 0000_0000h |
| A_0514h | CCB 2 Class 2 Context Register Word 5 (C2C2CTXR5) | 32 | RW | 0000_0000h |
| A_0518h | CCB 2 Class 2 Context Register Word 6 (C2C2CTXR6) | 32 | RW | 0000_0000h |
| A_051Ch | CCB 2 Class 2 Context Register Word 7 (C2C2CTXR7) | 32 | RW | 0000_0000h |
| A_0520h | CCB 2 Class 2 Context Register Word 8 (C2C2CTXR8) | 32 | RW | 0000_0000h |
| A_0524h | CCB 2 Class 2 Context Register Word 9 (C2C2CTXR9) | 32 | RW | 0000_0000h |
| A_0528h | CCB 2 Class 2 Context Register Word 10 (C2C2CTXR10) | 32 | RW | 0000_0000h |
| A_052Ch | CCB 2 Class 2 Context Register Word 11 (C2C2CTXR11) | 32 | RW | 0000_0000h |
| A_0530h | CCB 2 Class 2 Context Register Word 12 (C2C2CTXR12) | 32 | RW | 0000_0000h |
| A_0534h | CCB 2 Class 2 Context Register Word 13 (C2C2CTXR13) | 32 | RW | 0000_0000h |
| A_0538h | CCB 2 Class 2 Context Register Word 14 (C2C2CTXR14) | 32 | RW | 0000_0000h |
| A_053Ch | CCB 2 Class 2 Context Register Word 15 (C2C2CTXR15) | 32 | RW | 0000_0000h |
| A_0540h | CCB 2 Class 2 Context Register Word 16 (C2C2CTXR16) | 32 | RW | 0000_0000h |
| A_0544h | CCB 2 Class 2 Context Register Word 17 (C2C2CTXR17) | 32 | RW | 0000_0000h |
| A_0600h | CCB 2 Class 2 Key Register Word 0 (C2C2KEYR0) | 32 | RW | 0000_0000h |
| A_0604h | CCB 2 Class 2 Key Register Word 1 (C2C2KEYR1) | 32 | RW | 0000_0000h |
| A_0608h | CCB 2 Class 2 Key Register Word 2 (C2C2KEYR2) | 32 | RW | 0000_0000h |
| A_060Ch | CCB 2 Class 2 Key Register Word 3 (C2C2KEYR3) | 32 | RW | 0000_0000h |
| A_0610h | CCB 2 Class 2 Key Register Word 4 (C2C2KEYR4) | 32 | RW | 0000_0000h |
| A_0614h | CCB 2 Class 2 Key Register Word 5 (C2C2KEYR5) | 32 | RW | 0000_0000h |
| A_0618h | CCB 2 Class 2 Key Register Word 6 (C2C2KEYR6) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| A_061Ch | CCB 2 Class 2 Key Register Word 7 (C2C2KEYR7) | 32 | RW | 0000_0000h |
| A_0620h | CCB 2 Class 2 Key Register Word 8 (C2C2KEYR8) | 32 | RW | 0000_0000h |
| A_0624h | CCB 2 Class 2 Key Register Word 9 (C2C2KEYR9) | 32 | RW | 0000_0000h |
| A_0628h | CCB 2 Class 2 Key Register Word 10 (C2C2KEYR10) | 32 | RW | 0000_0000h |
| A_062Ch | CCB 2 Class 2 Key Register Word 11 (C2C2KEYR11) | 32 | RW | 0000_0000h |
| A_0630h | CCB 2 Class 2 Key Register Word 12 (C2C2KEYR12) | 32 | RW | 0000_0000h |
| A_0634h | CCB 2 Class 2 Key Register Word 13 (C2C2KEYR13) | 32 | RW | 0000_0000h |
| A_0638h | CCB 2 Class 2 Key Register Word 14 (C2C2KEYR14) | 32 | RW | 0000_0000h |
| A_063Ch | CCB 2 Class 2 Key Register Word 15 (C2C2KEYR15) | 32 | RW | 0000_0000h |
| A_0640h | CCB 2 Class 2 Key Register Word 16 (C2C2KEYR16) | 32 | RW | 0000_0000h |
| A_0644h | CCB 2 Class 2 Key Register Word 17 (C2C2KEYR17) | 32 | RW | 0000_0000h |
| A_0648h | CCB 2 Class 2 Key Register Word 18 (C2C2KEYR18) | 32 | RW | 0000_0000h |
| A_064Ch | CCB 2 Class 2 Key Register Word 19 (C2C2KEYR19) | 32 | RW | 0000_0000h |
| A_0650h | CCB 2 Class 2 Key Register Word 20 (C2C2KEYR20) | 32 | RW | 0000_0000h |
| A_0654h | CCB 2 Class 2 Key Register Word 21 (C2C2KEYR21) | 32 | RW | 0000_0000h |
| A_0658h | CCB 2 Class 2 Key Register Word 22 (C2C2KEYR22) | 32 | RW | 0000_0000h |
| A_065Ch | CCB 2 Class 2 Key Register Word 23 (C2C2KEYR23) | 32 | RW | 0000_0000h |
| A_0660h | CCB 2 Class 2 Key Register Word 24 (C2C2KEYR24) | 32 | RW | 0000_0000h |
| A_0664h | CCB 2 Class 2 Key Register Word 25 (C2C2KEYR25) | 32 | RW | 0000_0000h |
| A_0668h | CCB 2 Class 2 Key Register Word 26 (C2C2KEYR26) | 32 | RW | 0000_0000h |
| A_066Ch | CCB 2 Class 2 Key Register Word 27 (C2C2KEYR27) | 32 | RW | 0000_0000h |
| A_0670h | CCB 2 Class 2 Key Register Word 28 (C2C2KEYR28) | 32 | RW | 0000_0000h |
| A_0674h | CCB 2 Class 2 Key Register Word 29 (C2C2KEYR29) | 32 | RW | 0000_0000h |
| A_0678h | CCB 2 Class 2 Key Register Word 30 (C2C2KEYR30) | 32 | RW | 0000_0000h |
| A_067Ch | CCB 2 Class 2 Key Register Word 31 (C2C2KEYR31) | 32 | RW | 0000_0000h |
| A_07C0h | CCB 2 FIFO Status (C2FIFOSTA) | 32 | RO | 0000_0000h |
| A_07D0h | CCB 2 iNformation FIFO When STYPE Is Not 10 (C2NFIFO) | 32 | WO | 0000_0000h |
| A_07D0h | CCB 2 iNformation FIFO When STYPE Is 10 (C2NFIFO_2) | 32 | WO | 0000_0000h |
| A_07E0h | CCB 2 Input Data FIFO (C2IFIFO) | 32 | WO | 0000_0000h |
| A_07F0h | CCB 2 Output Data FIFO (C2OFIFO) | 64 | RO | 0000_0000_0000_0000h |
| A_0800h | DECO2 Job Queue Control Register, most-significant half (D2JQCR_MS) | 32 | RW | 0000_0000h |
| A_0804h | DECO2 Job Queue Control Register, least-significant half (D2JQCR_LS) | 32 | RO | 0000_0000h |
| A_0808h | DECO2 Descriptor Address Register (D2DAR) | 64 | RO | 0000_0000_0000_0000h |
| A_0810h | DECO2 Operation Status Register, most-significant half (D2OPSTA_MS) | 32 | RO | 0000_0000h |
| A_0814h | DECO2 Operation Status Register, least-significant half (D2OPSTA_LS) | 32 | RO | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| A_0818h | DECO2 Checksum Register (D2CKSUMR) | 32 | RW | 0000_0000h |
| A_0820h | DECO2 ICID Status Register (D2ISR) | 32 | RO | 0000_0000h |
| A_0820h | DECO2 SDID / Trusted ICID Status Register (D2SDIDSR) | 32 | RW | 0000_0000h |
| A_0840h | DECO2 Math Register 0_MS (D2MTH0_MS) | 32 | RW | 0000_0000h |
| A_0844h | DECO2 Math Register 0_LS (D2MTH0_LS) | 32 | RW | 0000_0000h |
| A_0848h | DECO2 Math Register 1_MS (D2MTH1_MS) | 32 | RW | 0000_0000h |
| A_084Ch | DECO2 Math Register 1_LS (D2MTH1_LS) | 32 | RW | 0000_0000h |
| A_0850h | DECO2 Math Register 2_MS (D2MTH2_MS) | 32 | RW | 0000_0000h |
| A_0854h | DECO2 Math Register 2_LS (D2MTH2_LS) | 32 | RW | 0000_0000h |
| A_0858h | DECO2 Math Register 3_MS (D2MTH3_MS) | 32 | RW | 0000_0000h |
| A_085Ch | DECO2 Math Register 3_LS (D2MTH3_LS) | 32 | RW | 0000_0000h |
| A_0860h | DECO2 Math Register 4_MS (D2MTH4_MS) | 32 | RW | 0000_0000h |
| A_0864h | DECO2 Math Register 4_LS (D2MTH4_LS) | 32 | RW | 0000_0000h |
| A_0868h | DECO2 Math Register 5_MS (D2MTH5_MS) | 32 | RW | 0000_0000h |
| A_086Ch | DECO2 Math Register 5_LS (D2MTH5_LS) | 32 | RW | 0000_0000h |
| A_0870h | DECO2 Math Register 6_MS (D2MTH6_MS) | 32 | RW | 0000_0000h |
| A_0874h | DECO2 Math Register 6_LS (D2MTH6_LS) | 32 | RW | 0000_0000h |
| A_0878h | DECO2 Math Register 7_MS (D2MTH7_MS) | 32 | RW | 0000_0000h |
| A_087Ch | DECO2 Math Register 7_LS (D2MTH7_LS) | 32 | RW | 0000_0000h |
| A_0880h | DECO2 Gather Table Register 0 Word 0 (D2GTR0_0) | 32 | RW | 0000_0000h |
| A_0884h | DECO2 Gather Table Register 0 Word 1 (D2GTR0_1) | 32 | RW | 0000_0000h |
| A_0888h | DECO2 Gather Table Register 0 Word 2 (D2GTR0_2) | 32 | RW | 0000_0000h |
| A_088Ch | DECO2 Gather Table Register 0 Word 3 (D2GTR0_3) | 32 | RW | 0000_0000h |
| A_0890h | DECO2 Gather Table Register 1 Word 0 (D2GTR1_0) | 32 | RW | 0000_0000h |
| A_0894h | DECO2 Gather Table Register 1 Word 1 (D2GTR1_1) | 32 | RW | 0000_0000h |
| A_0898h | DECO2 Gather Table Register 1 Word 2 (D2GTR1_2) | 32 | RW | 0000_0000h |
| A_089Ch | DECO2 Gather Table Register 1 Word 3 (D2GTR1_3) | 32 | RW | 0000_0000h |
| A_08A0h | DECO2 Gather Table Register 2 Word 0 (D2GTR2_0) | 32 | RW | 0000_0000h |
| A_08A4h | DECO2 Gather Table Register 2 Word 1 (D2GTR2_1) | 32 | RW | 0000_0000h |
| A_08A8h | DECO2 Gather Table Register 2 Word 2 (D2GTR2_2) | 32 | RW | 0000_0000h |
| A_08ACh | DECO2 Gather Table Register 2 Word 3 (D2GTR2_3) | 32 | RW | 0000_0000h |
| A_08B0h | DECO2 Gather Table Register 3 Word 0 (D2GTR3_0) | 32 | RW | 0000_0000h |
| A_08B4h | DECO2 Gather Table Register 3 Word 1 (D2GTR3_1) | 32 | RW | 0000_0000h |
| A_08B8h | DECO2 Gather Table Register 3 Word 2 (D2GTR3_2) | 32 | RW | 0000_0000h |
| A_08BCh | DECO2 Gather Table Register 3 Word 3 (D2GTR3_3) | 32 | RW | 0000_0000h |
| A_0900h | DECO2 Scatter Table Register 0 Word 0 (D2STR0_0) | 32 | RW | 0000_0000h |
| A_0904h | DECO2 Scatter Table Register 0 Word 1 (D2STR0_1) | 32 | RW | 0000_0000h |
| A_0908h | DECO2 Scatter Table Register 0 Word 2 (D2STR0_2) | 32 | RW | 0000_0000h |
| A_090Ch | DECO2 Scatter Table Register 0 Word 3 (D2STR0_3) | 32 | RW | 0000_0000h |
| A_0910h | DECO2 Scatter Table Register 1 Word 0 (D2STR1_0) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| A_0914h | DECO2 Scatter Table Register 1 Word 1 (D2STR1_1) | 32 | RW | 0000_0000h |
| A_0918h | DECO2 Scatter Table Register 1 Word 2 (D2STR1_2) | 32 | RW | 0000_0000h |
| A_091Ch | DECO2 Scatter Table Register 1 Word 3 (D2STR1_3) | 32 | RW | 0000_0000h |
| A_0920h | DECO2 Scatter Table Register 2 Word 0 (D2STR2_0) | 32 | RW | 0000_0000h |
| A_0924h | DECO2 Scatter Table Register 2 Word 1 (D2STR2_1) | 32 | RW | 0000_0000h |
| A_0928h | DECO2 Scatter Table Register 2 Word 2 (D2STR2_2) | 32 | RW | 0000_0000h |
| A_092Ch | DECO2 Scatter Table Register 2 Word 3 (D2STR2_3) | 32 | RW | 0000_0000h |
| A_0930h | DECO2 Scatter Table Register 3 Word 0 (D2STR3_0) | 32 | RW | 0000_0000h |
| A_0934h | DECO2 Scatter Table Register 3 Word 1 (D2STR3_1) | 32 | RW | 0000_0000h |
| A_0938h | DECO2 Scatter Table Register 3 Word 2 (D2STR3_2) | 32 | RW | 0000_0000h |
| A_093Ch | DECO2 Scatter Table Register 3 Word 3 (D2STR3_3) | 32 | RW | 0000_0000h |
| A_0A00h | DECO2 Descriptor Buffer Word 0 (D2DESB0) | 32 | RW | 0000_0000h |
| A_0A04h | DECO2 Descriptor Buffer Word 1 (D2DESB1) | 32 | RW | 0000_0000h |
| A_0A08h | DECO2 Descriptor Buffer Word 2 (D2DESB2) | 32 | RW | 0000_0000h |
| A_0A0Ch | DECO2 Descriptor Buffer Word 3 (D2DESB3) | 32 | RW | 0000_0000h |
| A_0A10h | DECO2 Descriptor Buffer Word 4 (D2DESB4) | 32 | RW | 0000_0000h |
| A_0A14h | DECO2 Descriptor Buffer Word 5 (D2DESB5) | 32 | RW | 0000_0000h |
| A_0A18h | DECO2 Descriptor Buffer Word 6 (D2DESB6) | 32 | RW | 0000_0000h |
| A_0A1Ch | DECO2 Descriptor Buffer Word 7 (D2DESB7) | 32 | RW | 0000_0000h |
| A_0A20h | DECO2 Descriptor Buffer Word 8 (D2DESB8) | 32 | RW | 0000_0000h |
| A_0A24h | DECO2 Descriptor Buffer Word 9 (D2DESB9) | 32 | RW | 0000_0000h |
| A_0A28h | DECO2 Descriptor Buffer Word 10 (D2DESB10) | 32 | RW | 0000_0000h |
| A_0A2Ch | DECO2 Descriptor Buffer Word 11 (D2DESB11) | 32 | RW | 0000_0000h |
| A_0A30h | DECO2 Descriptor Buffer Word 12 (D2DESB12) | 32 | RW | 0000_0000h |
| A_0A34h | DECO2 Descriptor Buffer Word 13 (D2DESB13) | 32 | RW | 0000_0000h |
| A_0A38h | DECO2 Descriptor Buffer Word 14 (D2DESB14) | 32 | RW | 0000_0000h |
| A_0A3Ch | DECO2 Descriptor Buffer Word 15 (D2DESB15) | 32 | RW | 0000_0000h |
| A_0A40h | DECO2 Descriptor Buffer Word 16 (D2DESB16) | 32 | RW | 0000_0000h |
| A_0A44h | DECO2 Descriptor Buffer Word 17 (D2DESB17) | 32 | RW | 0000_0000h |
| A_0A48h | DECO2 Descriptor Buffer Word 18 (D2DESB18) | 32 | RW | 0000_0000h |
| A_0A4Ch | DECO2 Descriptor Buffer Word 19 (D2DESB19) | 32 | RW | 0000_0000h |
| A_0A50h | DECO2 Descriptor Buffer Word 20 (D2DESB20) | 32 | RW | 0000_0000h |
| A_0A54h | DECO2 Descriptor Buffer Word 21 (D2DESB21) | 32 | RW | 0000_0000h |
| A_0A58h | DECO2 Descriptor Buffer Word 22 (D2DESB22) | 32 | RW | 0000_0000h |
| A_0A5Ch | DECO2 Descriptor Buffer Word 23 (D2DESB23) | 32 | RW | 0000_0000h |
| A_0A60h | DECO2 Descriptor Buffer Word 24 (D2DESB24) | 32 | RW | 0000_0000h |
| A_0A64h | DECO2 Descriptor Buffer Word 25 (D2DESB25) | 32 | RW | 0000_0000h |
| A_0A68h | DECO2 Descriptor Buffer Word 26 (D2DESB26) | 32 | RW | 0000_0000h |
| A_0A6Ch | DECO2 Descriptor Buffer Word 27 (D2DESB27) | 32 | RW | 0000_0000h |
| A_0A70h | DECO2 Descriptor Buffer Word 28 (D2DESB28) | 32 | RW | 0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| A_0A74h | DECO2 Descriptor Buffer Word 29 (D2DESB29) | 32 | RW | 0000_0000h |
| A_0A78h | DECO2 Descriptor Buffer Word 30 (D2DESB30) | 32 | RW | 0000_0000h |
| A_0A7Ch | DECO2 Descriptor Buffer Word 31 (D2DESB31) | 32 | RW | 0000_0000h |
| A_0A80h | DECO2 Descriptor Buffer Word 32 (D2DESB32) | 32 | RW | 0000_0000h |
| A_0A84h | DECO2 Descriptor Buffer Word 33 (D2DESB33) | 32 | RW | 0000_0000h |
| A_0A88h | DECO2 Descriptor Buffer Word 34 (D2DESB34) | 32 | RW | 0000_0000h |
| A_0A8Ch | DECO2 Descriptor Buffer Word 35 (D2DESB35) | 32 | RW | 0000_0000h |
| A_0A90h | DECO2 Descriptor Buffer Word 36 (D2DESB36) | 32 | RW | 0000_0000h |
| A_0A94h | DECO2 Descriptor Buffer Word 37 (D2DESB37) | 32 | RW | 0000_0000h |
| A_0A98h | DECO2 Descriptor Buffer Word 38 (D2DESB38) | 32 | RW | 0000_0000h |
| A_0A9Ch | DECO2 Descriptor Buffer Word 39 (D2DESB39) | 32 | RW | 0000_0000h |
| A_0AA0h | DECO2 Descriptor Buffer Word 40 (D2DESB40) | 32 | RW | 0000_0000h |
| A_0AA4h | DECO2 Descriptor Buffer Word 41 (D2DESB41) | 32 | RW | 0000_0000h |
| A_0AA8h | DECO2 Descriptor Buffer Word 42 (D2DESB42) | 32 | RW | 0000_0000h |
| A_0AACh | DECO2 Descriptor Buffer Word 43 (D2DESB43) | 32 | RW | 0000_0000h |
| A_0AB0h | DECO2 Descriptor Buffer Word 44 (D2DESB44) | 32 | RW | 0000_0000h |
| A_0AB4h | DECO2 Descriptor Buffer Word 45 (D2DESB45) | 32 | RW | 0000_0000h |
| A_0AB8h | DECO2 Descriptor Buffer Word 46 (D2DESB46) | 32 | RW | 0000_0000h |
| A_0ABCh | DECO2 Descriptor Buffer Word 47 (D2DESB47) | 32 | RW | 0000_0000h |
| A_0AC0h | DECO2 Descriptor Buffer Word 48 (D2DESB48) | 32 | RW | 0000_0000h |
| A_0AC4h | DECO2 Descriptor Buffer Word 49 (D2DESB49) | 32 | RW | 0000_0000h |
| A_0AC8h | DECO2 Descriptor Buffer Word 50 (D2DESB50) | 32 | RW | 0000_0000h |
| A_0ACCh | DECO2 Descriptor Buffer Word 51 (D2DESB51) | 32 | RW | 0000_0000h |
| A_0AD0h | DECO2 Descriptor Buffer Word 52 (D2DESB52) | 32 | RW | 0000_0000h |
| A_0AD4h | DECO2 Descriptor Buffer Word 53 (D2DESB53) | 32 | RW | 0000_0000h |
| A_0AD8h | DECO2 Descriptor Buffer Word 54 (D2DESB54) | 32 | RW | 0000_0000h |
| A_0ADCh | DECO2 Descriptor Buffer Word 55 (D2DESB55) | 32 | RW | 0000_0000h |
| A_0AE0h | DECO2 Descriptor Buffer Word 56 (D2DESB56) | 32 | RW | 0000_0000h |
| A_0AE4h | DECO2 Descriptor Buffer Word 57 (D2DESB57) | 32 | RW | 0000_0000h |
| A_0AE8h | DECO2 Descriptor Buffer Word 58 (D2DESB58) | 32 | RW | 0000_0000h |
| A_0AECh | DECO2 Descriptor Buffer Word 59 (D2DESB59) | 32 | RW | 0000_0000h |
| A_0AF0h | DECO2 Descriptor Buffer Word 60 (D2DESB60) | 32 | RW | 0000_0000h |
| A_0AF4h | DECO2 Descriptor Buffer Word 61 (D2DESB61) | 32 | RW | 0000_0000h |
| A_0AF8h | DECO2 Descriptor Buffer Word 62 (D2DESB62) | 32 | RW | 0000_0000h |
| A_0AFCh | DECO2 Descriptor Buffer Word 63 (D2DESB63) | 32 | RW | 0000_0000h |
| A_0E00h | DECO2 Debug Job (D2DJR) | 32 | RO | 0000_0000h |
| A_0E04h | DECO2 Debug DECO (D2DDR) | 32 | RO | 0000_0000h |
| A_0E08h | DECO2 Debug Job Pointer (D2DJP) | 64 | RO | 0000_0000_0000_0000h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| A_0E10h | DECO2 Debug Shared Pointer (D2SDP) | 64 | RO | 0000_0000_0000_0000h |
| A_0E18h | DECO2 Debug_ICID, most-significant half (D2DIR_MS) | 32 | RO | 0000_0000h |
| A_0E20h | Sequence Output Length Register (SOL2) | 32 | RW | 0000_0000h |
| A_0E24h | Variable Sequence Output Length Register (VSOL2) | 32 | RW | 0000_0000h |
| A_0E28h | Sequence Input Length Register (SIL2) | 32 | RW | 0000_0000h |
| A_0E2Ch | Variable Sequence Input Length Register (VSIL2) | 32 | RW | 0000_0000h |
| A_0E30h | Protocol Override Register (D2POVRD) | 32 | RW | 0000_0000h |
| A_0E34h | Variable Sequence Output Length Register; Upper 32 bits (UVSOL2) | 32 | RW | 0000_0000h |
| A_0E38h | Variable Sequence Input Length Register; Upper 32 bits (UVSIL2) | 32 | RW | 0000_0000h |
| A_0F00h (alias) | Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ) | 64 | RW | 0000_0000_0000_0000h |
| A_0F08h (alias) | Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| A_0F10h (alias) | Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ) | 64 | RW | 0000_0000_0000_0000h |
| A_0F18h (alias) | Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT) | 64 | RW | 0000_0000_0000_0000h |
| A_0F20h (alias) | Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT) | 64 | RW | 0000_0000_0000_0000h |
| A_0F28h (alias) | Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT) | 64 | RW | 0000_0000_0000_0000h |
| A_0F30h (alias) | Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED) | 64 | RW | 0000_0000_0000_0000h |
| A_0FA0h (alias) | CHA Revision Number Register, most-significant half (CRNR_MS) | 32 | RO | 1000_1026h |
| A_0FA4h (alias) | CHA Revision Number Register, least-significant half (CRNR_LS) | 32 | RO | 4413_0017h |
| A_0FA8h (alias) | Compile Time Parameters Register, most-significant half (CTPR_MS) | 32 | RO | 0ABF_0211h |
| A_0FACh (alias) | Compile Time Parameters Register, least-significant half (CTPR_LS) | 32 | RO | 0000_7FFFh |
| A_0FC0h (alias) | Fault Address Register (FAR) | 64 | RO | 0000_0000_0000_0000h |
| A_0FC8h (alias) | Fault Address ICID Register (FAICID) | 32 | RO | 0000_0000h |
| A_0FCCh (alias) | Fault Address Detail Register (FADR) | 32 | RO | 0000_0000h |
| A_0FD4h (alias) | SEC Status Register (SSTA) | 32 | RO | 0000_0402h |
| A_0FE0h (alias) | RTIC Version ID Register (RVID) | 32 | RO | 0F0A_0003h |
| A_0FE4h (alias) | CHA Cluster Block Version ID Register (CCBVID) | 32 | RO | 0800_0005h |

*Table continues on the next page...*

| Offset | Register | Width<br>(In bits) | Access | Reset value |
|---|---|---|---|---|
| A_0FE8h (alias) | CHA Version ID Register, most-significant half (CHAVID_MS) | 32 | RO | 3400_0001h |
| A_0FECh (alias) | CHA Version ID Register, least-significant half (CHAVID_LS) | 32 | RO | 1014_3004h |
| A_0FF0h (alias) | CHA Number Register, most-significant half (CHANUM_MS) | 32 | RO | 4300_1113h |
| A_0FF4h (alias) | CHA Number Register, least-significant half (CHANUM_LS) | 32 | RO | 1111_3033h |
| A_0FF8h (alias) | SEC Version ID Register, most-significant half (SECVID_MS) | 32 | RO | 0A11_0301h |
| A_0FFCh (alias) | SEC Version ID Register, least-significant half (SECVID_LS) | 32 | RO | 0000_0000h |

## 13.2 Master Configuration Register (MCFGR)

### 13.2.1 Offset

| Register | Offset |
|---|---|
| MCFGR | 4h |

### 13.2.2 Function

The Master Configuration Register is used to set some bus master configurations. This register is typically written at boot time, and in some debug scenarios.

*Register descriptions in this document are based on none.*

## 13.2.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SWRST | WDE | WDF | DMA_RST | WRHD | Reserved | | | | | DJPC | DBPC | DWT | Reserved | NSP | PS |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ARCACHE | | | | AWCACHE | | | | AXIPIPE | | | | Reserved | LARGE_BURST | Reserved | NORMAL_BURST |
| Reset | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

## 13.2.4 Fields

| Field | Function |
|---|---|
| 31<br>SWRST | Software Reset. Writing a 1 to this bit will cause most registers and state machines in SEC to reset. The following SEC registers are not reset: MCFGR, SCFGR, JR0ICID_MS, JR0ICID_LS, JR1ICID_MS, JR1ICID_LS, JR2ICID_MS, JR2ICID_LS, JR3ICID_MS, JR3ICID_LS, QIICID, JRSTARTR, RTICAICID_MS, RTICAICID_LS, RTICBICID_MS, RTICBICID_LS, RTICCICID_MS, RTICCICID_LS, RTICDICID_MS, RTICDICID_LS, DECO0ICID_MS, DECO0ICID_LS, DECO1ICID_MS, DECO1ICID_LS, DECO2ICID_MS, DECO2ICID_LS, DECO3ICID_MS, DECO3ICID_LS, DAR, PBSL, JDKEKR_0 - JDKEKR_7, TDKEKR_0 - TDKEKR_7, TDSKR_0 - TDSKR_7, SKNR, RTMCTL, RTSCMISC, RTPKRRNG, RTPKRMAX, RTPKRSQ, RTSDCTL, RTTOTSAM, RTSBLIM, RTFRQMIN, RTFRQCNT, RTFRQMAX, RTSCML, RTSCMC, RTSCR1L, RTSCR1C, RTSCR2C, RTSCR2L, RTSCR3L, RTSCR3C, RTSCR4L, RTSCR4C, RTSCR5L, RTSCR5C, RTSCR6PC, RTSCR6PL, RTSTATUS, RTENT0 - RTENT11, RTPKRCNT10, RTPKRCNT32, RTPKRCNT54, RTPKRCNT76, RTPKRCNT98, RTPKRCNTBA, RTPKRCNTDC, RTPKRCNTFE, RDSTA, RDINT0, RDINT1, RDHCNTL, RDHDIG, RDHBUF, but the remaining registers in SEC register page 0 **are** reset by SWRST. The Job Ring registers in SEC register pages 1 .. 3 **are** reset by SWRST. The RTIC registers in SEC register page 6 **are not** reset by SWRST. (If an RTIC descriptor is in execution or is waiting for execution when SWRST is requested, RTIC will abandon the current sweep through all the hash blocks and restart hashing at the first hash block.) The Queue Manager Interface registers in SEC register page 7 **are** reset by SWRST. The DECO and CCB registers in SEC register pages 8 -3 **are** reset by SWRST.<br>Note that SWRST will remain 1 (and the registers will be held in reset) until any outstanding SEC DMA transactions complete. Writing a 1 to SWRST will not cause a reset of the SEC DMA unless SWRST is already 1 and a 1 is also written to DMARST. Note that writing to MCFGR will overwrite the values in LARGE_BURST, AXIPIPE, AWCACHE and ARCACHE, so to avoid disrupting outstanding DMA transactions when initiating a SWRST, these fields should be written with their current values. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 30<br>WDE | DECO Watchdog Enable. Enables the DECO Watchdog Timer to run. The Timer is used to detect and flush a job that has caused a DECO to hang. If the DECO Watchdog Timer expires, the hung job is usually flushed from the DECO with an error status indication. In those cases in which a hung job is not flushed automatically, software can reset the DECO via the DECO Reset Register. Note that the watchdog expiration period is extended for certain DECO operations and for RNG reseeding, because these can take longer than the normal watchdog expiration period. |
| 29<br>WDF | Watchdog Fast. Causes the DECO Watchdog Timer to expire prematurely for testing purposes. |
| 28<br>DMA_RST | DMA Reset. If SWRST is already 1, writing a 1 to DMARST and SWRST on the same cycle will cause the DMA to be reset. The DMA will not be reset if SWRST is not already a 1 (i.e. a 1 was previously written to SWRST but DMA transactions have not completed). Following a DMA reset, system software should delay long enough for outstanding AXI transaction responses to finish. These orphaned responses will be ignored. |
| 27<br>WRHD | Write Handoff Disable. If WRHD=0, when DECO has initiated the last write transaction of the current job DECO will go idle without waiting for the bus slave's response to that write transaction. This allows DECO to start another job while awaiting the slave's response. If an error response is eventually received SEC will update the transaction status appropriately. If WRHD=1 DECO will wait for the bus slave's response to the last write transaction before DECO goes to the idle state. Setting WRHD=1 is intended for product testing, so WRHD should normally be left at its PO reset value. |
| 26-22<br>— | Reserved |
| 21<br>DJPC | Disable Job Performance Counters. When DJPC=1 counting is disabled in the following Performance Counter registers:<br>PC_REQ_DEQ<br>PC_OB_ENC_REQ<br>PC_IB_DEC_REQ<br>Note that these registers can still be read or written even if DJPC=1. |
| 20<br>DBPC | Disable Byte Performance Counters. When DJPC=1 counting is disabled in the following Performance Counter registers:<br>PC_OB_ENCRYPT<br>PC_OB_PROTECT<br>PC_IB_DECRYPT<br>PC_IB_VALIDATED<br>Note that these registers can still be read or written even if DBPC=1. |
| 19<br>DWT | Double Word Transpose. Setting this bit affects whether the two words within a Dword are transposed when a double-word register is accessed, or when DMA performs a Dword memory transaction or when MOVEDW commands or certain MATH commands are executed.<br>When a double-word SEC register is read or written by software, the two words are transposed when<br>! ( SSTAR[PLEND] XOR MCFGR[ DWT] ). That is,<br>• for a Little-Endian platform (PLEND=0), the two words will be transposed if DWT=1.<br>• for a Big-Endian platform (PLEND=1), the two words will be transposed if DWT=0.<br>When SEC DMA performs a DWord memory access, the two words are transposed when<br>! ( SSTAR[PLEND] XOR MCFGR[ DWT] XOR PEO XOR DWSO), where PEO and DWSO are from the appropriate Job Ring Configuration Register (JRCFGR_JR) or from the Queue Interface Control Registers (QICTL). |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | That is,<br>• for a Little-Endian platform (PLEND=0), the two words will be transposed if DWT=0 and PEO = DWSO.<br>• for a Big-Endian platform (PLEND=1), the two words will be transposed if DWT=1 and PEO = DWSO.<br>For this chip PLEND=1 and DWT=0, so the most-significant half of 64-bit address registers will appear at the lower address (the address given in the register description), unless the endianness has been changed by setting the PEO or DWSO bits.<br>For an explanation of how DWT affects the MOVEDW command, see the MOVE, MOVEB, MOVEDW and MOVE_LEN commands section.<br>For an explanation of how DWT affects the MATH command, see the MATH and MATHI command section. |
| 18<br>— | Reserved |
| 17<br>NSP | No Snoop. This bit controls whether SEC marks all subsequent memory access transactions by default as sharable or not sharable (coherent or non-coherent).<br>**NOTE:** The NSP configuration defines the SEC-generated default transaction attributes. The effective transaction attributes utilized in the SoC interconnect are subject to SMMU attribute translation.<br>0b - All SEC transactions are marked to be snooped by the coherency control logic of the SoC.<br>1b - All SEC transactions are marked to be not snooped by the coherency control logic of the SoC. |
| 16<br>PS | Pointer Size. This bit determines the size of address pointers. (see Address pointers).<br>0b - Pointers fit in one 32-bit word (pointers are 32-bit addresses).<br>1b - Pointers require two 32-bit words (pointers are 40-bit addresses). |
| 15-12<br>ARCACHE | AXI Read Transaction Attributes. This field provides default values for the generation of the ARCACHE[3:0] interface signals for read transactions. For a general description of ARCACHE signals refer to the AXI3/4 protocol specification.<br>The following functionality, limitations, and extensions exist in this version of SEC:<br>• ARCACHE[0] (Bufferable):<br>This bit is intended to indicate whether read data may be fetched from an intermediate point in the interconnect or must be fetched from the transaction target (for details see AXI4 specification).<br>• ARCACHE[1] (Cacheable/Modifiable):<br>A setting of 1 indicates the transaction attributes may be modified, e.g., to improve performance.<br>**NOTE:** ARCACHE[1] must be set if the transaction is marked as sharable (coherent).<br>**NOTE:** SEC may drive the ARCACHE[1] signal to 1 if read-safe is enabled, independent of the value of the ARCACHE[1] bit configuration.<br>• ARCACHE[3:2] (Cache check and allocate controls):<br>This setting and associated signal assertions are irrelevant because this SoC does not have a downstream cache.<br>**NOTE:** The ARCACHE[3:0] configuration defines the SEC-generated default transaction attributes. The effective transaction attributes utilized in the SoC interconnect are subject to SMMU attribute translation. |
| 11-8<br>AWCACHE | AXI Write Transaction Attributes. This field provides default values for the generation of the AWCACHE[3:0] interface signals for write transactions. For a general description of AWCACHE signals refer to the AXI3/4 protocol specifications.<br>The following functionality, limitations, and extensions exist for this version of SEC: |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | • AWCACHE[0] (Bufferable):<br>A setting of 1 indicates the transaction response may be generated from an intermediate point and the transaction may be delayed reaching its final destination (this setting is intended to reduce transaction latency and improve performance).<br>**NOTE:** This SoC ignores SEC-generated AWCACHE[0] signals and marks all write transactions as bufferable while guaranteeing that a response is generated only after write data using the same AWID has been made visible to other masters and in the order the transactions were issued.<br>• AWCACHE[1] (Cacheable/Modifiable):<br>A setting of 1 indicates the transaction attributes may be modified, e.g., to improve performance.<br>**NOTE:** AWCACHE[1] must be set if the transaction is marked as sharable (coherent).<br>• AWCACHE[3:2] (Cache check and allocate controls):<br>This setting and associated signal assertions are irrelevant because this SoC does not have a downstream cache.<br>**NOTE:** The AWCACHE[3:0] configuration defines the SEC-generated default transaction attributes. The effective transaction attributes utilized in the SoC interconnect are subject to SMMU attribute translation. |
| 7-4<br>AXIPIPE | AXI Pipeline Depth.<br>The AXIPIPE field is a debug field used to adjust the maximum number of outstanding DMA transactions that SEC is able to queue. Optimal performance will be achieved by retaining the default value of this field. |
| 3<br>— | Reserved |
| 2<br>LARGE_BURST | Enable Large Bursts. When LARGE_BURST=1, Job Descriptor reads, Shared Descriptor reads and reads of data for the Data FIFO can use transactions as large as the maximum AXI interface transaction size, which equals 16 times the width of the AXI data buses (i.e. 256 bytes for 128-bit data buses). When LARGE_BURST=0, all master bus transactions use the normal burst size. Changes to LARGE_BURST should be made only when SEC is not processing jobs. |
| 1<br>— | Reserved |
| 0<br>NORMAL_BURST | Normal Burst Size<br>This field indicates the normal burst (e.g. transaction) size limits for SEC's read and write accesses to memory. (Note that this field was writable in earlier versions of SEC, but this capability is now obsolete.)<br>0b - 32 byte burst size<br>1b - 64 byte burst size |

## 13.3 Security Configuration Register (SCFGR)

### 13.3.1 Offset

| Register | Offset |
|---|---|
| SCFGR | Ch |

### 13.3.2 Function

The Security Configuration Register is used to set security-related mode bits. These bits are used to switch from special boot-time operating modes to normal operating modes. At POR, all bits in SCFGR reset to 0.

When RNGSH0 is 0, RNG DRNG State Handle 0 can be instantiated in deterministic mode. This allows the secure boot software to run deterministic tests on the RNG and its State Handle 0 logic. Once the tests have been completed, the secure boot software can write a 1 to RNGSH0 to prevent State Handle 0 from being instantiated in deterministic mode. This ensures that random data, rather than deterministic data, is used for the Zeroizable Master Key Register in the low-power section of Security Monitor, the Differential Power Analysis Resistance Mask in the AESA, the Job Descriptor Key Encryption Register, the Trusted Decriptor Key Encryption Register, the Trusted Descriptor Signing Key Register and the random padding used by built-in protocols.

At POR the AESA will seed its Differential Power Analysis Resistance Mask using a default deterministic value. Writing a 1 to the RANDDPAR bit will cause the AESA to reseed its Differential Power Analysis Resistance Mask using data from RNG DRNG State Handle 0. Typically the secure boot software will write a 1 to RANDDPAR after RNG DRNG State Handle 0 has been instantiated in a non-deterministic mode. Once a 1 has been written to RANDDPAR, this bit will remain 1 until the next POR. Software can read this bit to determine whether the Differential Power Analysis Resistance Mask was seeded from the RNG.

The PRIBLOB field is used to select a private blob type during Trusted Mode. When a General Memory Blob or Secure Memory Blob is encapsulated or decapsulated during Trusted Mode, the PRIBLOB bits are used to modify the derivation of the Blob Key Encryption Key (see Blob encapsulation). This is used to enforce cryptographic separation of private blob types during the boot process (and thereafter). These bits reset to 0 at POR, but when a PRIBLOB bit is written to a 1, it remains a 1 until the next POR.

The PRIBLOB=00 setting allows secure boot software to have its own private blobs that cannot be decapsulated or encapsulated by other software, even software that later runs in Trusted Mode. This feature can be used to safeguard boot reference metrics (e.g. hash values over software). In this use case, the reference metrics might be initially verified via a public key signature and then encapsulated in a private secure boot blob. On subsequent boot cycles the protected reference metrics would be obtained by decapsulating the private secure boot blob, obviating the time-consuming public key signature verification process.

The PRIBLOB=01 and PRIBLOB=10 settings allow trusted provisioning software (e.g. software that handles DRM keys) to have private blobs that cannot be decapsulated or encapsulated by software that runs later in the boot process, even if that software runs in Trusted Mode.

As illustrated in Figure 13-1, typically the secure boot software would enter Trusted Mode, then encapsulate or decapsulate all of its private blobs, and then would write either a 01, 10 or 11 to PRIBLOB. For the remainder of the current power-on session, private secure boot blobs could no longer be encapsulated or decapsulated. The secure boot software would then either run provisioning software with the 10 or 01 setting, or would skip the provisioning software and run the normal boot software with the 11 setting. If the provisioning software runs, it would encapsulate or decapsulate its own private blobs and then write 11 to PRIBLOB. At this point PRIBLOB=11 for the remainder of the current power on session, and no software can encapsulate or decapsulate private secure boot software blobs or either type of private provisioning blobs.

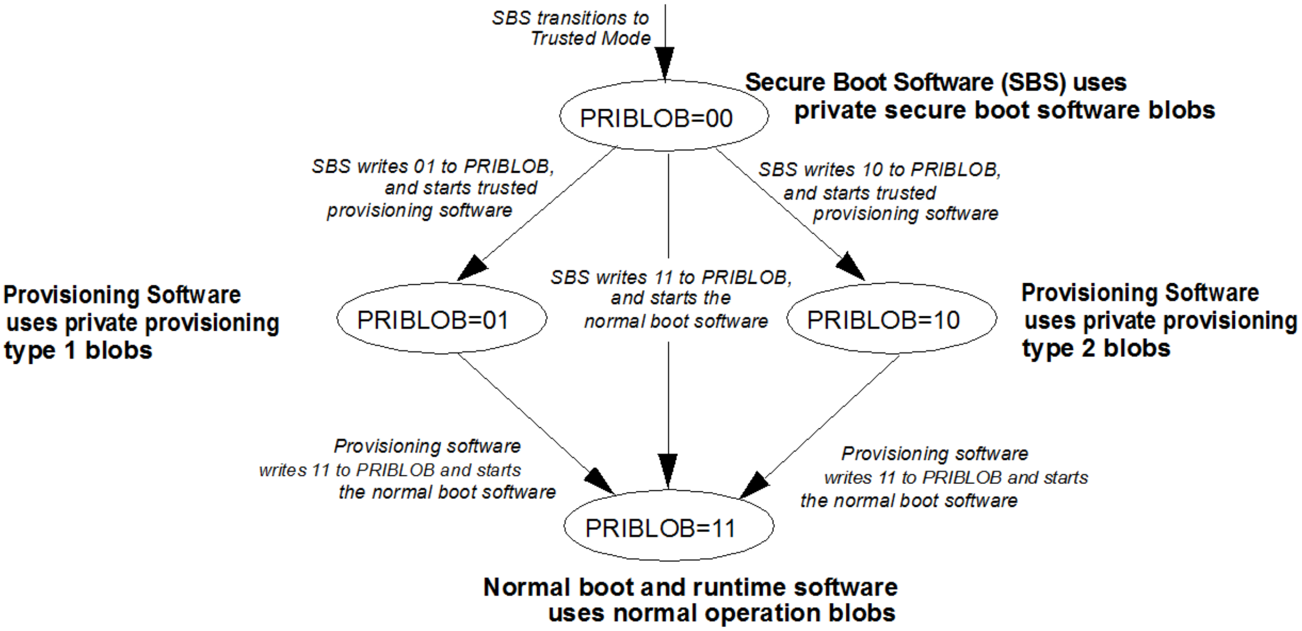


Figure 13-1. Process for Managing Private Blobs

## 13.3.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MPCURVE | | | | | MPMRL | Reserved | | | | | | | | | |
| W | | | | | MPPKRC | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | VIRT_EN | Reserved | | | LCK_TRNG | RDB | RNGSH0 | RANDDPAR | Reserved | | | | | | PRIBLOB | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.3.4 Fields

| Field | Function |
|---|---|
| 31-28<br>MPCURVE | Manufacturing Protection Curve. This shows the elliptic curve that was selected when the MPPrivK Generation protocol was run. |
| 27<br>MPPKRC | Manufacturing Protection Private Key Register Clear. Writing a 1 to this bit clears the Manufacturing Protection Private Key Register. |
| 26<br>MPMRL | Manufacturing Protection Message Register Lock. Writing a 1 to this bit locks the Manufacturing Protection Message Register for writing. The register remains locked until the next POR. |
| 25-16<br>— | Reserved |
| 15<br>VIRT_EN | Virtualization enable. Virtualization is disabled by default. Writing a 1 to this bit enables job ring virtualization. When job ring virtualization in enabled, the Start_JRa bits in the JRSTART register must be used to switch between writing the job ring registers in register page 0 and writing the job ring registers in register pages 1....4.<br>Note that the LAMTD bit in the JRaICID register cannot be written unless VIRT_EN is 1. |
| 14-12<br>— | Reserved |
| 11<br>LCK_TRNG | Lock TRNG Program Mode. Writing a 1 to this bit locks the TRNG. That is, when this bit is set TRNG can't go into program mode. If it is in program mode when this bit is set, the TRNG will immediately leave program mode. Once this bit has been written to a 1, it cannot be changed to a 0 until the next power on reset. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 10<br>RDB | Enable random data buffer. Set this bit to 1 to enable a buffer for data obtained from the RNG. This is a 196-byte buffer that is intended to improve performance when built-in protocols request random data to be used for initialization vectors or padding. When the buffer contains 64 or fewer bytes of random data, another 128 bytes is obtained from RNG State Handle 0. This bit would typically be set to 1 after any boot-time RNG testing has been completed and State Handle 0 has been instantiated in non-deterministic mode. Once this bit has been written to a 1, it cannot be changed to a 0 until the next power on reset. |
| 9<br>RNGSH0 | Random Number Generator State Handle 0.<br>0b - When RNGSH0 is 0, RNG DRNG State Handle 0 can be instantiated in any mode. RNGSH0 is set to 0 only for testing.<br>1b - When RNGSH0 is 1, RNG DRNG State Handle 0 cannot be instantiated in deterministic (test) mode. RNGSHO should be set to 1 before the RNG is instantiated. If it is currently instantiated in a deterministic mode, it will be un-instantiated. Once this bit has been written to a 1, it cannot be changed to a 0 until the next power on reset. |
| 8<br>RANDDPAR | Random Differential Power Analysis Resistance (DPAR) Mask. After RNGSH0 has been set and the RNG has been instantiated, RANDPAR should be set to 1.<br>0b - The AESA DPAR Mask was seeded using the default deterministic seed. This mode is used for testing.<br>1b - When RANDDPAR is written with a 1, the AESA DPAR Mask is reseeded from RNG DRNG State Handle 0. This is the normal runtime mode. Once RANDDPAR has been written with a 1, it cannot be changed to a 0 until the next power on reset. |
| 7-2<br>— | Reserved |
| 1-0<br>PRIBLOB | Private Blob. This field selects one of four different types of private blobs during Trusted Mode. All blobs encapsulated or decapsulated during Trusted Mode will be of the type specified in this field, until a 1 is written to any or the bits, or until the next POR. The bits of this field are "sticky", i.e. once a bit has been written to a 1, it cannot be changed to a 0 until the next power on reset.<br>00b - private secure boot software blobs<br>01b - private provisioning type 1 blobs<br>10b - private provisioning type 2 blobs<br>11b - normal operation blobs |

## 13.4 Job Ring a ICID Register - most significant half (JR0ICID_MS - JR3ICID_MS)

### 13.4.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| JRaICID_MS | 10h + (a × 8h) | used with job ring a |

## 13.4.2 Function

There is one JRaICID register per job ring and it is used to indicate the Security Domain that currently owns the job ring and to specify the ICID values that the SEC DMA asserts when reading or writing memory on behalf of descriptors fetched from a particular job ring.TrustZone SecureWorld can reserve a Job Ring for itself by setting the TZ bit to 1. Note that TZ can be set to 1 only if the register is written using a SecureWorld bus transaction. If the Job Ring is reserved by SecureWorld the Job Ring registers associated with this ring can be written only via SecureWorld bus transactions. NonSecureWorld writes to Job Ring registers reserved by SecureWorld will be ignored.

This register also contains a bit that grants permission for Trusted Descriptors to be created in this job ring. If the job ring is owned by TrustZone SecureWorld (TZ=1), any TrustedDescriptor created in this job ring is marked as a TrustZone Trusted Descriptor (see TDES field in HEADER command. Once a TrustZone Trusted Descriptor has been created, it can be executed in any job ring. If the job ring is not owned by TrustZone SecureWorld, any Trusted Descriptor created in this job ring is marked as a nonTrustZone Trusted Descriptor.

If virtualization mode is disabled in the Security Configuration register, the SDID field is not writable and will remain at the default all-0 value.

## 13.4.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | LICID | Reserved | | | | | | | | | | | | | LAMTD | AMTD |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | TZ | Reserved | | | SDID | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.4.4 Fields

| Field | Function |
|---|---|
| 31<br>LICID | Lock ICIDs. Once LICID has been set, no further changes can be made to the LICID, SEQ_ICID or NONSEQ_ICID fields until reset. The SDID field is not locked, and the AMTD is locked only if LAMTD=1. |
| 30-18<br>— | Reserved |
| 17<br>LAMTD | Lock AMTD. Once LAMTD has been set, no further changes can be made to the AMTD field. Note that the LAMTD bit cannot be written unless virtualization mode is enabled (SCFGR[VIRT_EN]=1). |
| 16<br>AMTD | Allow Make Trusted Descriptor.<br>If AMTD is set, the job ring associated with this register is permitted to issue jobs that create Trusted Descriptors. When DECO encounters a descriptor header with the MTD bit set, the options specified in the SIGNATURE command at the end of the descriptor determine whether DECO will execute the commands in the descriptor, or append a signature to the descriptor, or both. If the job ring is owned by TrustZone SecureWorld, the descriptor will be treated as a TrustZone Trusted Descriptor, otherwise the descriptor will be treated as a non-SecureWorld Trusted Descriptor.<br>If AMTD is not set, then executing a descriptor with the MTD bit set in the descriptor's header will result in an error, and no signature will be generated. |
| 15<br>TZ | TrustZone SecureWorld. This bit can be written only by TrustZone SecureWorld (i.e. a bus transaction with ns=0). If TZ=1, this job ring is owned by TrustZone SecureWorld and the SDID field is forced to all 0s. If TZ=0 this job ring is owned by non-SecureWorld and the SDID field is writable. |
| 14-12<br>— | Reserved |
| 11-0<br>SDID | Security Domain Identifier. If TZ=0, SDID indicates the Security Domain Identifier that owns this job ring. The 12-bit SDID value is used to tag Black Keys, Blobs and Trusted Descriptors so they can be used only by this Security Domain. If TZ=1, the SDID value is forced to all 0s and Black Keys, Blobs and Trusted Descriptors are tagged with the special TrustZone SecureWorld tag. If virtualization mode is disabled, the SDID is forced to all 0s. |

## 13.5 Job Ring a ICID Register - least significant half (JR0ICID_LS - JR3ICID_LS)

### 13.5.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| JRaICID_LS | 14h + (a × 8h) | used with job ring a |

## 13.5.2 Function

There is one JRaICID register per job ring and it is used to indicate the Security Domain that currently owns the job ring and to specify the ICID values that the SEC DMA asserts when reading or writing memory on behalf of descriptors fetched from a particular job ring. TrustZone SecureWorld can reserve a Job Ring for itself by setting the TZ bit to 1. Note that TZ can be set to 1 only if the register is written using a SecureWorld bus transaction. If the Job Ring is reserved by SecureWorld the Job Ring registers associated with this ring can be written only via SecureWorld bus transactions. NonSecureWorld writes to Job Ring registers reserved by SecureWorld will be ignored.

This register also contains a bit that grants permission for Trusted Descriptors to be created in this job ring. If the job ring is owned by TrustZone SecureWorld (TZ=1), any Trusted Descriptor created in this job ring is marked as a TrustZone Trusted Descriptor (see TDES field in HEADER command. Once a TrustZone Trusted Descriptor has been created, it can be executed in any job ring. If the job ring is not owned by TrustZone SecureWorld, any Trusted Descriptor created in this job ring is marked as a nonTrustZone Trusted Descriptor. The SEQ_ICID and NONSEQ_ICID fields are typically written at boot time and then locked.

If virtualization mode is disabled in the Security Configuration register, the SDID and LAMTD fields are not writable, and the SDID will remain at the default all 0 value.

## 13.5.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | NONSEQ_ICID | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | SEQ_ICID | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.5.4 Fields

| Field | Function |
|---|---|
| 31-28 | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 27-16<br>NONSEQ_ICID | Job Ring Non-SEQ ICID. This field defines the ICID value asserted for DMA transactions associated with external memory accesses for non-sequence commands, such as KEY, LOAD, and STORE. By default the Job Descriptor is read using this ICID, although that behavior can be changed by setting the JDIS bit in the corresponding job ring Configuration Register. Once the LICID bit in the JRaICID_MS register has been set to 1, until reset no further changes can be made to the NONSEQ_ICID field.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |
| 15-12<br>— | Reserved |
| 11-0<br>SEQ_ICID | SEQ ICID. This field defines the ICID value asserted for DMA transactions associated with external memory accesses for sequence commands, such as SEQ_KEY, SEQ_LOAD, and SEQ_STORE. Setting the JDIS bit in the corresponding job ring Configuration Register will cause the Job Queue to use this ICID for Job Descriptor reads. Once the LICID bit in the JRaICID_MS register has been set to 1, until reset no further changes can be made to the SEQ ICID field.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

## 13.6 Queue Manager Interface SDID Register (QISDID)

### 13.6.1 Offset

| Register | Offset |
|---|---|
| QISDID | 50h |

### 13.6.2 Function

The QISDID Register is used to indicate the Security Domain that is associated with the Queue Manager Interface. This SDID value is used to tag Black Keys and blobs that are referenced by descriptors executed via the Queue Manager Interface.

If virtualization mode is disabled in the Security Configuration register, the SDID field is not writable and will remain at the default all 0 value.

## 13.6.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | TZ | Reserved | | | SDID | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.6.4 Fields

| Field | Function |
|---|---|
| 31-16<br>— | Reserved |
| 15<br>TZ | TrustZone SecureWorld. This bit can be written only by TrustZone SecureWorld (i.e. a bus transaction with ns=0). If TZ=1, the Queue Manager Interface is associated with TrustZone SecureWorld and the SDID field is forced to all 0s. If TZ=0 the Queue Manager Interface is associated with non-SecureWorld and the SDID field is writable. |
| 14-12<br>— | Reserved |
| 11-0<br>SDID | Security Domain Identifier. If TZ=0, SDID indicates the Security Domain associated with the Queue Manager Interface. The SDID value is used to tag Black Keys, Blobs and Trusted Descriptors so they can be used only by this Security Domain. If TZ=1, the SDID value is forced to all 0s and Black Keys, Blobs and Trusted Descriptors are tagged with the special TrustZone SecureWorld tag. If virtualization mode is disabled, the SDID is forced to all 0s. |

# 13.7 Debug Control Register (DEBUGCTL)

## 13.7.1 Offset

| Register | Offset |
|---|---|
| DEBUGCTL | 58h |

## 13.7.2 Function

The DEBUGCTL Register is used to stop SEC from processing jobs so that a consistent read of the debug registers can be performed.

## 13.7.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | STOP_ACK | STOP |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.7.4 Fields

| Field | Function |
|---|---|
| 31-18<br>— | Reserved |
| 17<br>STOP_ACK | STOP_ACK will assert when the job queue controller and QI acknowledge that they are stopped. |
| 16<br>STOP | STOP is written to 1 to request that SEC stop processing jobs. This is intended to be a graceful halt. SEC will shut down in such a way that it will be able to resume processing where it left off once software has finished reading the debug registers. Note that the RTIC watchdog timer will continue to run during the halt. It is recommended that the DECO watchdog timer be turned off (see Master Config Register) prior to stopping SEC in order to prevent a possible watchdog error from a job in a stopped DECO. When STOP is asserted, the DECO Availability register can be used to monitor which DECOs are stopped or are available. |
| 15-0<br>— | Reserved. |

# 13.8 Job Ring Start Register (JRSTARTR)

## 13.8.1 Offset

| Register | Offset |
|---|---|
| JRSTARTR | 5Ch |

## 13.8.2 Function

The Job Ring Start register is used by the system software or TrustZone SecureWorld when configuring a job ring for a new user. Before the job ring is configured for a new user, the job ring must be in stop mode. Before the new user can set up the job ring and begin using it, the job ring must be in start mode.

The Job Ring Start register is not used and is not writable when virtualization mode is disabled in the Security Configuration register. All bits in this register will remain at the default 0 value when virtualization mode is disabled.

## 13.8.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | Start_JR3 | Start_JR2 | Start_JR1 | Start_JR0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.8.4 Fields

| Field | Function |
|---|---|
| 31-4 | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 3<br>Start_JR3 | Start Job Ring 3. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0).If Job Ring 3 is allocated to TrustZone SecureWorld (JR3ICID[TZ]=1), Start_JR3 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0.<br>0b - Stop Mode. The JR3ICID register for Job Ring 3 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 3 are NOT accessible. If Job Ring 3 is allocated to TrustZone SecureWorld (JR3ICID[TZ]=1), the JR3ICID register can be written only via a bus transaction that has ns=0.<br>1b - Start Mode. The JR3ICID register for Job Ring 3 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 3 ARE accessible. If Job Ring 3 is allocated to TrustZone SecureWorld (JR3ICID[TZ]=1), then the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 3 can be written only via a bus transaction that has ns=0. |
| 2<br>Start_JR2 | Start Job Ring 2. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0).If Job Ring 2 is allocated to TrustZone SecureWorld (JR2ICID[TZ]=1), Start_JR2 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0.<br>0b - Stop Mode. The JR2ICID register for Job Ring 2 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 2 are NOT accessible. If Job Ring 2 is allocated to TrustZone SecureWorld (JR2ICID[TZ]=1), the JR2ICID register can be written only via a bus transaction that has ns=0.<br>1b - Start Mode. The JR2ICID register for Job Ring 2 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 2 ARE accessible. If Job Ring 2 is allocated to TrustZone SecureWorld (JR2ICID[TZ]=1), then the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 2 can be written only via a bus transaction that has ns=0. |
| 1<br>Start_JR1 | Start Job Ring 1. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0).If Job Ring 1 is allocated to TrustZone SecureWorld (JR1ICID[TZ]=1), Start_JR1 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0.<br>0b - Stop Mode. The JR1ICID register for Job Ring 1 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 1 are NOT accessible. If Job Ring 1 is allocated to TrustZone SecureWorld (JR1ICID[TZ]=1), the JR1ICID register can be written only via a bus transaction that has ns=0.<br>1b - Start Mode. The JR1ICID register for Job Ring 1 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 1 ARE accessible. If Job Ring 1 is allocated to TrustZone SecureWorld (JR1ICID[TZ]=1), then the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 1 can be written only via a bus transaction that has ns=0. |
| 0<br>Start_JR0 | Start Job Ring 0. This bit is not writable if virtualization mode is disabled (SCFGR[VIRT_EN]=0).If Job Ring 0 is allocated to TrustZone SecureWorld (JR0ICID[TZ]=1), Start_JR0 can be changed only by writing to JRSTARTR via a bus transaction that has ns=0.<br>0b - Stop Mode. The JR0ICID register for Job Ring 0 can be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 0 are NOT accessible. If Job Ring 0 is allocated to TrustZone SecureWorld (JR0ICID[TZ]=1), the JR0ICID register can be written only via a bus transaction that has ns=0.<br>1b - Start Mode. The JR0ICID register for Job Ring 0 CANNOT be written but the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR for Job Ring 0 ARE accessible. If Job Ring 0 is allocated to TrustZone SecureWorld (JR0ICID[TZ]=1), then the IRBAR, IRSR, IRSAR, IRJAR, ORBAR, ORSR, ORJRR, ORSFR and JRSTAR registers for Job Ring 0 can be written only via a bus transaction that has ns=0. |

# 13.9 RTIC ICID Register for Block a - most significant half (RTICAICID_MS - RTICDICID_MS)

## 13.9.1 Offset

For a = A to D (0 to 3):

| Register | Offset | Description |
|---|---|---|
| RTICaICID_MS | 60h + (a × 8h) | used with Block a |

## 13.9.2 Function

There is one RTICaICID register per RTIC hash block. The RTIC ICID Register is used to specify the AXI bus ICID value that the SEC DMA asserts when reading a particular RTIC hash block from memory external to SEC. This register is typically written at boot time and then locked.

## 13.9.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | LCK | TZCTL | Reserved | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.9.4 Fields

| Field | Function |
|---|---|
| 31<br>LCK | RTIC ICID Lock. Once LCK has been set, no further changes can be made to RTICICID (including changes to LCK) until the next POR. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 30<br>TZCTL | TrustZone Control. The TZCTL field exists only in the RTICAICID register. This bit is reserved in RTICBICID, RTICCICID, and RTICDICID. The TZCTL bit can be written only if ips_nonsecure_access=0 (i.e. only via a TrustZone SecureWorld bus transaction). Software sets the TZCTL bit to 1 to indicate that TrustZone SecureWorld owns RTIC. If TZCTL=1 the RTIC registers (offsets 60h ... 7Ch and offsets 60000h .. 6FFFFh can be written only via a TrustZone SecureWorld bus transaction. If TZCTL=1 the RTICICID[TZ] bits can be written with either a 1 or a 0 (i.e. TrustZone SecureWorld software can decide whether SecureWorld or NonSecureWorld bus transactions are asserted for a particular RTIC memory block). If TZCTL=0 all RTICICID[TZ] bits are forced to 0 and all RTIC bus transactions are NonSecureWorld. |
| 29-0<br>— | Reserved |

## 13.10 RTIC ICID Register for Block a - least significant half (RTICAICID_LS - RTICDICID_LS)

### 13.10.1 Offset

For a = A to D (0 to 3):

| Register | Offset | Description |
|---|---|---|
| RTICaICID_LS | 64h + (a × 8h) | used with Block a |

### 13.10.2 Function

There is one RTICaICID register per RTIC hash block. The RTIC ICID Register is used to specify the AXI bus ICID value that the SEC DMA asserts when reading a particular RTIC hash block from memory external to SEC. This register is typically written at boot time and then locked.

### 13.10.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | R_ICID | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.10.4 Fields

| Field | Function |
|---|---|
| 31-12<br>— | Reserved |
| 11-0<br>R_ICID | RTIC ICID. This field defines the ICID value asserted when RTIC accesses the addresses in the hash block a regions of memory.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

## 13.11 DECO Request Source Register (DECORSR)

### 13.11.1 Offset

| Register | Offset |
|---|---|
| DECORSR | 94h |

### 13.11.2 Function

The DECO Request Source Register is used to indicate a particular job ring whose JRaICID register will be used to supply the ICID, TZ and SDID values when descriptor commands are executed under direct software control. The selected job ring will be used for all DECOs that are presently under direct software control regardless of whether those

DECOs were requested via a single write or via multiple writes to the DECO Request Register. The DECO Request Source Register is writable only when all bits in the DECO Request Register are 0 (i.e. no DECOs are requested or already under direct software control). If the job ring selected via the DECORSR is later changed to stop mode, all DECOs that are under direct software control are reset and returned to the pool of DECOs available for processing normal jobs.

Note that the DECO Request Source register is not used and is not writable when virtualization is disabled.

## 13.11.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | VALID | Reserved | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | JR | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.11.4 Fields

| Field | Function |
|---|---|
| 31<br>VALID | Valid. This bit will be set to 1 to indicate that the JR field contains a valid job ring number. If an invalid job ring number is written to JR, the VALID bit will be 0. The VALID bit will always remain 0 if virtualization mode is disabled. |
| 30-2<br>— | Reserved |
| 1-0<br>JR | Job Ring number. This job ring's JRaICID register will be used to supply the ICID, TZ and SDID values when decriptor commands are executed under direct software control. If the specified job ring is not implemented or not in start mode, the JR field will not be changed and the VALID bit will remain 0. If the DECORSR is written via a bus transaction that has ns=0, any implemented job ring can be selected. But if the bus transaction has ns=1, then only job rings with TZ=0 can be selected. If a TrustZone job ring number is written via an ns=1 bus write to DECORSR the JR field will not be changed and the VALID bit will remain 0. This field is not writable if virtualization mode is disabled. |

# 13.12 DECO Request Register (DECORR)

## 13.12.1 Offset

| Register | Offset |
|---|---|
| DECORR | 9Ch |

## 13.12.2 Function

This register is used when software wants to bypass the normal job queue controller mechanism and directly access one of DECO/CCB blocks. This interface would normally be used only for debugging and testing purposes since it is not as efficient as the Queue Manager Interface or Job Ring Interface. Note that multiple DECOs can be requested via a single write to DECORR, and additional DECOs can be requested by writing to DECORR multiple times. DECORR may be written only when all DECOs that have been requested have had that access granted (DENn=RQDn). The procedure for directly accessing a DECO/CCB block is described in detail in Register-based service interface.

Note that, unless virtualization mode is disabled in the Security Configuration register, a job ring that is in the start mode must be selected via the DECO Request Source Register prior to requesting a DECO via DECORR. If the DECO Request Source Register's VALID bit is not set, or if the selected job ring is not in start mode, the DECO Request Register cannot be written. The ICID, TZ and SDID value from the selected job ring's JRaICID register will be used when descriptor commands are executed under direct software control. If the job ring selected via the DECORSR is later changed to stop mode, all DECOs that are under direct software control are reset and returned to the pool of DECOs available for processing normal jobs.

If virtualization mode is disabled the DECO ICID registers supply the ICID values used when descriptor commands are executed under direct software control, and an all-zero SDID is used. The DECO Request Source register is not used when virtualization is disabled.

## 13.12.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | DEN2 | DEN1 | DEN0 |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | RQD2 | RQD1 | RQD0 |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.12.4 Fields

| Field | Function |
|---|---|
| 31-19<br>— | Reserved. Always 0. |
| 18<br>DEN2 | The job queue controller asserts this bit when permission is granted for the software to directly access DECO 2/CCB 2. |
| 17<br>DEN1 | The job queue controller asserts this bit when permission is granted for the software to directly access DECO 1/CCB 1. |
| 16<br>DEN0 | The job queue controller asserts this bit when permission is granted for the software to directly access DECO 0/CCB 0. |
| 15-3<br>— | Reserved. Always 0. |
| 2<br>RQD2 | This bit is set by software to request direct access to DECO 2/CCB 2. It cannot be cleared until the direct access operation is complete or SEC gets a software reset. |
| 1<br>RQD1 | This bit is set by software to request direct access to DECO 1/CCB 1. It cannot be cleared until the direct access operation is complete or SEC gets a software reset. |
| 0<br>RQD0 | This bit is set by software to request direct access to DECO 0/CCB 0. It cannot be cleared until the direct access operation is complete or SEC gets a software reset. |

## 13.13 DECOa ICID Register - most significant half (DECO0ICID_MS - DECO2ICID_MS)

### 13.13.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DECOaICID_MS | A0h + (a × 8h) | used with DECOa |

### 13.13.2 Function

This register is used when virtualization is disabled via the VIRT_EN bit in the Security Configuration Register. In that case this register specifies various bus signal values that SEC's DMA will assert when a DECO is under direct software control. When virtualization is enabled, this register is not used and instead the DECO Request Source Register is used to select one of the Job Rings whose JRaICID register will supply these bus signal values. The DECO ICID register is used to specify the ICID values that the SEC DMA asserts when reading or writing memory on behalf of a DECO that is under the direct control of software. This register is intended to be written by the same processor that writes to the DECORR.

### 13.13.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | LCK | Reserved | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.13.4 Fields

| Field | Function |
|---|---|
| 31<br>LCK | Lock. Once LCK is set, no further changes can be made to DECOaICID (including any changes to LCK). LCK is not writable when virtualization mode is enabled. |
| 30-0<br>— | Reserved |

## 13.14 DECOa ICID Register - least significant half (DECO0ICID_LS - DECO2ICID_LS)

### 13.14.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DECOaICID_LS | A4h + (a × 8h) | used with DECOa |

### 13.14.2 Function

This register is used when virtualization is disabled via the VIRT_EN bit in the Security Configuration Register. In that case this register specifies various bus signal values that SEC's DMA will assert when a DECO is under direct software control. When virtualization is enabled, this register is not used and instead the DECO Request Source Register is used to select one of the Job Rings whose JRaICID register will supply these bus signal values. The DECO ICID register is used to specify the ICID values that the SEC DMA asserts when reading or writing memory on behalf of a DECO that is under the direct control of software. This register is intended to be written by the same processor that writes to the DECORR.

## 13.14.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | DNSEQ_ICID | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | DSEQ_ICID | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.14.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved |
| 27-16<br>DNSEQ_ICID | DECO Non-SEQ ICID. This field defines the ICID value asserted for DMA transactions associated with external memory accesses for non-sequence commands, such as KEY, LOAD, and STORE, when this DECO is under the direct control of software.<br>This field is not writable when virtualization mode is enabled.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |
| 15-12<br>— | Reserved |
| 11-0<br>DSEQ_ICID | DECO SEQ ICID. This field defines the ICID value asserted for DMA transactions associated with external memory accesses for sequence commands, such as SEQ_KEY, SEQ_LOAD, and SEQ_STORE, when this DECO is under the direct control of software.<br>This field is not writable when virtualization mode is enabled.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

# 13.15 DECO Availability Register (DAR)

## 13.15.1 Offset

| Register | Offset |
|---|---|
| DAR | 120h |

## 13.15.2 Function

The DECO Availability Register can be used to determine whether DECOs are hung. If software writes a 1 to the DECO's NYA field, the corresponding DECO will clear that bit whenever the DECO is, or becomes, available. The bit can be polled to determine if the DECO is completing jobs. If STOP is asserted in the DEBUG Control register, the DECO Availability Register cannot be written. While STOP is asserted DECO Availability provides a status for whether each DECO is stopped or available. Any bit that is zero in this case indicates a DECO that is still running and needs to stop before the Debug Control Register can assert STOP_ACK.

## 13.15.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | NYA2 | NYA1 | NYA0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.15.4 Fields

| Field | Function |
|---|---|
| 31-3<br>— | Reserved. Always 0. |
| 2<br>NYA2 | This bit is set by software to start polling for the availability of DECO 2. This bit will be reset when DECO 2 is or becomes, available. |
| 1<br>NYA1 | This bit is set by software to start polling for the availability of DECO 1. This bit will be reset when DECO 1 is or becomes, available. |
| 0<br>NYA0 | This bit is set by software to start polling for the availability of DECO 0. This bit will be reset when DECO 0 is or becomes, available. |

# 13.16 DECO Reset Register (DRR)

## 13.16.1 Offset

| Register | Offset |
|---|---|
| DRR | 124h |

## 13.16.2 Function

The DECO Reset Register can be used to force a soft reset of one or more DECOs with appropriate status write back (error code 20h). Note that using this can result in lost DMA transactions and/or memory leaks. In some cases a soft reset of a DECO will not result in a status write back, or may not free a hung DECO. If a hung DECO cannot be freed via a soft DECO reset, then a software SEC reset or a POR will be required.

## 13.16.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | RST2 | RST1 | RST0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.16.4 Fields

| Field | Function |
|---|---|
| 31-3 | Reserved. Always 0. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 2<br>RST2 | Software writes a 1 to this bit to initiate a soft reset of DECO 2. This bit is self-clearing after one Clock cycle. |
| 1<br>RST1 | Software writes a 1 to this bit to initiate a soft reset of DECO 1. This bit is self-clearing after one clock cycle. |
| 0<br>RST0 | Software writes a 1 to this bit to initiate a soft reset of DECO 0. This bit is self-clearing after one clock cycle. |

## 13.17 DMA Control Register (DMAC - DMA_CTRL)

### 13.17.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMAC | 204h | DMA_CTRL is an alias of DMAC. DMAC is provided only for backward compatibility. Its usage is deprecated. |
| DMA_CTRL | 504h | For new software the usage of DMA_CTRL is preferred over DMAC. |

### 13.17.2 Function

The DMA Control register is used to configure the behavior of the DMA engine.

**NOTE**

Note that for backward compatibility the same registers are readable at two different addresses. The preferred addresses are in the range 00500..005FF. Usage of the legacy addresses in the range 00240..002CF is deprecated.

### 13.17.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | WSE | RSE |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

### 13.17.4 Fields

| Field | Function |
|---|---|
| 31-2<br>— | Reserved |
| 1<br>WSE | Write Safe Enable. When WSE=1, the Write Safe feature is enabled for the DMA engine(s). When WSE=0, the Write Safe feature is disabled for the DMA engine(s). (see DMA interface write-safe transactions |
| 0<br>RSE | Read Safe Enable. When RSE=1, the Read Safe feature is enabled for the DMA engine(s). When RSE=0, the Read Safe feature is disabled for the DMA engine(s). (see DMA read-safe transactions |

## 13.18 Peak Bandwidth Smoothing Limit Register (PBSL)

### 13.18.1 Offset

| Register | Offset |
|---|---|
| PBSL | 220h |

### 13.18.2 Function

The Peak Bandwidth Smoothing Limit Register is used to limit the maximum bus bandwidth consumed by SEC.

### 13.18.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | PBSL | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.18.4 Fields

| Field | Function |
|---|---|
| 31-7<br>— | Reserved |
| 6-0<br>PBSL | Whenever the number of outstanding AXI read bursts exceeds the value programmed in this field, the QI and Job Rings will be prevented from issuing additional AXI reads. While the number of outstanding AXI read burst exceeds the PBSL, DECOs may continue to issue additional AXI read requests. The QI and Job Rings will be allowed to issue additional AXI reads only when the number of outstanding AXI read bursts drops to, or below, the PBSL. Throttling the AXI reads reduces the SEC peak bandwidth on the AXI bus, and giving priority to DECOs improves SEC performance when SEC is heavily loaded with jobs. A limit of PBSL=0 indicates that no AXI read smoothing will be performed. |

## 13.19 DMA0_AIDL_MAP_MS (DMA0_AIDL_MAP_MS)

### 13.19.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA0_AIDL_MAP_MS | 240h | Mapping for DMA AXI IDs 7 ... 4 |

### 13.19.2 Function

The four registers DMAn_AIDL_MAP_MS, DMAn_AIDL_MAP_LS, DMAn_AIDM_MAP_MS and DMAn_AIDM_MAP_LS show the mapping of AXI transaction IDs to SEC internal blocks. These assignments are made via hardwired signals and are SoC-specific. The value of each 8-bit field indicates the internal ID of the SEC block that will use the AXI ID corresponding to the field. For example, AID2BID=00001000 means that AXI ID 2 (0010) will be used for all AXI transactions by DECO0 (internal block ID 00001000). (Note that the DMAn AXI ID Enable Register shows which of the 16 possible AXI transaction IDs are available for use by the DMA. If a particular AXI transaction ID is disabled, then the corresponding AIDxBID field will read as 00000000.)

**NOTE**

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

The SEC internal block IDs are encoded as follows:

| Internal Block ID | Internal Logic Block |
|---|---|
| 00000001b | Job Rings (The block ID for Job Ring 0 is used to represent all of the Job Rings.) |
| 00000100b | Burst Buffer |
| 00000111b | Queue Interface |
| 00001000b | DECO0 |
| 00001001b | DECO1 |
| 00001010b | DECO2 |
| All other values are reserved. | |

**NOTE**

For backward compatibility the same registers are readable at two different addresses. The preferred addresses are in the range 00500..005FF. The addresses in the range 00240..002CF are deprecated.

### 13.19.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID7_BID | | | | | | | | AID6_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID5_BID | | | | | | | | AID4_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

### 13.19.4 Fields

| Field | Function |
|---|---|
| 31-24<br>AID7_BID | This field shows the SEC Block ID that uses AXI ID 7. |
| 23-16<br>AID6_BID | This field shows the SEC Block ID that uses AXI ID 6. |
| 15-8<br>AID5_BID | This field shows the SEC Block ID that uses AXI ID 5. |
| 7-0<br>AID4_BID | This field shows the SEC Block ID that uses AXI ID 4. |

## 13.20 DMA0_AIDL_MAP_LS (DMA0_AIDL_MAP_LS)

### 13.20.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA0_AIDL_MAP_LS | 244h | Mapping for DMA AXI IDs 3 ... 0 |

## 13.20.2 Function

The four registers DMAn_AIDL_MAP_MS, DMAn_AIDL_MAP_LS, DMAn_AIDM_MAP_MS and DMAn_AIDM_MAP_LS show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMAn_AID_7_4_MAP for additional details.

**NOTE**

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

## 13.20.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID3_BID | | | | | | | | AID2_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID1_BID | | | | | | | | AID0_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

## 13.20.4 Fields

| Field | Function |
|---|---|
| 31-24<br>AID3_BID | This field shows the SEC Block ID that uses AXI ID 3. |
| 23-16<br>AID2_BID | This field shows the SEC Block ID that uses AXI ID 2. |
| 15-8<br>AID1_BID | This field shows the SEC Block ID that uses AXI ID 1. |
| 7-0<br>AID0_BID | This field shows the SEC Block ID that uses AXI ID 0. |

# 13.21 DMA0_AIDM_MAP_MS (DMA0_AIDM_MAP_MS)

## 13.21.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA0_AIDM_MAP_MS | 248h | Mapping for DMA AXI IDs 15 ... 12 |

## 13.21.2 Function

The four registers DMAn_AIDL_MAP_MS, DMAn_AIDL_MAP_LS, DMAn_AIDM_MAP_MS and DMAn_AIDM_MAP_LS show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMAn_AID_7_4_MAP for additional details.

**NOTE**

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

## 13.21.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID15_BID | | | | | | | | AID14_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID13_BID | | | | | | | | AID12_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

## 13.21.4 Fields

| Field | Function |
|---|---|
| 31-24<br>AID15_BID | This field shows the SEC Block ID that uses AXI ID 15. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 23-16<br>AID14_BID | This field shows the SEC Block ID that uses AXI ID 14. |
| 15-8<br>AID13_BID | This field shows the SEC Block ID that uses AXI ID 13. |
| 7-0<br>AID12_BID | This field shows the SEC Block ID that uses AXI ID 12. |

## 13.22 DMA0_AIDM_MAP_LS (DMA0_AIDM_MAP_LS)

### 13.22.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA0_AIDM_MAP_LS | 24Ch | Mapping for DMA AXI IDs 11 ... 8 |

### 13.22.2 Function

The four registers DMAn_AIDL_MAP_MS, DMAn_AIDL_MAP_LS, DMAn_AIDM_MAP_MS and DMAn_AIDM_MAP_LS show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMAn_AID_7_4_MAP for additional details.

**NOTE**

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

### 13.22.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID11_BID | | | | | | | | AID10_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID9_BID | | | | | | | | AID8_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

### 13.22.4 Fields

| Field | Function |
|---|---|
| 31-24<br>AID11_BID | This field shows the SEC Block ID that uses AXI ID 11. |
| 23-16<br>AID10_BID | This field shows the SEC Block ID that uses AXI ID 10. |
| 15-8<br>AID9_BID | This field shows the SEC Block ID that uses AXI ID 9. |
| 7-0<br>AID8_BID | This field shows the SEC Block ID that uses AXI ID 8. |

## 13.23 DMA0 AXI ID Enable Register (DMA0_AID_ENB)

### 13.23.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA0_AID_ENB | 250h | Use of this register alias is deprecated. Instead, use the register alias DMA_X_AID_15_0_EN |

## 13.23.2 Function

The DMA AXI ID Enable register can be read to determine which AXI transaction IDs are available for use by the DMAs. These enables are configured via hardwired signals and are SOC-specific. The DMA will use a unique AXI ID for each SEC internal connected to it. The assignments are made using the lowest-numbered, available IDs.

**NOTE**

Note that for backward compatibility the same register is readable at two different addresses.

**NOTE**

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

## 13.23.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID15E | AID14E | AID13E | AID12E | AID11E | AID10E | AID9E | AID8E | AID7E | AID6E | AID5E | AID4E | AID3E | AID2E | AID1E | AID0E |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

## 13.23.4 Fields

| Field | Function |
|---|---|
| 31-16<br>— | Reserved. |
| 15<br>AID15E | If AID15E=1 then AXI ID 15 is enabled for this DMA engine. |
| 14<br>AID14E | If AID14E=1 then AXI ID 14 is enabled for this DMA engine. |
| 13 | If AID13E=1 then AXI ID 13 is enabled for this DMA engine. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| AID13E | |
| 12<br>AID12E | If AID12E=1 then AXI ID 12 is enabled for this DMA engine. |
| 11<br>AID11E | If AID11E=1 then AXI ID 11 is enabled for this DMA engine. |
| 10<br>AID10E | If AID10E=1 then AXI ID 10 is enabled for this DMA engine. |
| 9<br>AID9E | If AID9E=1 then AXI ID 9 is enabled for this DMA engine. |
| 8<br>AID8E | If AID8E=1 then AXI ID 8 is enabled for this DMA engine. |
| 7<br>AID7E | If AID7E=1 then AXI ID 7 is enabled for this DMA engine. |
| 6<br>AID6E | If AID6E=1 then AXI ID 6 is enabled for this DMA engine. |
| 5<br>AID5E | If AID5E=1 then AXI ID 5 is enabled for this DMA engine. |
| 4<br>AID4E | If AID4E=1 then AXI ID 4 is enabled for this DMA engine. |
| 3<br>AID3E | If AID3E=1 then AXI ID 3 is enabled for this DMA engine. |
| 2<br>AID2E | If AID2E=1 then AXI ID 2 is enabled for this DMA engine. |
| 1<br>AID1E | If AID1E=1 then AXI ID 1 is enabled for this DMA engine. |
| 0<br>AID0E | If AID0E=1 then AXI ID 0 is enabled for this DMA engine. |

## 13.24 DMA0 AXI Read Timing Check Register (DMA0_ARD_TC)

### 13.24.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA0_ARD_TC | 260h | The addresses of the two halves of the register are unaffected by the endianness configuration. |

## 13.24.2 Function

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

After the AXI Read Latency Register is read, the sample count, late count, and latency total are cleared and read timing checks resume with the next AXI read.

**NOTE**

Note that the DMA_X_ARTC_CTL register located in the address range 00530..005DF provides functionality similar to the DMAn_ARD_TC register located in the address range 00260..002EF. Writing to either register affects the corresponding fields in the other register. But note that some fields in the DMAn_ARD_TC register have been rearranged in the DMA_X_ARTC_CTL register or moved to the new DMA_X_ARTC_LC register or DMA_X_ARTC_SC register. The preferred registers are located in the address range 00530..005DF. The use of the DMAn_ARD_TC register located in the address range 00260..002EF is deprecated.

## 13.24.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ARTCE | ARCT | ARTT | ARTL | ARL | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | ARLC | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ARLC | | | | | | | | Reserved | | | | ARSC | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ARSC | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.24.4 Fields

| Field | Function |
|---|---|
| 63<br>ARTCE | AXI Read Timing Check Enable. When ARTCE=0, ARL, ARLC, and ARSC in DMAn_ARD_TC and SARL in DMAn_SARL and ARL in DMA_X_ARTC_CTL, ARLC in DMA_X_ARTC_LC, ARSC in DMA_X_ARTC_SC and SARL in DMA_X_ARTC_LC are writeable. When ARTCE=1, AXI read timing checks are enabled and these fields are read-only. Note that writing ARTCE in either DMAn_ARD_TC or DMA_X_ARTC_CTL has the same effect. |
| 62<br>ARCT | AXI Read Counter Test. When ARCT=1, ARLC and ARSC in DMAn_ARD_TC, ARLC in DMA_X_ARTC_LC, ARSC in DMA_X_ARTC_SC, and SARL in DMAn_SARL and DMA_X_ARTC_LC, are not cleared when timing checks are enabled and when timing checks resume after reading DMAn_ARD_TC and DMAn_SARL or DMA_X_ARTC_LC, DMA_X_ARTC_SC and DMA_X_SARL. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested. |
| 61<br>ARTT | AXI Read Timer Test. When ARTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. The count can optionally be modified to count until the last beat of data instead of the first by setting the ARTL (AXI Read Timer Last) bit. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field ARTT in the DMAn_ARD_TC register is aliased to bit field ARTT in the DMA_X_ARTC_CTL register, i.e. writing to either ARTT bit field alters the ARTT value in the other register. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 60<br>ARTL | AXI Read Timer Last. This bit controls whether the last or first beat of data signals the end of a transaction's counter measurement.<br>0b - A read transaction counter measurement is stopped when the first beat of data arrives<br>1b - A read transaction counter measurement is stopped when the last beat of data arrives |
| 59-48<br>ARL | AXI Read Limit. The AXI Read Timer measures latency by counting the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. If the latency equals or exceeds the AXI Read Limit, the read response is considered late and the AXI Read Late Count (ARLC) is incremented along with the AXI Read Sample Count (ARSC). The latency is added to the Sum of AXI Read Latencies (SARL) in DMAn_SARL /DMA_X_ARTC_LAT. This field is writeable only when ARTCE=0.Note that bit field ARL in the DMAn_ARD_TC register is aliased to bit field ARL in the DMA_X_ARTC_CTL register, i.e. writing to either ARL bit field alters the ARL value in the other register. |
| 47-44<br>— | Reserved |
| 43-24<br>ARLC | AXI Read Late Count. This field is incremented whenever the AXI Read Timer equals or exceeds the AXI Read Limit. AXI read timing checks are suspended when ARLC=FFFFFh. This field is writeable only when ARTCE=0. |
| 23-20<br>— | Reserved |
| 19-0<br>ARSC | AXI Read Sample Count. This field is incremented after each read timing check. AXI read timing checks are suspended when ARSC=FFFFFh. This field is writeable only when ARTCE=0. |

## 13.25 DMA0 Read Timing Check Latency Register (DMA0_ ARD_LAT)

### 13.25.1 Offset

| Register | Offset |
|---|---|
| DMA0_ARD_LAT | 26Ch |

### 13.25.2 Function

While AXI Read Timing Checks are enabled and not suspended, this register maintains a running total of AXI read latencies.

**NOTE**

Note that the DMAn_ARTC_LAT register located in the address range 0053C..005DF is identical to the DMAn_SARL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

## 13.25.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SARL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SARL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.25.4 Fields

| Field | Function |
|---|---|
| 31-0<br>SARL | Sum of the AXI Read Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Read Latencies (SARL) in DMAn_SARL. This field is writeable only when ARTCE=0. |

# 13.26 DMA0 AXI Write Timing Check Register (DMA0_AWR_TC)

## 13.26.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA0_AWR_TC | 270h | The addresses of the two halves of the register are unaffected by the endianness configuration. |

### 13.26.2 Function

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the write response. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. This count can optionally be modified to count until the last beat of data by setting the ARTL (AXI Read Timer Last) bit. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is read

After the AXI Write Latency Register is read, the sample count, late count, and latency total are cleared and write timing checks resume with the next AXI write.

**NOTE**

Note that the DMA_X_AWTC_CTL register located in the address range 00540..005DF provides functionality similar to the DMAn_AWR_TC register located in the address range 00270..002EF. Writing to either register affects the corresponding fields in the other register. But note that some fields in the DMAn_AWR_TC register have been rearranged in the DMA_X_AWTC_CTL register or moved to the new DMA_X_TC_SAWL register or DMA_X_AWTC_SC register. The preferred registers are located in the address range 00540..005DF. The use of the DMAn_AWR_TC register located in the address range 00270..002EF is deprecated.

## 13.26.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AWTCE | AWCT | AWTT | Reserved | AWL | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | AWLC | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AWLC | | | | | | | | Reserved | | | | AWSC | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AWSC | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.26.4 Fields

| Field | Function |
|---|---|
| 63<br>AWTCE | AXI Write Timing Check Enable. When AWTCE=0, AWL, AWLC, and AWSC in DMAn_AWR_TC and SAWL in DMAn_AWL and AWL in DMA_X_AWTC_CTL, AWLC in DMA_X_TC_SAWL, AWSC in DMA_X_AWTC_SC and SAWL in DMA_X_TC_SAWL are writeable. When AWTCE=1, AXI write timing checks are enabled and these fields are read-only. Note that writing AWTCE in either DMAn_AWR_TC or DMA_X_AWTC_CTL has the same effect. |
| 62<br>AWCT | AXI Write Counter Test. When AWCT=1, AWLC and AWSC in DMAn_AWR_TC, AWLC in DMA_X_TC_SAWL, AWSC in DMA_X_AWTC_SC, and SAWL in DMAn_AWL and DMA_X_TC_SAWL, are not cleared when timing checks are enabled and when timing checks resume after writing DMAn_AWR_TC and DMAn_AWL or DMA_X_TC_SAWL, DMA_X_AWTC_SC and DMA_X_AWL. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested. |
| 61<br>AWTT | AXI Write Timer Test. When AWTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding Write data transaction. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field AWTT in the DMAn_AWR_TC register is aliased to bit field AWTT in the DMA_X_AWTC_CTL register, i.e. writing to either AWTT bit field alters the AWTT value in the other register. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 60<br>— | Reserved |
| 59-48<br>AWL | AXI Write Limit. The AXI Write Timer measures latency by counting the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding Write data transaction. If the latency equals or exceeds the AXI Write Limit, the Write response is considered late and the AXI Write Late Count (AWLC) is incremented along with the AXI Write Sample Count (AWSC). The latency is added to the Sum of AXI Write Latencies (SAWL) in DMAn_AWL /DMA_X_AWTC_LAT. This field is writeable only when ARTCE=0.Note that bit field AWL in the DMAn_AWR_TC register is aliased to bit field AWL in the DMA_X_AWTC_CTL register, i.e. writing to either AWL bit field alters the AWL value in the other register. |
| 47-44<br>— | Reserved |
| 43-24<br>AWLC | AXI Write Late Count. This field is incremented whenever the AXI Write Timer equals or exceeds the AXI Write Limit. AXI write timing checks are suspended when AWLC=FFFFFh. This field is writeable only when AWTCE=0. |
| 23-20<br>— | Reserved |
| 19-0<br>AWSC | AXI Write Sample Count. This field is incremented after each write timing check. AXI write timing checks are suspended when AWSC=FFFFFh. This field is writeable only when AWTCE=0. |

## 13.27 DMA0 Write Timing Check Latency Register (DMA0_ AWR_LAT)

### 13.27.1 Offset

| Register | Offset |
|---|---|
| DMA0_AWR_LAT | 27Ch |

### 13.27.2 Function

While AXI Write Timing Checks are enabled and not suspended, this register maintains a running total of AXI write latencies.

**NOTE**

Note that the DMAn_AWTC_LAT register located in the address range 0053C..005DF is identical to the DMAn_AWL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

### 13.27.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SAWL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SAWL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.27.4 Fields

| Field | Function |
|---|---|
| 31-0<br>SAWL | Sum of the AXI Write Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Write Latencies (SAWL) in DMAn_AWL. This field is writeable only when AWTCE=0. |

## 13.28 Manufacturing Protection Private Key Register (MPPKR0 - MPPKR63)

### 13.28.1 Offset

For a = 0 to 63:

| Register | Offset |
|---|---|
| MPPKRa | 300h + (a × 1h) |

### 13.28.2 Function

The Manufacturing Protection Private Key register is used when authenticating the SOC to the OEM's server. This authentication process can be used to ensure that the SOC is a genuine NXP part, is the correct part type, has been properly configured via fuses, is running authenticated OEM software, and is currently in the Secure or Trusted mode. The SOC attests to all this by signing a message using the private key stored in the MPPKR. Software running on the SOC then sends this attestation message to the OEM's server. The OEM's server can verify that all this information is correct by verifying the signature over the signed message. The server can then be assured that it is safe to download proprietary data to the SOC over a secured connection.

### 13.28.3 Diagram

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | MPPrivK | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.28.4 Fields

| Field | Function |
|---|---|
| 7-0<br>MPPrivK | MPPrivK. The 512-bit Manufacturing Protection Private Key. |

## 13.29 Manufacturing Protection Message Register (MPMR0 - MPMR31)

### 13.29.1 Offset

For a = 0 to 31:

| Register | Offset |
|---|---|
| MPMRa | 380h + (a × 1h) |

### 13.29.2 Function

The Manufacturing Protection Message register is used when authenticating the SOC to the OEM's server. This authentication process can be used to ensure that the SOC is a genuine NXP part, is the correct part type, has been properly configured via fuses, is running authenticated OEM software, and is currently in the Secure or Trusted mode. The SOC attests to this by signing a message using the private key stored in the MPPKR. The message is composed, in part, of the content of the MPMR. The value in the MPMR is written by ROM-resident boot software. The value normally includes the hash of the public key used to verify the signature over the signed code image. Software running on the SOC then sends this signed message to the OEM's server. The OEM's server can confirm that all this information is correct by verifying the signature over the signed message. The server can then be assured that it is safe to download proprietary data to the SOC over a secured connection.

### 13.29.3 Diagram

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R<br>W | MPMSG | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.29.4 Fields

| Field | Function |
|---|---|
| 7-0<br>MPMSG | Holds 256 bits of message data that will be prepended to the input data to the MPSIGN operation. When accessed via the register bus this should be treated as a byte array (although it must be accessed as eight 32-bit words). |

## 13.30 Manufacturing Protection Test Register (MPTESTR0 - MPTESTR31)

### 13.30.1 Offset

For a = 0 to 31:

| Register | Offset |
|---|---|
| MPTESTRa | 3C0h + (a × 1h) |

### 13.30.2 Function

The Manufacturing Protection TEST register is used only for hardware verification.

### 13.30.3 Diagram

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R | TEST_VALUE | | | | | | | |
| W | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.30.4 Fields

| Field | Function |
|---|---|
| 7-0<br>TEST_VALUE | TEST_VALUE. When accessed via the register bus this should be treated as a byte array with the first byte in offset 3C0h (although it must be accessed as eight 32-bit words).( |

## 13.31 Job Descriptor Key Encryption Key Register (JDKEKR0 - JDKEKR7)

### 13.31.1 Offset

For a = 0 to 7:

| Register | Offset |
|---|---|
| JDKEKRa | 400h + (a × 4h) |

## 13.31.2 Function

The Job Descriptor Key Encryption Key Register contains the Job Descriptor Key Encryption Key (JDKEK), which can be used when encrypting or decrypting Black Keys (see Black keys). Since Black Keys are not intended for storage of keys across SOC power cycles (SEC's Blob mechanism is intended for this purpose), the value in the JDKEKR is not preserved at SOC power-down. Instead, a new 256-bit secret value is loaded into the JDKEKR from the RNG for use during the new power-on session. The JDKEKR cannot be read or written from the register bus while SEC is in Secure Mode or Trusted Mode, but JDKEKR can be read and written while SEC is in Non-secure Mode.The JDKEK is loaded by executing a special descriptor, which can be run in any security mode. (see RNG functional description )

Note that the Secure Mode/Trusted Mode value in JDKEKR is not available when SEC is in Non-secure Mode because the only possible transitions between Trusted Mode or Secure Mode that lead to Non-secure Mode cause SEC to pass through Fail Mode, and JDKEKR is cleared whenever SEC enters Fail Mode.

The Job Descriptor Key Encryption Key is 256 bits, so it is read or written via eight 32-bit word addresses. The first byte is in offset 400h.

**NOTE**

The register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG.

## 13.31.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | JDKEK | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | JDKEK | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

### 13.31.4 Fields

| Field | Function |
|---|---|
| 31-0<br>JDKEK | The 256-bit Job Descriptor Key Encryption Key used to encrypt and decrypt Black Keys. |

## 13.32 Trusted Descriptor Key Encryption Key Register (TDKEKR0 - TDKEKR7)

### 13.32.1 Offset

For a = 0 to 7:

| Register | Offset |
|---|---|
| TDKEKRa | 420h + (a × 4h) |

### 13.32.2 Function

The Trusted Descriptor Key Encryption Key Register contains the Trusted Descriptor Key Encryption Key (TDKEK), which can be used when encrypting or decrypting Black Keys (see Black keys. The TDKEKR operates exactly like the JDKEKR, except that the TDKEKR is usable only by Trusted Descriptors. This allows Trusted Descriptors to protect particularly sensitive keys from access by Job Descriptors. Trusted Descriptors can use either the JDKEKR or the TDKEKR, so Trusted Descriptors can be used to derive non-Trusted Black keys for use by Job Descriptors from Trusted Black Keys that contain master secrets. The Trusted Descriptor Key Encryption Key is 256 bits, so it is read or written via eight 32-bit word addresses. The first byte is in offset 420h.

**NOTE**

The register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG.

### 13.32.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | TDKEK | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | TDKEK | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

### 13.32.4 Fields

| Field | Function |
|---|---|
| 31-0<br>TDKEK | The 256-bit Trusted Descriptor Key Encryption Key used to encrypt and decrypt Black Keys. |

## 13.33 Trusted Descriptor Signing Key Register (TDSKR0 - TDSKR7)

### 13.33.1 Offset

For a = 0 to 7:

| Register | Offset |
|---|---|
| TDSKRa | 440h + (a × 4h) |

### 13.33.2 Function

The Trusted Descriptor Signing Key Register contains the TDSK, which is used to generate and verify signatures on Trusted Descriptors. The TDSKR is loaded in the same fashion as the JDKEK. The TDSK is 256 bits, so it is read or written via eight 32-bit word addresses. The first byte is in offset 440h.

**NOTE**

The register resets to all 0 at POR, but then is immediately loaded with a random value obtained from the RNG.

### 13.33.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | TDSK | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | TDSK | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

### 13.33.4 Fields

| Field | Function |
|---|---|
| 31-0<br>TDSK | The 256-bit Trusted Descriptor Signing Key used to sign and verify Trusted Descriptors. |

## 13.34 Secure Key Nonce Register (SKNR)

### 13.34.1 Offset

| Register | Offset | Description |
|---|---|---|
| SKNR | 4E0h | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R). |

### 13.34.2 Function

The Secure Key Nonce Register holds a nonce value that is used for Black Key encryption. To ensure that a nonce is never reused during a power-on session, the nonce is used and incremented whenever a Black Key is encrypted using AES-CCM encryption (i.e. a FIFO STORE with EKT=1, of the PKHA E Memory, the Class 1 Key Register or the Class 2 Key Register.) The SKNR is reset to all 0 at power on reset or when SEC enters Fail mode, but it is not reset at software-initiated SEC reset. Since the SKNR holds more than 32 bits, it is accessed over the IP bus as two 32-bit words. Note that two or more DECOs could encrypt a Black Key at the same time and since all DECOs share the same SKNR, the DECO identification number is concatenated with the value in the SKNR to ensure that each Black Key is encrypted using a unique nonce.

**NOTE**

This register is writable only when SEC is in NonSecure mode.

### 13.34.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | SK_NONCE_MS | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SK_NONCE_LS | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SK_NONCE_LS | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.34.4 Fields

| Field | Function |
|---|---|
| 63-46<br>— | Reserved |
| 45-32<br>SK_NONCE_MS | Secure Key Nonce - Most Significant Bits. This field holds the 14 most-significant bits of the auto-incrementing secure key nonce field. See the description of the SK_NONCE_LS field for more information. |
| 31-0<br>SK_NONCE_LS | Secure Key Nonce - Least Significant Bits. This field holds the 32 least-significant bits of the auto-incrementing secure key nonce field. The actual nonce value that is used during AES-CCM encryption of Black Keys consists of the SK_NONCE_MS, the SK_NONCE_LS and the DECO_ID. The DECO_ID indicates which DECO is requesting a nonce. If two or more DECOs happen to request a nonce during the same clock cycle, this ensures that each DECO receives a different nonce. |

## 13.35 DMA Status Register (DMA_STA)

### 13.35.1 Offset

| Register | Offset |
|---|---|
| DMA_STA | 50Ch |

### 13.35.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | DMA0_IDLE | Reserved | | DMA0_ETIF | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.35.3 Fields

| Field | Function |
|---|---|
| 31-8<br>— | Reserved |
| 7<br>DMA0_IDLE | DMA0 is idle. DMA0's command queue is empty. |
| 6-5<br>— | Reserved |
| 4-0<br>DMA0_ETIF | DMA0 External Transactions in Flight. DMA0_ETIF indicates the number of transactions the DMA0 engine currently has in flight on SEC's external AXI bus. |

# 13.36 DMA_X_AID_7_4_MAP (DMA_X_AID_7_4_MAP)

## 13.36.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA_X_AID_7_4_MAP | 510h | Mapping for DMA AXI IDs 7 ... 4 |

## 13.36.2 Function

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to SEC internal blocks. These assignments are made via hardwired signals and are SOC-specific. The value of each 8-bit field indicates the internal ID of the SEC block that will use the AXI ID corresponding to the field. For example, AID2BID=00001000 means that AXI ID 2 (0010) will be used for all AXI transactions by DECO0 (internal block ID 00001000). (Note that the DMA_X_ AXI ID Enable Register shows which of the 16 possible AXI transaction IDs are available for use by the DMA. If a particular AXI transaction ID is disabled, then the corresponding AIDxBID field will read as 00000000.)

The SEC internal block IDs are encoded as follows:

| Internal Block ID | Internal Logic Block |
|---|---|
| 00000001b | Job Rings (The block ID for Job Ring 0 is used to represent all of the Job Rings.) |
| 00000100b | Burst Buffer |
| 00000111b | Queue Interface |
| 00001000b | DECO0 |
| 00001001b | DECO1 |
| 00001010b | DECO2 |
| All other values are reserved. | |

**NOTE**

For backward compatibility the same registers are readable at two different addresses. The preferred addresses are in the range 00500..005FF. The addresses in the range 00240..002CF are deprecated.

**NOTE**

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

## 13.36.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID7_BID | | | | | | | | AID6_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID5_BID | | | | | | | | AID4_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

## 13.36.4 Fields

| Field | Function |
|---|---|
| 31-24<br>AID7_BID | This field shows the SEC Block ID that uses AXI ID 7. |
| 23-16 | This field shows the SEC Block ID that uses AXI ID 6. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| AID6_BID | |
| 15-8<br>AID5_BID | This field shows the SEC Block ID that uses AXI ID 5. |
| 7-0<br>AID4_BID | This field shows the SEC Block ID that uses AXI ID 4. |

## 13.37 DMA_X_AID_3_0_MAP (DMA_X_AID_3_0_MAP)

### 13.37.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA_X_AID_3_0_MAP | 514h | Mapping for DMA AXI IDs 3 ... 0 |

### 13.37.2 Function

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMA_X_AID_7_4_MAP for additional details.

**NOTE**

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

### 13.37.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID3_BID | | | | | | | | AID2_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID1_BID | | | | | | | | AID0_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

### 13.37.4 Fields

| Field | Function |
|---|---|
| 31-24<br>AID3_BID | This field shows the SEC Block ID that uses AXI ID 3. |
| 23-16<br>AID2_BID | This field shows the SEC Block ID that uses AXI ID 2. |
| 15-8<br>AID1_BID | This field shows the SEC Block ID that uses AXI ID 1. |
| 7-0<br>AID0_BID | This field shows the SEC Block ID that uses AXI ID 0. |

## 13.38 DMA_X_AID_15_12_MAP (DMA_X_AID_15_12_MAP)

### 13.38.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA_X_AID_15_12_MAP | 518h | Mapping for DMA AXI IDs 15 ... 12 |

### 13.38.2 Function

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMA_X_AID_7_4_MAP for additional details.

**NOTE**

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

### 13.38.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID15_BID | | | | | | | | AID14_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID13_BID | | | | | | | | AID12_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

### 13.38.4 Fields

| Field | Function |
|---|---|
| 31-24<br>AID15_BID | This field shows the SEC Block ID that uses AXI ID 15. |
| 23-16<br>AID14_BID | This field shows the SEC Block ID that uses AXI ID 14. |
| 15-8<br>AID13_BID | This field shows the SEC Block ID that uses AXI ID 13. |
| 7-0<br>AID12_BID | This field shows the SEC Block ID that uses AXI ID 12. |

# 13.39 DMA_X_AID_11_8_MAP (DMA_X_AID_11_8_MAP)

## 13.39.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA_X_AID_11_8_MAP | 51Ch | Mapping for DMA AXI IDs 11 ... 8 |

## 13.39.2 Function

The four registers DMA_X_AID_7_4_MAP, DMA_X_AID_3_0_MAP, DMA_X_AID_15_12_MAP and DMA_X_AID_11_8_MAP show the mapping of AXI transaction IDs to SEC internal blocks. See the description for register DMA_X_AID_7_4_MAP for additional details.

**NOTE**

The values read from this register are determined by hardwired inputs to SEC and are SoC-specific.

## 13.39.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID11_BID | | | | | | | | AID10_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID9_BID | | | | | | | | AID8_BID | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u | u |

## 13.39.4 Fields

| Field | Function |
|---|---|
| 31-24<br>AID11_BID | This field shows the SEC Block ID that uses AXI ID 11. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 23-16<br>AID10_BID | This field shows the SEC Block ID that uses AXI ID 10. |
| 15-8<br>AID9_BID | This field shows the SEC Block ID that uses AXI ID 9. |
| 7-0<br>AID8_BID | This field shows the SEC Block ID that uses AXI ID 8. |

## 13.40 DMA_X AXI ID Map Enable Register (DMA_X_AID_15_ 0_EN)

### 13.40.1 Offset

| Register | Offset | Description |
|---|---|---|
| DMA_X_AID_15_0_EN | 524h | For new software DMA_X_AID_15_0_EN (address 510h) should be used rather than DMA_0_AID_ENB (address 250h).<br>**NOTE:** The values read from this register are determined by hardwired inputs to SEC and are SoC-specific. |

### 13.40.2 Function

The DMA_X_AID_15_0_EN register can be read to determine which AXI transaction IDs are available for use by the DMAs. These enables are configured via hardwired signals and are SOC-specific. The DMA will use a unique AXI ID for each SEC internal connected to it. The assignments are made using the lowest-numbered, available IDs.

**NOTE**

Note that for backward compatibility the same information is readable at two different addresses, 250h and 524h. The 250h address is deprecated.

## 13.40.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AID15E | AID14E | AID13E | AID12E | AID11E | AID10E | AID9E | AID8E | AID7E | AID6E | AID5E | AID4E | AID3E | AID2E | AID1E | AID0E |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.40.4 Fields

| Field | Function |
|---|---|
| 31-16<br>— | Reserved. |
| 15<br>AID15E | If AID15E=1 then AXI ID 15 is enabled for this DMA engine. |
| 14<br>AID14E | If AID14E=1 then AXI ID 14 is enabled for this DMA engine. |
| 13<br>AID13E | If AID13E=1 then AXI ID 13 is enabled for this DMA engine. |
| 12<br>AID12E | If AID12E=1 then AXI ID 12 is enabled for this DMA engine. |
| 11<br>AID11E | If AID11E=1 then AXI ID 11 is enabled for this DMA engine. |
| 10<br>AID10E | If AID10E=1 then AXI ID 10 is enabled for this DMA engine. |
| 9<br>AID9E | If AID9E=1 then AXI ID 9 is enabled for this DMA engine. |
| 8<br>AID8E | If AID8E=1 then AXI ID 8 is enabled for this DMA engine. |
| 7<br>AID7E | If AID7E=1 then AXI ID 7 is enabled for this DMA engine. |
| 6<br>AID6E | If AID6E=1 then AXI ID 6 is enabled for this DMA engine. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 5<br>AID5E | If AID5E=1 then AXI ID 5 is enabled for this DMA engine. |
| 4<br>AID4E | If AID4E=1 then AXI ID 4 is enabled for this DMA engine. |
| 3<br>AID3E | If AID3E=1 then AXI ID 3 is enabled for this DMA engine. |
| 2<br>AID2E | If AID2E=1 then AXI ID 2 is enabled for this DMA engine. |
| 1<br>AID1E | If AID1E=1 then AXI ID 1 is enabled for this DMA engine. |
| 0<br>AID0E | If AID0E=1 then AXI ID 0 is enabled for this DMA engine. |

## 13.41 DMA_X AXI Read Timing Check Control Register (DMA_X_ARTC_CTL)

### 13.41.1 Offset

| Register | Offset |
|---|---|
| DMA_X_ARTC_CTL | 530h |

### 13.41.2 Function

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. This count can optionally be modified to count until the last beat of data by setting the ARTL (AXI Read Timer Last) bit. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

After the DMA_X AXI Read Latency Register or DMA_X Read Timing Check Latency Register is read, the sample count, late count, and latency total are cleared and read timing checks resume with the next AXI read.

**NOTE**

> Note that the DMA_X_ARTC_CTL register located in the address range 00530..005DF provides functionality similar to the DMAn_ARD_TC register located in the address range 00260..002EF. Some of the fields are aliased, i.e. writing to these fields in either register affects the corresponding fields in the other register. But note that some fields in the DMAn_ARD_TC register have been rearranged in the DMA_X_ARTC_CTL register or moved to the new DMA_X_ARTC_LC register or the DMA_X_ARTC_SC register.

## 13.41.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ARTCE | ARCT | ARTT | ARTL | ARL | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | ART | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.41.4 Fields

| Field | Function |
|---|---|
| 31<br>ARTCE | AXI Read Timing Check Enable. When ARTCE=0, ARL, ARLC, and ARSC in DMA_X_ARTC and SARL in DMAn_ARD_LAT and ARL in DMA_X_ARTC_CTL, ARLC in DMA0_ARTC_LC, ARSC in DMA0ARTC_SC and SARL in DMA0ARL_LAT are writeable. When ARTCE=1, AXI read timing checks are enabled and these fields are read-only.<br>**NOTE:** Note that writing ARTCE in either DMA_X_ARTC_B or DMA_X_ARTC_CTL has the same effect. |
| 30<br>ARCT | AXI Read Counter Test. When ARCT=1, ARLC and ARSC in DMA_X_ARTC_CTL, ARLC in DMA_X_ARTC_LC, ARSC in DMA_X_ARTC_SC, and SARL in DMAn_ARD_LAT and DMA_X_ARTC_LC, are not cleared when timing checks are enabled and when timing checks resume |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | after reading DMAn_ARD_TC and DMAn_ARD_LAT or DMA_X_ARTC_LC, DMA_X_ARTC_SC and DMAn_ARD_LAT. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested. |
| 29<br>ARTT | AXI Read Timer Test. When ARTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field ARTT in the DMA_X_ARTC_CTL register is aliased to bit field ARTT in the DMAn_ARD_TC register, i.e. writing to either ARTT bit field alters the ARTT value in the other register. |
| 28<br>ARTL | AXI Read Timer Last. This bit controls whether the last or first beat of data signals the end of a transaction's counter measurement.<br>0b - A read transaction counter measurement is stopped when the first beat of data arrives<br>1b - A read transaction counter measurement is stopped when the last beat of data arrives |
| 27-16<br>ARL | AXI Read Limit. The AXI Read Timer measures latency by counting the number of AXI clock cycles from the AXI read address transaction to the beginning of the corresponding read data transaction. If the latency equals or exceeds the AXI Read Limit, the read response is considered late and the AXI Read Late Count (ARLC) is incremented along with the AXI Read Sample Count (ARSC). The latency is added to the Sum of AXI Read Latencies (SARL) in DMAn_ARD_LAT/DMA_X_ARTC_LAT. This field is writeable only when ARTCE=0. Note that bit field ARL in the DMA_X_ARTC_CTL register is aliased to bit field ARL in the DMAn_ARD_TC register, i.e. writing to either ARL bit field alters the ARL value in the other register. |
| 15-12<br>— | Reserved |
| 11-0<br>ART | AXI Read Timer. The number of AXI clock cycles from the latest external AXI read address transaction initiated by this DMA to the beginning of the corresponding read data transaction. This field is writeable only when ARTCE=0. |

## 13.42 DMA_X AXI Read Timing Check Late Count Register (DMA_X_ARTC_LC)

### 13.42.1 Offset

| Register | Offset |
|---|---|
| DMA_X_ARTC_LC | 534h |

## 13.42.2 Function

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the ARTC_LC register is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

**NOTE**

Note that the DMA_X_ARTC_LC register provides functionality similar to the AXI Read Timing Late Check fields in the DMAn_ARD_TC register located in the address range 00260..002EF, but the fields have been rearranged. Usage of the DMAn_ARD_TC register is deprecated.

## 13.42.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | ARLC | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ARLC | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.42.4 Fields

| Field | Function |
|---|---|
| 31-20<br>— | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 19-0<br>ARLC | AXI Read Late Count. This field is incremented each time that the ART field exceeds the ARL field in the DMA_X AXI Read Timing Check Control Register. AXI read timing checks are suspended when ARLC=FFFFFh. Note that this field is an alias of the ARLC field in the DMAn_ARD_TC Register. Reading or writing the ARLC field in either the 00200 address block or the 00500 address block will yield identical results, and the value written can be read from the other register. When DMAn_ARD_TC/ DMA_X_ARTC_TC[ARTCE]=0, the ARLC bit field in DMAn_ARD_TC and the DMA_X_ARTC_LC register are writeable. When ARTCE=1, AXI read timing checks are enabled and the ARLC bit fields are read-only. |

## 13.43 DMA_X AXI Read Timing Check Sample Count Register (DMA_X_ARTC_SC)

### 13.43.1 Offset

| Register | Offset |
|---|---|
| DMA_X_ARTC_SC | 538h |

### 13.43.2 Function

When AXI Read Timing Checks are enabled, the DMA measures the latencies of selected AXI read transactions. A timer measures the latency by counting the number of AXI clock cycles from the read address transaction to the beginning of the corresponding read data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI read. Timing checks are suspended when:

- the AXI read sample count value reaches FFFFFh, or
- the AXI read latency total reaches FFFFFFFFh, or
- the AXI Read Timing Check Register is read

After the AXI Read Latency Register is read, the sample count, late count, and latency total are cleared and read timing checks resume with the next AXI read.

**NOTE**

Note that the ARSC field in the DMA_X_ARTC_SC register located in the address range 00530..005DF provides functionality equivalent to the ARSC field in the

DMAn_ARD_TC register located in the address range 00260..002EF. Writing to the ARSC bit field in either register affects the ARSC bit field in the other register.

### 13.43.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | ARSC | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ARSC | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.43.4 Fields

| Field | Function |
|---|---|
| 31-20<br>— | Reserved. |
| 19-0<br>ARSC | AXI Read Sample Count. This field is incremented after each read timing check. AXI read timing checks are suspended when ARSC=FFFFFh. This field is writeable only when ARTCE=0. |

## 13.44 DMA_X Read Timing Check Latency Register (DMA_X_ARTC_LAT)

### 13.44.1 Offset

| Register | Offset |
|---|---|
| DMA_X_ARTC_LAT | 53Ch |

### 13.44.2 Function

While AXI Read Timing Checks are enabled and not suspended, this register maintains a running total of AXI read latencies.

**NOTE**

Note that the DMA_X_ARTC_LAT register located in the address range 0053C..005DF is identical to the DMAn_SARL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

### 13.44.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SARL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SARL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.44.4 Fields

| Field | Function |
|---|---|
| 31-0<br>SARL | Sum of the AXI Read Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Read Latencies (SARL) in DMAn_SARL. This field is writeable only when ARTCE=0. |

## 13.45 DMA_X AXI Write Timing Check Control Register (DMA_X_AWTC_CTL)

### 13.45.1 Offset

| Register | Offset |
|---|---|
| DMA_X_AWTC_CTL | 540h |

### 13.45.2 Function

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the beginning of the corresponding write data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is read

After the DMA_X AXI Write Latency Register or DMA_X Write Timing Check Latency Register is read, the sample count, late count, and latency total are cleared and write timing checks resume with the next AXI write.

**NOTE**

Note that the DMA_X_AWTC_CTL register located in the address range 00540..005DF provides functionality similar to the DMAn_AWR_TC register located in the address range 00270..002EF. Some of the fields are aliased, i.e. writing to these fields in either register affects the corresponding fields in the other register. But note that some fields in the DMAn_AWR_TC register have been rearranged in the DMA_X_AWTC_CTL register or moved to the new DMA_X_AWTC_LC register or the DMA_X_AWTC_SC register.

## 13.45.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AWTCE | AWCT | AWTT | Reserved | AWL | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | AWT | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.45.4 Fields

| Field | Function |
|---|---|
| 31<br>AWTCE | AXI Write Timing Check Enable. When AWTCE=0, AWL, AWLC, and AWSC in DMA_X_AWTC and SAWL in DMAn_AWR_LAT and AWL in DMA_X_AWTC_CTL, AWLC in DMA0_AWTC_LC, AWSC in DMA0AWTC_SC and SARL in DMA0AWL_LAT are writeable. When AWTCE=1, AXI Write timing checks are enabled and these fields are Write-only.<br>**NOTE:** Note that writing AWTCE in either DMAn_AWR_TC or DMA_X_AWTC_CTL has the same effect. |
| 30<br>AWCT | AXI Write Counter Test. When AWCT=1, AWLC and AWSC in DMA_X_AWTC_CTL, AWLC in DMA_X_AWTC_LC, ARSC in DMA_X_WRTC_SC, and SAWL in DMAn_AWR_LAT and DMA_X_ARTC_LC, are not cleared when timing checks are enabled and when timing checks resume after reading DMAn_AWR_TC and DMAn_AWR_LAT or DMA_X_AWTC_LC, DMA_X_AWTC_SC and DMA_X_AWTC_LAT. This bit is used only for manufacturing test. It allows the counters to be initialized to non-zero values for the start of timing checks. This shortens the counting range so that terminal count behavior can be tested. |
| 29<br>AWTT | AXI Write Timer Test. When AWTT=1, the 12-bit timer used for each timing measurement is initialized to FF0h instead of 000h. This bit is used only for manufacturing test. The timer counts the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding Write data transaction. The test bit shortens the number of cycles to reach the terminal value FFFh. The timer stops at the terminal value until the next timing check starts. Note that bit field AWTT in the DMA_X_AWTC_CTL register is aliased to bit field AWTT in the DMAn_AWR_TC register, i.e. writing to either AWTT bit field alters the AWTT value in the other register. |
| 28<br>— | Reserved |
| 27-16<br>AWL | AXI Write Limit. The AXI Write Timer measures latency by counting the number of AXI clock cycles from the AXI Write address transaction to the beginning of the corresponding write data transaction. If the latency equals or exceeds the AXI Write Limit, the write response is considered late and the AXI Write Late Count (AWLC) is incremented along with the AXI Write Sample Count (AWSC). The latency is added to the Sum of AXI Write Latencies (SAWL) in DMAn_AWR_LAT/DMA_X_ARTC_LAT. This field is writeable only when AWTCE=0. Note that bit field AWL in the DMA_X_AWTC_CTL register is aliased to bit field AWL in the DMAn_AWR_TC register, i.e. writing to either AWL bit field alters the AWL value in the other register. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 15-12<br>— | Reserved |
| 11-0<br>AWT | AXI Write Timer. The number of AXI clock cycles from the latest external AXI write address transaction initiated by this DMA to the beginning of the corresponding write data transaction. This field is writeable only when AWTCE=0. |

## 13.46 DMA_X AXI Write Timing Check Late Count Register (DMA_X_AWTC_LC)

### 13.46.1 Offset

| Register | Offset |
|---|---|
| DMA_X_AWTC_LC | 544h |

### 13.46.2 Function

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the beginning of the corresponding write data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the AWTC_LC register is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is Write

**NOTE**

Note that the DMA_X_AWTC_LC register provides functionality similar to the AXI Write Timing Late Check fields in the DMAn_AWR_TC register located in the address range 00270..002EF, but the fields have been rearranged. Usage of the DMAn_AWR_TC register is deprecated.

### 13.46.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | AWLC | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AWLC | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.46.4 Fields

| Field | Function |
|---|---|
| 31-20<br>— | Reserved |
| 19-0<br>AWLC | AXI Write Late Count. This field is incremented each time that the ART field exceeds the ARL field in the DMA_X AXI Write Timing Check Control Register. AXI Write timing checks are suspended when AWLC=FFFFFh. Note that this field is an alias of the AWLC field in the DMAn_AWR_TC Register. Reading or writing the AWLC field in either the 00200 address block or the 00500 address block will yield identical results, and the value written can be read from the other register. When DMAn_AWR_TC/ DMA_X_AWTC_TC[AWTCE]=0, the AWLC bit field in DMAn_AWR_TC and the DMA_X_AWTC_LC register are writeable. When AWTCE=1, AXI write timing checks are enabled and the AWLC bit fields are read-only. |

## 13.47 DMA_X AXI Write Timing Check Sample Count Register (DMA_X_AWTC_SC)

### 13.47.1 Offset

| Register | Offset |
|---|---|
| DMA_X_AWTC_SC | 548h |

### 13.47.2 Function

When AXI Write Timing Checks are enabled, the DMA measures the latencies of selected AXI write transactions. A timer measures the latency by counting the number of AXI clock cycles from the write address transaction to the beginning of the corresponding write data transaction. The sample count is incremented and, if the latency equals or exceeds the programmed limit, the late count is incremented. The latency value is added to the running total of latencies. After completion of each timing check, the process is repeated for the next AXI write. Timing checks are suspended when:

- the AXI write sample count value reaches FFFFFh, or
- the AXI write latency total reaches FFFFFFFFh, or
- the AXI Write Timing Check Register is read

After the AXI Write Latency Register is read, the sample count, late count, and latency total are cleared and write timing checks resume with the next AXI write.

**NOTE**

Note that the AWSC field in the DMA_X_AWTC_SC register located in the address range 00530..005DF provides functionality equivalent to the AWSC field in the DMAn_AWR_TC register located in the address range 00260..002EF. Writing to the AWSC bit field in either register affects the AWSC bit field in the other register.

### 13.47.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | AWSC | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | AWSC | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.47.4 Fields

| Field | Function |
|---|---|
| 31-20<br>— | Reserved. |
| 19-0<br>AWSC | AXI Write Sample Count. This field is incremented after each write timing check. AXI write timing checks are suspended when AWSC=FFFFFh. This field is writeable only when AWTCE=0. |

## 13.48 DMA_X Write Timing Check Latency Register (DMA_X_AWTC_LAT)

### 13.48.1 Offset

| Register | Offset |
|---|---|
| DMA_X_AWTC_LAT | 54Ch |

### 13.48.2 Function

While AXI Write Timing Checks are enabled and not suspended, this register maintains a running total of AXI write latencies.

**NOTE**

Note that the DMA_X_AWTC_LAT register located in the address range 0053C..005DF is identical to the DMAn_AWL register located in the address range 00260..002EF. The register has simply been given two different addresses in order to consolidate legacy registers and new registers into two different continuous address ranges. Some registers in the 00500 address range have been reorganized to facilitate operation in both big-endian and little-endian SoCs.

### 13.48.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SAWL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SAWL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.48.4 Fields

| Field | Function |
|---|---|
| 31-0<br>SAWL | Sum of the AXI Write Latencies. After each AXI read timing check, the latency is added to the Sum of AXI Write Latencies (SAWL) in DMAn_AWL. This field is writeable only when AWTCE=0. |

## 13.49 RNG TRNG Miscellaneous Control Register (RTMCTL)

### 13.49.1 Offset

| Register | Offset |
|---|---|
| RTMCTL | 600h |

### 13.49.2 Function

These registers are intended to be used when testing the RNG. They would not be used during normal operation. During normal operation the RNG is configured and data is obtained from the RNG via Job Descriptors.

The RNG TRNG Miscellaneous Control Register is a read/write register used to control the RNG's True Random Number Generator (TRNG) access, operation and test.

**NOTE**

Note that in many cases two RNG registers share the same address, and a particular register at the shared address is selected based upon the value in the PRGM field of the RTMCTL register.

## 13.49.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | PRGM |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | TSTOP_OK | ERR | TST_OUT | ENT_VAL | FCT_VAL | FCT_FAIL | FORCE_SYSCLK | | TRNG_ACC | CLK_OUT_EN | OSC_DIV | | SAMP_MODE | |
| W | | | | W1C | | | | | | RST_DEF | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

## 13.49.4 Fields

| Field | Function |
|---|---|
| 31-17<br>— | Reserved. |
| 16<br>PRGM | Programming Mode Select. When this bit is 1, the TRNG is in Program Mode, otherwise it is in Run Mode. No Entropy value will be generated while the TRNG is in Program Mode. Note that different RNG registers are accessible at the same address depending on whether PRGM is set to 1 or 0. This is noted in the RNG register descriptions. |
| 15-14<br>— | Reserved. |
| 13<br>TSTOP_OK | TRNG_OK_TO_STOP. Software should check that this bit is a 1 before transitioning SEC to low power mode (SEC clock stopped). SEC turns on the TRNG free-running ring oscillator whenever new entropy is being generated and turns off the ring oscillator when entropy generation is complete. If the SEC clock is stopped while the TRNG ring oscillator is running, the oscillator will continue running even though the SEC clock is stopped. TSTOP_OK is asserted when the TRNG ring oscillator is not running. and therefore it is OK to stop the SEC clock. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 12<br>ERR | Read: Error status. 1 = error detected. 0 = no error.<br>Write: Write 1 to clear errors. Writing 0 has no effect. |
| 11<br>TST_OUT | Read only: Test point inside ring oscillator. |
| 10<br>ENT_VAL | Read only: Entropy Valid. Will assert only if TRNG ACC bit is set, and then after an entropy value is generated. Will be cleared when RTENT15 is read. (RTENT0 through RTENT14 should be read before reading RTENT15). |
| 9<br>FCT_VAL | Read only: Frequency Count Valid. Indicates that a valid frequency count may be read from RTFRQCNT. |
| 8<br>FCT_FAIL | Read only: Frequency Count Fail. The frequency counter has detected a failure. This may be due to improper programming of the RTFRQMAX and/or RTFRQMIN registers, or a hardware failure in the ring oscillator. This error may be cleared by writing a 1 to the ERR bit. |
| 7<br>FORCE_SYSCLK | Force System Clock. If set, the system clock is used to operate the TRNG, instead of the ring oscillator. This is for test use only, and indeterminate results may occur. This bit is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously to writing this bit. This bit is cleared by writing the RST_DEF bit to 1. |
| 6<br>RST_DEF | Reset Defaults. Writing a 1 to this bit clears various TRNG registers, and bits within registers, to their default state. This bit is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously to writing this bit. Reading this bit always produces a 0. |
| 5<br>TRNG_ACC | TRNG Access Mode. If this bit is set to 1, the TRNG will generate an Entropy value that can be read via the RTENT registers. The Entropy value may be read once the ENT VAL bit is asserted. This Entropy value will never be used by the RNG.<br>IMPORTANT: If this bit is set, no Entropy value can be generated for the RNG, which can prevent the RNG from generating data for the SEC system. |
| 4<br>CLK_OUT_EN | Clock Output Enable. If set, the ring oscillator output is gated to an output pad. If this bit is set and PRGM mode is selected, this allows external viewing of the ring oscillator. |
| 3-2<br>OSC_DIV | Oscillator Divide. Determines the amount of dividing done to the ring oscillator before it is used by the TRNG.<br>This field is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously to writing this field. This field is cleared to 00 by writing the RST_DEF bit to 1.<br>00b - use ring oscillator with no divide<br>01b - use ring oscillator divided-by-2<br>10b - use ring oscillator divided-by-4<br>11b - use ring oscillator divided-by-8 |
| 1-0<br>SAMP_MODE | Sample Mode. Determines the method of sampling the ring oscillator while generating the Entropy value:<br>This field is writable only if PRGM bit is 1, or PRGM bit is being written to 1 simultaneously with writing this field. This field is cleared to 01 by writing the RST_DEF bit to 1.<br>00b - use Von Neumann data into both Entropy shifter and Statistical Checker<br>01b - use raw data into both Entropy shifter and Statistical Checker<br>10b - use Von Neumann data into Entropy shifter. Use raw data into Statistical Checker<br>11b - undefined/reserved. |

# 13.50 RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC)

## 13.50.1 Offset

| Register | Offset |
|---|---|
| RTSCMISC | 604h |

## 13.50.2 Function

The RNG TRNG Statistical Check Miscellaneous Register contains the Long Run Maximum Limit value and the Retry Count value. This register is accessible only when the RTMCTL[PRGM] bit is 1, otherwise this register will read zeroes, and cannot be written.

**NOTE**

Reset occurs at POR, and when RTMCTL[RST_DEF] is written to 1.

## 13.50.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | RTY_CNT | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | LRUN_MAX | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

## 13.50.4 Fields

| Field | Function |
|---|---|
| 31-20 | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 19-16<br>RTY_CNT | RETRY COUNT. If a statistical check fails during the TRNG Entropy Generation, the RTY_CNT value indicates the number of times a retry should occur before generating an error. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 1h by writing the RTMCTL[RST_DEF] bit to 1. |
| 15-8<br>— | Reserved. |
| 7-0<br>LRUN_MAX | LONG RUN MAX LIMIT. This value is the largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 34 by writing the RTMCTL[RST_DEF] bit to 1. |

# 13.51 RNG TRNG Poker Range Register (RTPKRRNG)

## 13.51.1 Offset

| Register | Offset |
|---|---|
| RTPKRRNG | 608h |

## 13.51.2 Function

The RNG TRNG Poker Range Register defines the difference between the TRNG Poker Maximum Limit and the minimum limit. These limits are used during the TRNG Statistical Check Poker Test.

## 13.51.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PKR_RNG | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

### 13.51.4 Fields

| Field | Function |
|---|---|
| 31-16<br>— | Reserved. Always 0. |
| 15-0<br>PKR_RNG | Poker Range. During the TRNG Statistical Checks, a "Poker Test" is run which requires a maximum and minimum limit. The maximum is programmed in the RTPKRMAX[PKR_MAX] register, and the minimum is derived by subtracting the PKR_RNG value from the programmed maximum value. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 09A3h (decimal 2467) by writing the RTMCTL[RST_DEF] bit to 1. Note that the minimum allowable Poker result is PKR_MAX - PKR_RNG + 1. |

## 13.52 RNG TRNG Poker Square Calculation Result Register (RTPKRSQ)

### 13.52.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTPKRSQ | 60Ch | Accessible at this address when RTMCTL[PRGM] = 0] |

### 13.52.2 Function

The RNG TRNG Poker Square Calculation Result Register is a read-only register used to read the result of the TRNG Statistical Check Poker Test's Square Calculation. This test starts with the RTPKRMAX value and decreases towards a final result, which is read here. For the Poker Test to pass, this final result must be less than the programmed RTPKRRNG value. Note that this offset (060Ch) is used as RTPKRMAX if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTPKRSQ readback register, as described here.

### 13.52.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | PKR_SQ | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_SQ | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.52.4 Fields

| Field | Function |
|---|---|
| 31-24<br>— | Reserved. Always 0. |
| 23-0<br>PKR_SQ | Poker Square Calculation Result. During the TRNG Statistical Checks, a "Poker Test" is run which starts with the value RTPKRMAX[PKR_MAX]. This value decreases according to a "sum of squares" algorithm, and must remain greater than zero, but less than the RTPKRRNG[PKR_RNG] limit. The resulting value may be read through this register, if RTMCTL[PRGM] bit is 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Poker Test Maximum Limit in register RTPKRMAX, as defined in the previous section. |

## 13.53 RNG TRNG Poker Maximum Limit Register (RTPK RMAX)

### 13.53.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTPKRMAX | 60Ch | Accessible at this address when RTMCTL[PRGM] = 1] |

### 13.53.2 Function

The RNG TRNG Poker Maximum Limit Register defines Maximum Limit allowable during the TRNG Statistical Check Poker Test. Note that this offset (060Ch) is used as RTPKRMAX only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTPKRSQ readback register.

### 13.53.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | PKR_MAX | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PKR_MAX | | | | | | | | | | | | | | | |
| Reset | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

### 13.53.4 Fields

| Field | Function |
|---|---|
| 31-24<br>— | Reserved. Always 0. |
| 23-0<br>PKR_MAX | Poker Maximum Limit. During the TRNG Statistical Checks, a "Poker Test" is run which requires a maximum and minimum limit. The maximum allowable result is programmed in the RTPKRMAX[PKR_MAX] register. This field is writable only if RTMCTL[PRGM] bit is 1. This register is cleared to 006920h (decimal 26912) by writing the RTMCTL[RST_DEF] bit to 1. Note that the RTPKRMAX and RTPKRRNG registers combined are used to define the minimum allowable Poker result, which is PKR_MAX - PKR_RNG + 1. Note that if RTMCTL[PRGM] bit is 0, this register address is used to read the Poker Test Square Calculation result in register RTPKRSQ, as defined in the following section. |

## 13.54 RNG TRNG Seed Control Register (RTSDCTL)

## 13.54.1 Offset

| Register | Offset |
|---|---|
| RTSDCTL | 610h |

## 13.54.2 Function

The RNG TRNG Seed Control Register contains two fields. One field defines the length (in system clocks) of each Entropy sample (ENT_DLY), and the other field indicates the number of samples that will be taken during each TRNG Entropy generation (SAMP_SIZE).

## 13.54.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ENT_DLY | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SAMP_SIZE | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

## 13.54.4 Fields

| Field | Function |
|---|---|
| 31-16<br>ENT_DLY | Entropy Delay. Defines the length (in system clocks) of each Entropy sample taken. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 00C80h (decimal 3200) by writing the RTMCTL[RST_DEF] bit to 1. |
| 15-0<br>SAMP_SIZE | Sample Size. Defines the total number of Entropy samples that will be taken during Entropy generation. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 09C4h (decimal 2500) by writing the RTMCTL[RST_DEF] bit to 1. |

# 13.55 RNG TRNG Total Samples Register (RTTOTSAM)

## 13.55.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTTOTSAM | 614h | Accessible at this address when RTMCTL[PRGM] = 0] |

## 13.55.2 Function

The RNG TRNG Total Samples Register is a read-only register used to read the total number of samples taken during Entropy generation. It is used to give an indication of how often a sample is actually used during Von Neumann sampling. Note that this offset (0614h) is used as RTSBLIM if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTTOTSAM readback register, as described here.

## 13.55.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | TOT_SAM | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | TOT_SAM | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.55.4 Fields

| Field | Function |
|---|---|
| 31-20<br>— | Reserved. Always 0. |
| 19-0<br>TOT_SAM | Total Samples. During Entropy generation, the total number of raw samples is counted. This count is useful in determining how often a sample is used during Von Neumann sampling. The count may be read through this register, if RTMCTL[PRGM] bit is 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Sparse Bit Limit in register RTSBLIM, as defined in the previous section. |

# 13.56 RNG TRNG Sparse Bit Limit Register (RTSBLIM)

## 13.56.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSBLIM | 614h | Accessible at this address when RTMCTL[PRGM] = 1] |

## 13.56.2 Function

The RNG TRNG Sparse Bit Limit Register is used when Von Neumann sampling is selected during Entropy Generation. It defines the maximum number of consecutive Von Neumann samples which may be discarded before an error is generated. Note that this address (0614h) is used as RTSBLIM only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTTOTSAM readback register.

## 13.56.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | SB_LIM | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

## 13.56.4 Fields

| Field | Function |
|---|---|
| 31-10 | Reserved. Always 0. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 9-0<br>SB_LIM | Sparse Bit Limit. During Von Neumann sampling (if enabled by RTMCTL[SAMP_MODE], samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy. The Sparse Bit Limit defines the maximum number of consecutive samples that may be discarded before an error is generated. This field is writable only if RTMCTL[PRGM] bit is 1. This register is cleared to 03hF by writing the RTMCTL[RST_DEF] bit to 1. Note that if RTMCTL[PRGM] bit is 0, this register address is used to read the Total Samples count in register RTTOTSAM, as defined in the following section. |

## 13.57 RNG TRNG Frequency Count Minimum Limit Register (RTFRQMIN)

### 13.57.1 Offset

| Register | Offset |
|---|---|
| RTFRQMIN | 618h |

### 13.57.2 Function

The RNG TRNG Frequency Count Minimum Limit Register defines the minimum allowable count taken by the Entropy sample counter during each Entropy sample. During any sample period, if the count is less than this programmed minimum, a Frequency Count Fail is flagged in RTMCTL[FCT_FAIL] and an error is generated.

### 13.57.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | FRQ_MIN | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | FRQ_MIN | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

### 13.57.4 Fields

| Field | Function |
|---|---|
| 31-22<br>— | Reserved. Always 0. |
| 21-0<br>FRQ_MIN | Frequency Count Minimum Limit. Defines the minimum allowable count taken during each entropy sample. This field is writable only if RTMCTL[PRGM] bit is 1. This field will read zeroes if RTMCTL[PRGM] = 0. This field is cleared to 000190h by writing the RTMCTL[RST_DEF] bit to 1. |

## 13.58 RNG TRNG Frequency Count Register (RTFRQCNT)

### 13.58.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTFRQCNT | 61Ch | RNG TRNG Frequency Count accessible at this address when RTMCTL[PRGM] = 0] |

### 13.58.2 Function

The RNG TRNG Frequency Count Register is a read-only register used to read the frequency counter within the TRNG entropy generator. It will read all zeroes unless RTMCTL[TRNG_ACC] = 1. Note that this offset (061Ch) is used as RTFRQMAX if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTFRQCNT readback register, as described here.

### 13.58.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | FRQ_CNT | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | FRQ_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.58.4 Fields

| Field | Function |
|---|---|
| 31-22<br>— | Reserved. Always 0. |
| 21-0<br>FRQ_CNT | Frequency Count. If RTMCTL[TRNG_ACC] = 1, reads a sample frequency count taken during entropy generation. Requires RTMCTL[PRGM] = 0. The value read from FRQ_CNT is valid only if RTMCTL[FCT_VAL] = 1. |

## 13.59 RNG TRNG Frequency Count Maximum Limit Register (RTFRQMAX)

### 13.59.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTFRQMAX | 61Ch | Accessible at this address when RTMCTL[PRGM] = 1] |

### 13.59.2 Function

The RNG TRNG Frequency Count Maximum Limit Register defines the maximum allowable count taken by the Entropy sample counter during each Entropy sample. During any sample period, if the count is greater than this programmed maximum, a Frequency Count Fail is flagged in RTMCTL[FCT_FAIL] and an error is generated. Note that this address (061C) is used as RTFRQMAX only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTFRQCNT readback register.

### 13.59.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | FRQ_MAX | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | FRQ_MAX | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.59.4 Fields

| Field | Function |
|---|---|
| 31-22<br>— | Reserved. Always 0. |
| 21-0<br>FRQ_MAX | Frequency Counter Maximum Limit. Defines the maximum allowable count taken during each entropy sample. This field is writable only if RTMCTL[PRGM] bit is 1. This register is cleared to 00190h by writing the RTMCTL[RST_DEF] bit to 1. Note that if RTMCTL[PRGM] bit is 0, this register address is used to read the Frequency Count result in register RTFRQCNT, as defined in the following section. |

## 13.60 RNG TRNG Statistical Check Monobit Count Register (RTSCMC)

## 13.60.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCMC | 620h | Accessible at this address when RTMCTL[PRGM] = 0] |

## 13.60.2 Function

The RNG TRNG Statistical Check Monobit Count Register is a read-only register used to read the final monobit count after entropy generation. This counter starts with the value in RTSCML[MONO_MAX], and is decremented each time a one is sampled. Note that this offset (0620h) is used as RTSCML if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCMC readback register, as described here.

## 13.60.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MONO_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.60.4 Fields

| Field | Function |
|---|---|
| 31-16<br>— | Reserved. Always 0. |
| 15-0<br>MONO_CNT | Monobit Count. Reads the final Monobit count after entropy generation. Requires RTMCTL[PRGM] = 0. Note that if RTMCTL[PRGM] bit is 1, this register address is used to access the Statistical Check Monobit Limit in register RTSCML, as defined in the previous section. |

# 13.61 RNG TRNG Statistical Check Monobit Limit Register (RTSCML)

## 13.61.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCML | 620h | Accessible at this address when RTMCTL[PRGM] = 1] |

## 13.61.2 Function

The RNG TRNG Statistical Check Monobit Limit Register defines the allowable maximum and minimum number of ones/zero detected during entropy generation. To pass the test, the number of ones/zeroes generated must be less than the programmed maximum value, and the number of ones/zeroes generated must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this offset (0620h) is used as RTSCML only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCMC readback register.

## 13.61.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MONO_RNG | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MONO_MAX | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |

### 13.61.4 Fields

| Field | Function |
|---|---|
| 31-16<br>MONO_RNG | Monobit Range. The number of ones/zeroes detected during entropy generation must be greater than MONO_MAX - MONO_RNG, else a retry or error will occur. This register is cleared to 000112h (decimal 274) by writing the RTMCTL[RST_DEF] bit to 1. |
| 15-0<br>MONO_MAX | Monobit Maximum Limit. Defines the maximum allowable count taken during entropy generation. The number of ones/zeroes detected during entropy generation must be less than MONO_MAX, else a retry or error will occur. This register is cleared to 00056Bh (decimal 1387) by writing the RTMCTL[RST_DEF] bit to 1. |

## 13.62 RNG TRNG Statistical Check Run Length 1 Count Register (RTSCR1C)

### 13.62.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR1C | 624h | Accessible at this address when RTMCTL[PRGM] = 0] |

### 13.62.2 Function

The RNG TRNG Statistical Check Run Length 1 Counters Register is a read-only register used to read the final Run Length 1 counts after entropy generation. These counters start with the value in RTSCR1L[RUN1_MAX]. The R1_1_COUNT decrements each time a single one is sampled (preceded by a zero and followed by a zero). The R1_0_COUNT decrements each time a single zero is sampled (preceded by a one and followed by a one). Note that this offset (0624h) is used as RTSCR1L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR1C readback register, as described here.

## 13.62.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | R1_1_COUNT | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | R1_0_COUNT | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.62.4 Fields

| Field | Function |
|---|---|
| 31<br>— | Reserved. Always 0. |
| 30-16<br>R1_1_COUNT | Runs of One, Length 1 Count. Reads the final Runs of Ones, length 1 count after entropy generation. Requires RTMCTL[PRGM] = 0. |
| 15<br>— | Reserved. Always 0. |
| 14-0<br>R1_0_COUNT | Runs of Zero, Length 1 Count. Reads the final Runs of Zeroes, length 1 count after entropy generation. Requires RTMCTL[PRGM] = 0. |

# 13.63 RNG TRNG Statistical Check Run Length 1 Limit Register (RTSCR1L)

## 13.63.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR1L | 624h | Accessible at this address when RTMCTL[PRGM] = 1] |

## 13.63.2 Function

The RNG TRNG Statistical Check Run Length 1 Limit Register defines the allowable maximum and minimum number of runs of length 1 detected during entropy generation. To pass the test, the number of runs of length 1 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 1 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0624h) is used as RTSCR1L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR1C readback register.

## 13.63.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | RUN1_RNG | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | RUN1_MAX | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |

### 13.63.4 Fields

| Field | Function |
|---|---|
| 31<br>— | Reserved. Always 0. |
| 30-16<br>RUN1_RNG | Run Length 1 Range. The number of runs of length 1 (for both 0 and 1) detected during entropy generation must be greater than RUN1_MAX - RUN1_RNG, else a retry or error will occur. This register is cleared to 0102h (decimal 258) by writing the RTMCTL[RST_DEF] bit to 1. |
| 15<br>— | Reserved. Always 0. |
| 14-0<br>RUN1_MAX | Run Length 1 Maximum Limit. Defines the maximum allowable runs of length 1 (for both 0 and 1) detected during entropy generation. The number of runs of length 1 detected during entropy generation must be less than RUN1_MAX, else a retry or error will occur. This register is cleared to 01E5h (decimal 485) by writing the RTMCTL[RST_DEF] bit to 1. |

## 13.64 RNG TRNG Statistical Check Run Length 2 Count Register (RTSCR2C)

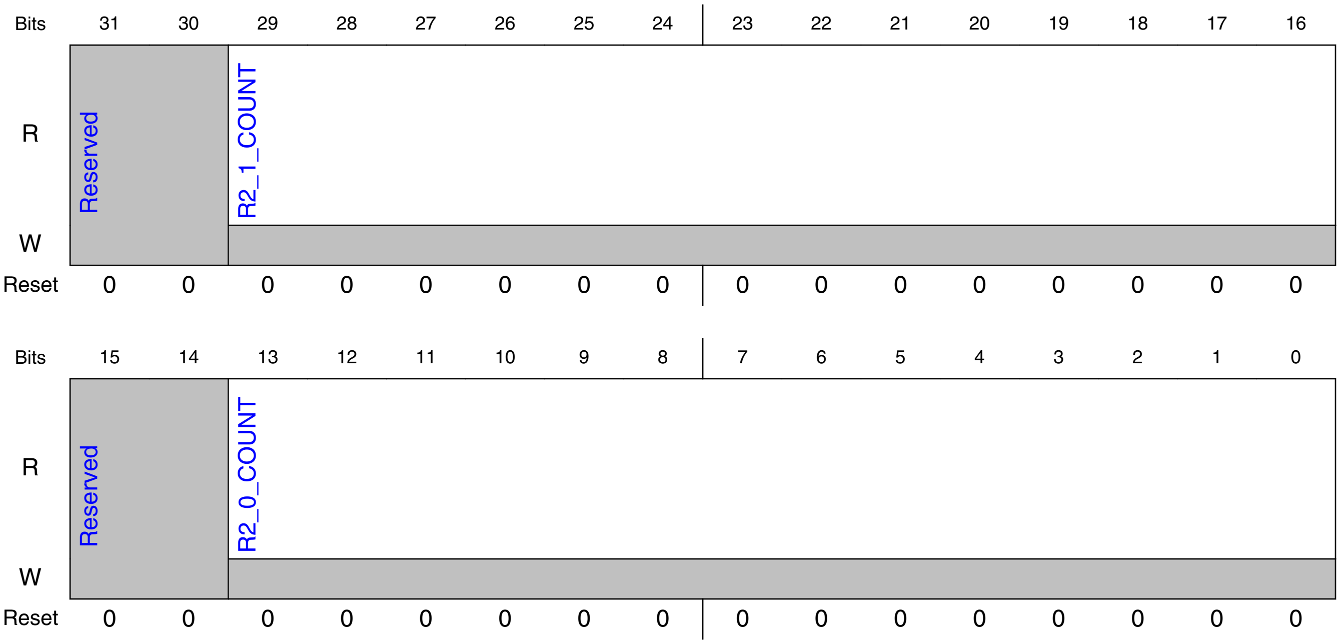
### 13.64.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR2C | 628h | Accessible at this address when RTMCTL[PRGM] = 0] |

### 13.64.2 Function

The RNG TRNG Statistical Check Run Length 2 Counters Register is a read-only register used to read the final Run Length 2 counts after entropy generation. These counters start with the value in RTSCR2L[RUN2_MAX]. The R2_1_COUNT decrements each time two consecutive ones are sampled (preceded by a zero and followed by a zero). The R2_0_COUNT decrements each time two consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0628h) is used as RTSCR2L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR2C readback register, as described here.

## 13.64.3 Diagram



| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | R2_1_COUNT | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | R2_0_COUNT | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.64.4 Fields

| Field | Function |
|---|---|
| 31-30<br>— | Reserved. Always 0. |
| 29-16<br>R2_1_COUNT | Runs of One, Length 2 Count. Reads the final Runs of Ones, length 2 count after entropy generation. Requires RTMCTL[PRGM] = 0. |
| 15-14<br>— | Reserved. Always 0. |
| 13-0<br>R2_0_COUNT | Runs of Zero, Length 2 Count. Reads the final Runs of Zeroes, length 2 count after entropy generation. Requires RTMCTL[PRGM] = 0. |

# 13.65 RNG TRNG Statistical Check Run Length 2 Limit Register (RTSCR2L)

## 13.65.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR2L | 628h | Accessible at this address when RTMCTL[PRGM] = 1] |

## 13.65.2 Function

The RNG TRNG Statistical Check Run Length 2 Limit Register defines the allowable maximum and minimum number of runs of length 2 detected during entropy generation. To pass the test, the number of runs of length 2 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 2 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0628h) is used as RTSCR2L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR2C readback register.

## 13.65.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | RUN2_RNG | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | RUN2_MAX | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |

### 13.65.4 Fields

| Field | Function |
|---|---|
| 31-30<br>— | Reserved. Always 0. |
| 29-16<br>RUN2_RNG | Run Length 2 Range. The number of runs of length 2 (for both 0 and 1) detected during entropy generation must be greater than RUN2_MAX - RUN2_RNG, else a retry or error will occur. This register is cleared to 007Ah (decimal 122) by writing the RTMCTL[RST_DEF] bit to 1. |
| 15-14<br>— | Reserved. Always 0. |
| 13-0<br>RUN2_MAX | Run Length 2 Maximum Limit. Defines the maximum allowable runs of length 2 (for both 0 and 1) detected during entropy generation. The number of runs of length 2 detected during entropy generation must be less than RUN2_MAX, else a retry or error will occur. This register is cleared to 00DCh (decimal 220) by writing the RTMCTL[RST_DEF] bit to 1. |

## 13.66 RNG TRNG Statistical Check Run Length 3 Limit Register (RTSCR3L)

### 13.66.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR3L | 62Ch | Accessible at this address when RTMCTL[PRGM] = 1] |

### 13.66.2 Function

The RNG TRNG Statistical Check Run Length 3 Limit Register defines the allowable maximum and minimum number of runs of length 3 detected during entropy generation. To pass the test, the number of runs of length 3 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 3 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (062Ch) is used as RTSCR3L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR3C readback register.

### 13.66.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | RUN3_RNG | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | RUN3_MAX | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

### 13.66.4 Fields

| Field | Function |
|---|---|
| 31-29<br>— | Reserved. Always 0. |
| 28-16<br>RUN3_RNG | Run Length 3 Range. The number of runs of length 3 (for both 0 and 1) detected during entropy generation must be greater than RUN3_MAX - RUN3_RNG, else a retry or error will occur. This register is cleared to 0058h (decimal 88) by writing the RTMCTL[RST_DEF] bit to 1. |
| 15-13<br>— | Reserved. Always 0. |
| 12-0<br>RUN3_MAX | Run Length 3 Maximum Limit. Defines the maximum allowable runs of length 3 (for both 0 and 1) detected during entropy generation. The number of runs of length 3 detected during entropy generation must be less than RUN3_MAX, else a retry or error will occur. This register is cleared to 007Dh (decimal 125) by writing the RTMCTL[RST_DEF] bit to 1. |

## 13.67 RNG TRNG Statistical Check Run Length 3 Count Register (RTSCR3C)

### 13.67.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR3C | 62Ch | Accessible at this address when RTMCTL[PRGM] = 0] |

## 13.67.2 Function

The RNG TRNG Statistical Check Run Length 3 Counters Register is a read-only register used to read the final Run Length 3 counts after entropy generation. These counters start with the value in RTSCR3L[RUN3_MAX]. The R3_1_COUNT decrements each time three consecutive ones are sampled (preceded by a zero and followed by a zero). The R3_0_COUNT decrements each time three consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (062Ch) is used as RTSCR3L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR3C readback register, as described here.

## 13.67.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | R3_1_COUNT | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | R3_0_COUNT | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.67.4 Fields

| Field | Function |
|---|---|
| 31-29<br>— | Reserved. Always 0. |
| 28-16<br>R3_1_COUNT | Runs of Ones, Length 3 Count. Reads the final Runs of Ones, length 3 count after entropy generation. Requires RTMCTL[PRGM] = 0. |
| 15-13<br>— | Reserved. Always 0. |
| 12-0<br>R3_0_COUNT | Runs of Zeroes, Length 3 Count. Reads the final Runs of Zeroes, length 3 count after entropy generation. Requires RTMCTL[PRGM] = 0. |

# 13.68 RNG TRNG Statistical Check Run Length 4 Limit Register (RTSCR4L)

## 13.68.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR4L | 630h | Accessible at this address when RTMCTL[PRGM] = 1] |

## 13.68.2 Function

The RNG TRNG Statistical Check Run Length 4 Limit Register defines the allowable maximum and minimum number of runs of length 4 detected during entropy generation. To pass the test, the number of runs of length 4 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 4 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0630h) is used as RTSCR4L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR4C readback register.

## 13.68.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | RUN4_RNG | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | RUN4_MAX | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |

### 13.68.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved. Always 0. |
| 27-16<br>RUN4_RNG | Run Length 4 Range. The number of runs of length 4 (for both 0 and 1) detected during entropy generation must be greater than RUN4_MAX - RUN4_RNG, else a retry or error will occur. This register is cleared to 0040h (decimal 64) by writing the RTMCTL[RST_DEF] bit to 1. |
| 15-12<br>— | Reserved. Always 0. |
| 11-0<br>RUN4_MAX | Run Length 4 Maximum Limit. Defines the maximum allowable runs of length 4 (for both 0 and 1) detected during entropy generation. The number of runs of length 4 detected during entropy generation must be less than RUN4_MAX, else a retry or error will occur. This register is cleared to 004Bh (decimal 75) by writing the RTMCTL[RST_DEF] bit to 1. |

## 13.69 RNG TRNG Statistical Check Run Length 4 Count Register (RTSCR4C)

### 13.69.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR4C | 630h | Accessible at this address when RTMCTL[PRGM] = 0] |

### 13.69.2 Function

The RNG TRNG Statistical Check Run Length 4 Counters Register is a read-only register used to read the final Run Length 4 counts after entropy generation. These counters start with the value in RTSCR4L[RUN4_MAX]. The R4_1_COUNT decrements each time four consecutive ones are sampled (preceded by a zero and followed by a zero). The R4_0_COUNT decrements each time four consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0630h) is used as RTSCR4L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR4C readback register, as described here.

### 13.69.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | R4_1_COUNT | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | R4_0_COUNT | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.69.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved. Always 0. |
| 27-16<br>R4_1_COUNT | Runs of One, Length 4 Count. Reads the final Runs of Ones, length 4 count after entropy generation. Requires RTMCTL[PRGM] = 0. |
| 15-12<br>— | Reserved. Always 0. |
| 11-0<br>R4_0_COUNT | Runs of Zero, Length 4 Count. Reads the final Runs of Ones, length 4 count after entropy generation. Requires RTMCTL[PRGM] = 0. |

## 13.70 RNG TRNG Statistical Check Run Length 5 Count Register (RTSCR5C)

### 13.70.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR5C | 634h | Accessible at this address when RTMCTL[PRGM] = 0] |

## 13.70.2 Function

The RNG TRNG Statistical Check Run Length 5 Counters Register is a read-only register used to read the final Run Length 5 counts after entropy generation. These counters start with the value in RTSCR5L[RUN5_MAX]. The R5_1_COUNT decrements each time five consecutive ones are sampled (preceded by a zero and followed by a zero). The R5_0_COUNT decrements each time five consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0634h) is used as RTSCR5L if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR5C readback register, as described here.

## 13.70.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | R5_1_COUNT | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | R5_0_COUNT | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.70.4 Fields

| Field | Function |
|---|---|
| 31-27<br>— | Reserved. Always 0. |
| 26-16<br>R5_1_COUNT | Runs of One, Length 5 Count. Reads the final Runs of Ones, length 5 count after entropy generation. Requires RTMCTL[PRGM] = 0. |
| 15-11<br>— | Reserved. Always 0. |
| 10-0<br>R5_0_COUNT | Runs of Zero, Length 5 Count. Reads the final Runs of Ones, length 5 count after entropy generation. Requires RTMCTL[PRGM] = 0. |

# 13.71 RNG TRNG Statistical Check Run Length 5 Limit Register (RTSCR5L)

## 13.71.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR5L | 634h | Accessible at this address when RTMCTL[PRGM] = 1] |

## 13.71.2 Function

The RNG TRNG Statistical Check Run Length 5 Limit Register defines the allowable maximum and minimum number of runs of length 5 detected during entropy generation. To pass the test, the number of runs of length 5 (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 5 must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this address (0634h) is used as RTSCR5L only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this address is used as RTSCR5C readback register.

## 13.71.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | RUN5_RNG | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | RUN5_MAX | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

### 13.71.4 Fields

| Field | Function |
|---|---|
| 31-27<br>— | Reserved. Always 0. |
| 26-16<br>RUN5_RNG | Run Length 5 Range. The number of runs of length 5 (for both 0 and 1) detected during entropy generation must be greater than RUN5_MAX - RUN5_RNG, else a retry or error will occur. This register is cleared to 002Eh (decimal 46) by writing the RTMCTL[RST_DEF] bit to 1. |
| 15-11<br>— | Reserved. Always 0. |
| 10-0<br>RUN5_MAX | Run Length 5 Maximum Limit. Defines the maximum allowable runs of length 5 (for both 0 and 1) detected during entropy generation. The number of runs of length 5 detected during entropy generation must be less than RUN5_MAX, else a retry or error will occur. This register is cleared to 002Fh (decimal 47) by writing the RTMCTL[RST_DEF] bit to 1. |

## 13.72 RNG TRNG Statistical Check Run Length 6+ Limit Register (RTSCR6PL)

### 13.72.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR6PL | 638h | Accessible at this address when RTMCTL[PRGM] = 1] |

### 13.72.2 Function

The RNG TRNG Statistical Check Run Length 6+ Limit Register defines the allowable maximum and minimum number of runs of length 6 or more detected during entropy generation. To pass the test, the number of runs of length 6 or more (for samples of both 0 and 1) must be less than the programmed maximum value, and the number of runs of length 6 or more must be greater than (maximum - range). If this test fails, the Retry Counter in RTSCMISC will be decremented, and a retry will occur if the Retry Count has not reached zero. If the Retry Count has reached zero, an error will be generated. Note that this offset (0638h) is used as RTSCR6PL only if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR6PC readback register.

### 13.72.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | RUN6P_RNG | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | RUN6P_MAX | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

### 13.72.4 Fields

| Field | Function |
|---|---|
| 31-27<br>— | Reserved. Always 0. |
| 26-16<br>RUN6P_RNG | Run Length 6+ Range. The number of runs of length 6 or more (for both 0 and 1) detected during entropy generation must be greater than RUN6P_MAX - RUN6P_RNG, else a retry or error will occur. This register is cleared to 002Eh (decimal 46) by writing the RTMCTL[RST_DEF] bit to 1. |
| 15-11<br>— | Reserved. Always 0. |
| 10-0<br>RUN6P_MAX | Run Length 6+ Maximum Limit. Defines the maximum allowable runs of length 6 or more (for both 0 and 1) detected during entropy generation. The number of runs of length 6 or more detected during entropy generation must be less than RUN6P_MAX, else a retry or error will occur. This register is cleared to 002Fh (decimal 47) by writing the RTMCTL[RST_DEF] bit to 1. |

## 13.73 RNG TRNG Statistical Check Run Length 6+ Count Register (RTSCR6PC)

### 13.73.1 Offset

| Register | Offset | Description |
|---|---|---|
| RTSCR6PC | 638h | Accessible at this address when RTMCTL[PRGM] = 0] |

### 13.73.2 Function

The RNG TRNG Statistical Check Run Length 6+ Counters Register is a read-only register used to read the final Run Length 6+ counts after entropy generation. These counters start with the value in RTSCR6PL[RUN6P_MAX]. The R6P_1_COUNT decrements each time six or more consecutive ones are sampled (preceded by a zero and followed by a zero). The R6P_0_COUNT decrements each time six or more consecutive zeroes are sampled (preceded by a one and followed by a one). Note that this offset (0638h) is used as RTSCR6PL if RTMCTL[PRGM] is 1. If RTMCTL[PRGM] is 0, this offset is used as RTSCR6PC readback register, as described here.

### 13.73.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | R6P_1_COUNT | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | R6P_0_COUNT | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.73.4 Fields

| Field | Function |
|---|---|
| 31-27<br>— | Reserved. Always 0. |
| 26-16<br>R6P_1_COUNT | Runs of One, Length 6+ Count. Reads the final Runs of Ones, length 6+ count after entropy generation. Requires RTMCTL[PRGM] = 0. |
| 15-11<br>— | Reserved. Always 0. |
| 10-0<br>R6P_0_COUNT | Runs of Zero, Length 6+ Count. Reads the final Runs of Ones, length 6+ count after entropy generation. Requires RTMCTL[PRGM] = 0. |

## 13.74 RNG TRNG Status Register (RTSTATUS)

### 13.74.1 Offset

| Register | Offset |
|---|---|
| RTSTATUS | 63Ch |

### 13.74.2 Function

Various statistical tests are run as a normal part of the TRNG's entropy generation process. If the RNG TRNG Miscellaneous Control Register (RTMCTL) ERR field indicates an error, the least-significant 16 bits of the RTSTATUS register will indicate which test(s) have failed. The status of these bits will be valid when the TRNG has finished its entropy generation process. Software can determine when this occurs by polling the ENT_VAL bit in RTMCTL. If RTMCTL[ERR] indicates no error, then RTSTATUS register does not contain valid test status data.

Note that there is a very small probability that a statistical test will fail even though the TRNG is operating properly. If this happens the TRNG will automatically retry the entire entropy generation process, including running all the statistical tests. The value in RETRY_COUNT is decremented each time an entropy generation retry occurs. If a statistical check fails when the retry count is nonzero, a retry is initiated. But if a statistical check fails when the retry count is zero, an error is generated by the RNG. By default RETRY_COUNT is initialized to 1, but software can increase the retry count by writing to the RTY_CNT field in the RTSCMISC register (see RNG TRNG Statistical Check Miscellaneous Register (RTSCMISC)).

All 0s will be returned if this register address is read while the RNG is in Program Mode (see PRGM field in RTMCTL register (see RNG TRNG Miscellaneous Control Register (RTMCTL)). If this register is read while the RNG is in Run Mode the value returned will be formatted as follows.

## 13.74.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | RETRY_COUNT | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | FMBTF | FPTF | FLRTF | FSBTF | F6PBR1TF | F6PBR0TF | F5BR1TF | F5BR0TF | F4BR1TF | F4BR0TF | F3BR1TF | F3BR01TF | F2BR1TF | F2BR0TF | F1BR1TF | F1BR0TF |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.74.4 Fields

| Field | Function |
|---|---|
| 31-20<br>— | Reserved. Always 0. |
| 19-16<br>RETRY_COUNT | RETRY COUNT. This represents the current number of entropy generation retries left before a statistical text failure will cause the RNG to generate an error condition. |
| 15<br>FMBTF | Mono Bit Test Fail. If MBTF=1, the Mono Bit Test has failed. |
| 14<br>FPTF | Poker Test Fail. If PTF=1, the Poker Test has failed. |
| 13<br>FLRTF | Long Run Test Fail. If LRTF=1, the Long Run Test has failed. |
| 12<br>FSBTF | Sparse Bit Test Fail. If SBTF=1, the Sparse Bit Test has failed. |
| 11<br>F6PBR1TF | 6 Plus Bit Run, Sampling 1s, Test Fail. If 6PBR1TF=1, the 6 Plus Bit Run, Sampling 1s Test has failed. |
| 10<br>F6PBR0TF | 6 Plus Bit Run, Sampling 0s, Test Fail. If 6PBR0TF=1, the 6 Plus Bit Run, Sampling 0s Test has failed. |
| 9<br>F5BR1TF | 5-Bit Run, Sampling 1s, Test Fail. If 5BR1TF=1, the 5-Bit Run, Sampling 1s Test has failed. |
| 8<br>F5BR0TF | 5-Bit Run, Sampling 0s, Test Fail. If 5BR0TF=1, the 5-Bit Run, Sampling 0s Test has failed. |
| 7 | 4-Bit Run, Sampling 1s, Test Fail. If 4BR1TF=1, the 4-Bit Run, Sampling 1s Test has failed. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| F4BR1TF | |
| 6<br>F4BR0TF | 4-Bit Run, Sampling 0s, Test Fail. If 4BR0TF=1, the 4-Bit Run, Sampling 0s Test has failed. |
| 5<br>F3BR1TF | 3-Bit Run, Sampling 1s, Test Fail. If 3BR1TF=1, the 3-Bit Run, Sampling 1s Test has failed. |
| 4<br>F3BR01TF | 3-Bit Run, Sampling 0s, Test Fail. If 3BR0TF=1, the 3-Bit Run, Sampling 0s Test has failed. |
| 3<br>F2BR1TF | 2-Bit Run, Sampling 1s, Test Fail. If 2BR1TF=1, the 2-Bit Run, Sampling 1s Test has failed. |
| 2<br>F2BR0TF | 2-Bit Run, Sampling 0s, Test Fail. If 2BR0TF=1, the 2-Bit Run, Sampling 0s Test has failed. |
| 1<br>F1BR1TF | 1-Bit Run, Sampling 1s, Test Fail. If 1BR1TF=1, the 1-Bit Run, Sampling 1s Test has failed. |
| 0<br>F1BR0TF | 1-Bit Run, Sampling 0s, Test Fail. If 1BR0TF=1, the 1-Bit Run, Sampling 0s Test has failed. |

## 13.75 RNG TRNG Entropy Read Register (RTENT0 - RTEN T15)

### 13.75.1 Offset

For a = 0 to 15:

| Register | Offset |
|---|---|
| RTENTa | 640h + (a × 4h) |

### 13.75.2 Function

The RNG TRNG can be programmed to generate an entropy value that is readable via the SkyBlue bus. To do this, set the RTMCTL[TRNG_ACC] bit to 1. Once the entropy value has been generated, the RTMCTL[ENT_VAL] bit will be set to 1. At this point, RTENT0 through RTENT15 may be read to retrieve the 512-bit entropy value. Note that once RTENT15 is read, the entropy value will be cleared and a new value will begin generation, so it is important that RTENT15 be read last. Also note that the entropy value read from the RTENT0 - RTENT15 registers will never be used by the SEC for any purpose other than to be read via these registers. Any entropy value used for any security function cannot be read. These registers are readable only when RTMCTL[PRGM] = 0 (Run Mode), RTMCTL[TRNG_ACC] = 1 (TRNG access mode) and RTMCTL[ENT_VAL] = 1, otherwise zeroes will be read.

### 13.75.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ENT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ENT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.75.4 Fields

| Field | Function |
|---|---|
| 31-0<br>ENT | Entropy Value. Will be non-zero only if RTMCTL[PRGM] = 0 (Run Mode) and RTMCTL[ENT_VAL] = 1 (Entropy Valid). The most significant bits of the entropy are read from the lowest offset, and the least significant bits are read from the highest offset. Note that reading the highest offset also clears the entire entropy value, and starts a new entropy generation. |

## 13.76 RNG TRNG Statistical Check Poker Count 1 and 0 Register (RTPKRCNT10)

### 13.76.1 Offset

| Register | Offset |
|---|---|
| RTPKRCNT10 | 680h |

### 13.76.2 Function

The RNG TRNG Statistical Check Poker Count 1 and 0 Register is a read-only register used to read the final Poker test counts of 1h and 0h patterns. The Poker 0h Count increments each time a nibble of sample data is found to be 0h. The Poker 1h Count increments each time a nibble of sample data is found to be 1h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

### 13.76.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_1_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_0_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.76.4 Fields

| Field | Function |
|---|---|
| 31-16<br>PKR_1_CNT | Poker 1h Count. Total number of nibbles of sample data which were found to be 1h. Requires RTMCTL[PRGM] = 0. |
| 15-0<br>PKR_0_CNT | Poker 0h Count. Total number of nibbles of sample data which were found to be 0h. Requires RTMCTL[PRGM] = 0. |

## 13.77 RNG TRNG Statistical Check Poker Count 3 and 2 Register (RTPKRCNT32)

### 13.77.1 Offset

| Register | Offset |
|---|---|
| RTPKRCNT32 | 684h |

### 13.77.2 Function

The RNG TRNG Statistical Check Poker Count 3 and 2 Register is a read-only register used to read the final Poker test counts of 3h and 2h patterns. The Poker 2h Count increments each time a nibble of sample data is found to be 2h. The Poker 3h Count increments each time a nibble of sample data is found to be 3h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

### 13.77.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_3_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_2_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.77.4 Fields

| Field | Function |
|---|---|
| 31-16<br>PKR_3_CNT | Poker 3h Count. Total number of nibbles of sample data which were found to be 3h. Requires RTMCTL[PRGM] = 0. |
| 15-0<br>PKR_2_CNT | Poker 2h Count. Total number of nibbles of sample data which were found to be 2h. Requires RTMCTL[PRGM] = 0. |

# 13.78 RNG TRNG Statistical Check Poker Count 5 and 4 Register (RTPKRCNT54)

## 13.78.1 Offset

| Register | Offset |
|---|---|
| RTPKRCNT54 | 688h |

## 13.78.2 Function

The RNG TRNG Statistical Check Poker Count 5 and 4 Register is a read-only register used to read the final Poker test counts of 5h and 4h patterns. The Poker 4h Count increments each time a nibble of sample data is found to be 4h. The Poker 5h Count increments each time a nibble of sample data is found to be 5h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

## 13.78.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_5_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_4_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.78.4 Fields

| Field | Function |
|---|---|
| 31-16<br>PKR_5_CNT | Poker 5h Count. Total number of nibbles of sample data which were found to be 5h. Requires RTMCTL[PRGM] = 0. |
| 15-0<br>PKR_4_CNT | Poker 4h Count. Total number of nibbles of sample data which were found to be 4h. Requires RTMCTL[PRGM] = 0. |

# 13.79 RNG TRNG Statistical Check Poker Count 7 and 6 Register (RTPKRCNT76)

## 13.79.1 Offset

| Register | Offset |
|---|---|
| RTPKRCNT76 | 68Ch |

## 13.79.2 Function

The RNG TRNG Statistical Check Poker Count 7 and 6 Register is a read-only register used to read the final Poker test counts of 7h and 6h patterns. The Poker 6h Count increments each time a nibble of sample data is found to be 6h. The Poker 7h Count increments each time a nibble of sample data is found to be 7h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

## 13.79.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_7_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_6_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.79.4 Fields

| Field | Function |
|---|---|
| 31-16 | Poker 7h Count. Total number of nibbles of sample data which were found to be 7h. Requires RTMCTL[PRGM] = 0. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| PKR_7_CNT | |
| 15-0<br>PKR_6_CNT | Poker 6h Count. Total number of nibbles of sample data which were found to be 6h. Requires RTMCTL[PRGM] = 0. |

# 13.80 RNG TRNG Statistical Check Poker Count 9 and 8 Register (RTPKRCNT98)

## 13.80.1 Offset

| Register | Offset |
|---|---|
| RTPKRCNT98 | 690h |

## 13.80.2 Function

The RNG TRNG Statistical Check Poker Count 9 and 8 Register is a read-only register used to read the final Poker test counts of 9h and 8h patterns. The Poker 8h Count increments each time a nibble of sample data is found to be 8h. The Poker 9h Count increments each time a nibble of sample data is found to be 9h. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

## 13.80.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_9_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_8_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.80.4 Fields

| Field | Function |
|---|---|
| 31-16<br>PKR_9_CNT | Poker 9h Count. Total number of nibbles of sample data which were found to be 9h. Requires RTMCTL[PRGM] = 0. |
| 15-0<br>PKR_8_CNT | Poker 8h Count. Total number of nibbles of sample data which were found to be 8h. Requires RTMCTL[PRGM] = 0. |

## 13.81 RNG TRNG Statistical Check Poker Count B and A Register (RTPKRCNTBA)

### 13.81.1 Offset

| Register | Offset |
|---|---|
| RTPKRCNTBA | 694h |

### 13.81.2 Function

The RNG TRNG Statistical Check Poker Count B and A Register is a read-only register used to read the final Poker test counts of Bh and Ah patterns. The Poker Ah Count increments each time a nibble of sample data is found to be Ah. The Poker Bh Count increments each time a nibble of sample data is found to be Bh. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

### 13.81.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_B_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_A_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.81.4 Fields

| Field | Function |
|---|---|
| 31-16<br>PKR_B_CNT | Poker Bh Count. Total number of nibbles of sample data which were found to be Bh. Requires RTMCTL[PRGM] = 0. |
| 15-0<br>PKR_A_CNT | Poker Ah Count. Total number of nibbles of sample data which were found to be Ah. Requires RTMCTL[PRGM] = 0. |

## 13.82 RNG TRNG Statistical Check Poker Count D and C Register (RTPKRCNTDC)

### 13.82.1 Offset

| Register | Offset |
|---|---|
| RTPKRCNTDC | 698h |

### 13.82.2 Function

The RNG TRNG Statistical Check Poker Count D and C Register is a read-only register used to read the final Poker test counts of Dh and Ch patterns. The Poker Ch Count increments each time a nibble of sample data is found to be Ch. The Poker Dh Count increments each time a nibble of sample data is found to be Dh. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

### 13.82.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_D_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_C_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.82.4 Fields

| Field | Function |
|---|---|
| 31-16<br>PKR_D_CNT | Poker Dh Count. Total number of nibbles of sample data which were found to be Dh. Requires RTMCTL[PRGM] = 0. |
| 15-0<br>PKR_C_CNT | Poker Ch Count. Total number of nibbles of sample data which were found to be Ch. Requires RTMCTL[PRGM] = 0. |

## 13.83 RNG TRNG Statistical Check Poker Count F and E Register (RTPKRCNTFE)

## 13.83.1 Offset

| Register | Offset |
|---|---|
| RTPKRCNTFE | 69Ch |

## 13.83.2 Function

The RNG TRNG Statistical Check Poker Count F and E Register is a read-only register used to read the final Poker test counts of Fh and Eh patterns. The Poker Eh Count increments each time a nibble of sample data is found to be Eh. The Poker Fh Count increments each time a nibble of sample data is found to be Fh. Note that this register is readable only if RTMCTL[PRGM] is 0, otherwise zeroes will be read.

## 13.83.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_F_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKR_E_CNT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.83.4 Fields

| Field | Function |
|---|---|
| 31-16<br>PKR_F_CNT | Poker Fh Count. Total number of nibbles of sample data which were found to be Fh. Requires RTMCTL[PRGM] = 0. |
| 15-0<br>PKR_E_CNT | Poker Eh Count. Total number of nibbles of sample data which were found to be Eh. Requires RTMCTL[PRGM] = 0. |

# 13.84 RNG DRNG Status Register (RDSTA)

## 13.84.1 Offset

| Register | Offset | Description |
|---|---|---|
| RDSTA | 6C0h | Accessible at this address when RTMCTL[PRGM] = 0] |

## 13.84.2 Function

The RNG DRNG Status Register shows the current status of the DRNG portion of the RNG.

## 13.84.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SKVT | SKVN | Reserved | | | | | | | | | CE | ERRCODE | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | Reserved | | TF1 | TF0 | Reserved | | PR1 | PR0 | Reserved | | IF1 | IF0 |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.84.4 Fields

| Field | Function |
|---|---|
| 31<br>SKVT | Secure Key Valid Test. The secure keys (JDKEK, TDKEK and TDSK) were generated by a test (deterministic) instance. |
| 30<br>SKVN | Secure Key Valid Non-Test. The secure keys (JDKEK, TDKEK and TDSK) were generated by a non-test (non-deterministic) instance. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 29-21<br>— | Reserved |
| 20<br>CE | Catastrophic Error. A catastrophic error will occur when the RNG gets a hardware error while requesting new entropy and the current State Handle is instantiated as a non-test (non-deterministic) instance. |
| 19-16<br>ERRCODE | Error Code. These bits represent the current error in the RNG. |
| 15-12<br>— | Reserved |
| 11-10<br>— | Reserved |
| 9<br>TF1 | Test Flag State Handle 1. State handle 1 has been instantiated as a test (deterministic) instance. |
| 8<br>TF0 | Test Flag State Handle 0. State handle 0 has been instantiated as a test (deterministic) instance. |
| 7-6<br>— | Reserved |
| 5<br>PR1 | Prediction Resistance Flag State Handle 1. State Handle 1 has been instantiated to support prediction resistance. |
| 4<br>PR0 | Prediction Resistance Flag State Handle 0. State Handle 0 has been instantiated to support prediction resistance. |
| 3-2<br>— | Reserved |
| 1<br>IF1 | Instantiated Flag State Handle 1. State Handle 1 has been instantiated. |
| 0<br>IF0 | Instantiated Flag State Handle 0. State Handle 0 has been instantiated. |

## 13.85 RNG DRNG State Handle 0 Reseed Interval Register (RDINT0)

### 13.85.1 Offset

| Register | Offset |
|---|---|
| RDINT0 | 6D0h |

### 13.85.2 Function

The RNG DRNG State Handle 0 Reseed Interval Register shows the current value of the reseed interval for State Handle 0. This value represents the number of requests for random data from this State Handle before this State Handle is automatically reseeded with entropy from the TRNG. The reset value is zero, but a new reseed interval value is loaded when the RNG State Handle is instantiated. If the value in the Class 1 Data Size register is nonzero at the time that the instantiation command is executed, RDINT0 will be loaded with this value. If the value in the Class 1 Data Size register is 0, the default reseed interval value (10,000,000) is loaded into RDINT0. Note that the State Handle is instantiated by executing a descriptor that contains an ALGORITHM OPERATION RNG Instantiate command (see RNG operations).

### 13.85.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | RESINT0 | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | RESINT0 | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.85.4 Fields

| Field | Function |
|---|---|
| 31-0<br>RESINT0 | RESINT0. This read-only register holds the Reseed Interval for State Handle 0. |

## 13.86 RNG DRNG State Handle 1 Reseed Interval Register (RDINT1)

## 13.86.1 Offset

| Register | Offset |
|---|---|
| RDINT1 | 6D4h |

## 13.86.2 Function

The RNG DRNG State Handle 1 Reseed Interval Register shows the current value of the reseed interval for State Handle 1. This value represents the number of requests for random data from this State Handle before this State Handle is automatically reseeded with entropy from the TRNG. The reset value is zero, but a new reseed interval value is loaded when the RNG State Handle is instantiated. If the value in the Class 1 Data Size register is nonzero at the time that the instantiation command is executed, RDINT1 will be loaded with this value. If the value in the Class 1 Data Size register is 0, the default reseed interval value (10,000,000) is loaded into RDINT1. Note that the State Handle is instantiated by executing a descriptor that contains an ALGORITHM OPERATION RNG Instantiate command (see RNG operations).

## 13.86.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | RESINT1 | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | RESINT1 | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.86.4 Fields

| Field | Function |
|---|---|
| 31-0<br>RESINT1 | RESINT1. This read-only register holds the Reseed Interval for State Handle 1. |

# 13.87 RNG DRNG Hash Control Register (RDHCNTL)

## 13.87.1 Offset

| Register | Offset |
|---|---|
| RDHCNTL | 6E0h |

## 13.87.2 Function

The RNG DRNG Hash Control Register is used to gain control of the SHA-256 hashing engine that is internal to the RNG. Once Hashing test mode is initialized then the user can begin the hashing operation and poll for the done bit.

## 13.87.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | HTM | | | HD |
| W | | | | | | | | | | | | HTC | | HI | HB | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.87.4 Fields

| Field | Function |
|---|---|
| 31-5<br>— | Reserved |
| 4<br>HTC | Hashing Test Mode Clear. Writing this bit will take the RNG out of hashing test mode. |
| 3 | Hashing Test Mode. Writing this bit will put RNG in Hashing Test Mode. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| HTM | |
| 2<br>HI | Hashing Initialize. Writing to this bit will initialize the Hashing Engine. |
| 1<br>HB | Hashing Begin. Writing this bit will causing the Hashing Engine to begin hashing. |
| 0<br>HD | Hashing Done. This bit asserts when the hashing engine is done. |

# 13.88 RNG DRNG Hash Digest Register (RDHDIG)

## 13.88.1 Offset

| Register | Offset |
|---|---|
| RDHDIG | 6E4h |

## 13.88.2 Function

The RNG DRNG Hash Digest Register allows user access to the eight 32-bit message digest registers of the SHA-256 hashing engine that is internal to the RNG. All eight registers are read in order from most-significant bits to least-significant bits by reading this address eight times. These registers are only readable while in Hashing Test Mode and when the Hashing Engine is done.

## 13.88.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | HASHMD | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | HASHMD | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.88.4 Fields

| Field | Function |
|---|---|
| 31-0<br>HASHMD | HASHMD. Hashing Message Digest Register. This register needs to be read 8 times to retrieve the entire message digest. |

## 13.89 RNG DRNG Hash Buffer Register (RDHBUF)

### 13.89.1 Offset

| Register | Offset |
|---|---|
| RDHBUF | 6E8h |

### 13.89.2 Function

The RNG DRNG Hash buffer allows access to the SHA-256 hashing engine that is internal to the RNG for the purpose of conformance testing. To fill the buffer this register must be written 16 times at this address. This register is writable only while the RNG is in Hashing Test mode. This mode can be selected via the RNG DRNG Hash Control Register.

### 13.89.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | | | | | | | | | | | | | | |
| W | HASHBUF | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | | | | | | | | | | | | | | |
| W | HASHBUF | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.89.4 Fields

| Field | Function |
|---|---|
| 31-0<br>HASHBUF | HASHBUF. This write-only register provides access to the internal SHA-256 hashing engine's 64-byte buffer. This register must be written 16 times to fill the buffer. |

## 13.90 Recoverable Error Indication Status (REIS)

### 13.90.1 Offset

| Register | Offset |
|---|---|
| REIS | B00h |

### 13.90.2 Function

REIS indicates the assertion status of different SEC recoverable error indication sources (1 bit per source). Software can clear a bit in REIS by writing a 1 to that bit.

### 13.90.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | JBAE3 | JBAE2 | JBAE1 | JBAE0 | Reserved | | | | | | | RBAE |
| W | | | | | W1C | W1C | W1C | W1C | | | | | | | | W1C |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | Reserved | | QBAE | QHLT | Reserved | | | | | | | CWDE |
| W | | | | | | | W1C | W1C | | | | | | | | W1C |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.90.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved |
| 27<br>JBAE3 | A job descriptor executed from Job Ring 3 caused a bus access error. |
| 26<br>JBAE2 | A job descriptor executed from Job Ring 2 caused a bus access error. |
| 25<br>JBAE1 | A job descriptor executed from Job Ring 1 caused a bus access error. |
| 24<br>JBAE0 | A job descriptor executed from Job Ring 0 caused a bus access error. |
| 23-17<br>— | Reserved |
| 16<br>RBAE | A bus transaction initiated by SEC RTIC resulted in a bus access error. |
| 15-12<br>— | Reserved |
| 11-10<br>— | Reserved |
| 9<br>QBAE | A job initiated by SEC's Queue Manager Interface resulted in a bus access error. |
| 8<br>QHLT | SEC's Queue Manager Interface halted due to stop or stop on error. |
| 7-1<br>— | Reserved |
| 0<br>CWDE | The SEC watchdog timer expired. |

## 13.91 Recoverable Error Indication Halt (REIH)

## 13.91.1 Offset

| Register | Offset |
|---|---|
| REIH | B0Ch |

## 13.91.2 Function

Writing a 1 to an REIH bit indicates that SEC should be halted if the associated recoverable error occurs.

## 13.91.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | JBAE3 | JBAE2 | JBAE1 | JBAE0 | Reserved | | | | | | | RBAE |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | QFDD | QIVE | QBAE | QHLT | Reserved | | | | | | | CWDE |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.91.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved |
| 27<br>JBAE3 | Halt SEC if JR3-initiated job execution caused bus access error. |
| 26<br>JBAE2 | Halt SEC if JR2-initiated job execution caused bus access error. |
| 25<br>JBAE1 | Halt SEC if JR1-initiated job execution caused bus access error. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 24<br>JBAE0 | Halt SEC if JR0-initiated job execution caused bus access error. |
| 23-17<br>— | Reserved |
| 16<br>RBAE | Halt SEC if RTIC-initiated job execution caused bus access error. |
| 15-12<br>— | Reserved |
| 11<br>QFDD | Halt SEC if QI frame descriptor dropped. |
| 10<br>QIVE | Halt SEC if QI isolation violation error. |
| 9<br>QBAE | Halt SEC if QI-initiated job execution caused bus access error. |
| 8<br>QHLT | Halt SEC if QI halted (due to stop or stop on error). |
| 7-1<br>— | Reserved |
| 0<br>CWDE | Halt SEC if SEC watchdog timer expires. |

# 13.92 SEC Version ID Register, most-significant half (SECVID_MS)

## 13.92.1 Offset

| Register | Offset |
|---|---|
| SECVID_MS | BF8h (alias) |
| SECVID_MS | FF8h (alias) |
| SECVID_MS | 1_0FF8h (alias) |
| SECVID_MS | 2_0FF8h (alias) |
| SECVID_MS | 3_0FF8h (alias) |
| SECVID_MS | 4_0FF8h (alias) |
| SECVID_MS | 6_0FF8h (alias) |
| SECVID_MS | 7_0FF8h (alias) |

*Table continues on the next page...*

| Register | Offset |
|---|---|
| SECVID_MS | 8_0FF8h (alias) |
| SECVID_MS | 9_0FF8h (alias) |
| SECVID_MS | A_0FF8h (alias) |

## 13.92.2 Function

This register contains the ID for SEC and major and minor revision numbers. It also contains the integration options, ECO revision, and configuration options. Since this register holds more than 32 bits, it holds a 48-bit value but registers are accessible only as 32-bit words, the counter accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64KB address spaces. The register and its fields are described in the figure and table below.

## 13.92.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | IP_ID | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MAJ_REV | | | | | | | | MIN_REV | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

## 13.92.4 Fields

| Field | Function |
|---|---|
| 31-16<br>IP_ID | ID for SEC. |
| 15-8<br>MAJ_REV | Major revision number for SEC. |
| 7-0<br>MIN_REV | Minor revision number for SEC. |

# 13.93 SEC Version ID Register, least-significant half (SECVID_LS)

## 13.93.1 Offset

| Register | Offset |
|---|---|
| SECVID_LS | BFCh (alias) |
| SECVID_LS | FFCh (alias) |
| SECVID_LS | 1_0FFCh (alias) |
| SECVID_LS | 2_0FFCh (alias) |
| SECVID_LS | 3_0FFCh (alias) |
| SECVID_LS | 4_0FFCh (alias) |
| SECVID_LS | 6_0FFCh (alias) |
| SECVID_LS | 7_0FFCh (alias) |
| SECVID_LS | 8_0FFCh (alias) |
| SECVID_LS | 9_0FFCh (alias) |
| SECVID_LS | A_0FFCh (alias) |

## 13.93.2 Function

This register contains the ID for SEC and major and minor revision numbers. It also contains the integration options, ECO revision, and configuration options. Since this register holds more than 32 bits, it holds a 48-bit value but registers are accessible only as 32-bit words, the counter accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. The register and its fields are described in the figure and table below.

### 13.93.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | COMPILE_OPT | | | | | | | | INTG_OPT | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ECO_REV | | | | | | | | CONFIG_OPT | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.93.4 Fields

| Field | Function |
|---|---|
| 31-24<br>COMPILE_OPT | Compile options for SEC. |
| 23-16<br>INTG_OPT | Integration options for SEC. |
| 15-8<br>ECO_REV | ECO revision for SEC. |
| 7-0<br>CONFIG_OPT | Configuration options for SEC. |

## 13.94 Holding Tank 0 Job Descriptor Address (HT0_JD_ADDR)

### 13.94.1 Offset

| Register | Offset | Description |
|---|---|---|
| HT0_JD_ADDR | C00h | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |

### 13.94.2 Function

The HTa_JD_ADDR register holds the address of a Job Descriptor that is in a "holding tank" waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution.

The Job Queue Debug Select Register (JQ_DEBUG_SEL) HT_SEL field controls which holding tank supplies the Job Descriptor Address to the HTa_JD_ADDR.

### 13.94.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | JD_ADDR | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JD_ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JD_ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.94.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved |
| 39-0<br>JD_ADDR | Job Descriptor Address. |

# 13.95 Holding Tank 0 Shared Descriptor Address (HT0_SD_ADDR)

## 13.95.1 Offset

| Register | Offset | Description |
|---|---|---|
| HT0_SD_ADDR | C08h | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |

## 13.95.2 Function

The HTa_SD_ADDR register holds the address of a Shared Descriptor that is in a Holding Tank waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution via a job ring.

The Job Queue Debug Select Register (JQ_DEBUG_SEL) HT_SEL field controls which holding tank supplies the Shared Descriptor Address to the HTa_SD_ADDR.

### 13.95.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | SD_ADDR | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SD_ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SD_ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.95.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved |
| 39-0<br>SD_ADDR | Shared Descriptor Address. |

## 13.96 Holding Tank 0 Job Queue Control, most-significant half (HT0_JQ_CTRL_MS)

## 13.96.1 Offset

| Register | Offset | Description |
|---|---|---|
| HT0_JQ_CTRL_MS | C10h | Note that the addresses of the two halves of this register are unaffected by the endianness configuration. |

## 13.96.2 Function

The HTa_JQ_CTRL register holds the control information for a descriptor that is in a "holding tank" waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution. The most-significant half of HTa_JQ_CTRL is formatted the same as the DECO Job Queue Control Register, except that there is no STEP field or SING field as in the DECO Job Queue Control Register.

The Job Queue Debug Select Register (JQ_DEBUG_SEL) HT_SEL field controls which holding tank supplies the Shared Descriptor Address to the HTa_SD_ADDR.

## 13.96.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | WHL | FOUR | ILE | SHR_FROM | | | | | Reserved | | DWORD_SWAP | HT_ERROR | | SOB |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AMTD | JDIS | Reserved | | | SRC | | | Reserved | | | | ID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.96.4 Fields

| Field | Function |
|---|---|
| 31-30<br>— | Reserved |
| 29<br>WHL | Whole Descriptor. In versions of SEC that implement prefetching, the WHL field is interpreted in combination with the SOB field. In versions that don't implement prefetching, WHL=1 indicates that HT is passing the full job descriptor to DECO and therefore DECO does not need to fetch any additional Job Descriptor words from external memory. |
| 28<br>FOUR | Four Words. Job Queue Controller will pass at least 4 words of the descriptor to DECO. |
| 27<br>ILE | Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors.<br>ILE = 0: No byte-swapping is performed for data transferred to or from the Descriptor Buffer.<br>ILE = 1: Byte-swapping is performed when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: Input Data FIFO, Output Data FIFO, and Class 1 Context, Class 2 Context, Class1 Key and Class 2 Key registers. |
| 26-22<br>SHR_FROM | Share From. This is the DECO block from which the DECO block that runs this job will get the Shared Descriptor. This field is only used if the job queue controller wants this DECO to use a Shared Descriptor that is already in a DECO. This field is ignored when running descriptors via the IP bus (i.e. under the direct control of software). |
| 21-20<br>— | Reserved |
| 19<br>DWORD_SWAP | Double Word Swap.<br>0b - DWords are in the order most-significant word, least-significant word.<br>1b - DWords are in the order least-significant word, most-significant word. |
| 18-17<br>HT_ERROR | Holding Tank Error. (This field is implemented only in versions of SEC that support prefetching.)<br>00b - No error<br>01b - Job Descriptor or Shared Descriptor length error<br>10b - AXI_error while reading a job ring or QI Shared Descriptor or the remainder of a job ring Job Descriptor<br>11b - AXI error while reading QI input frame data |
| 16<br>SOB | Shared or Burst. (This field is implemented only in versions of SEC that support prefetching.) The SOB field is interpreted along with the WHL field as follows:<br>SOB=0 WHL=0 - No prefetch, not whole descriptor<br>SOB=0 WHL=1 - Got whole Job Descriptor, no Shared Descriptor or input frame data<br>SOB=1 WHL=0 - Got Shared Descriptor, no input frame data<br>SOB=1 WHL=1 - Got whole Job Descriptor and input frame data |
| 15<br>AMTD | Allow Make Trusted Descriptor. This field is read-only. If this bit is a 1, then a Job Descriptor with the MTD (Make Trusted Descriptor) bit set is allowed to execute. The bit will be 1 only if the Job Descriptor was run from a job ring with the AMTD bit set to 1 in the job ring's JRaICID Register. |
| 14<br>JDIS | Job Descriptor ICID Select. Determines whether the SEQ ICID or the Non-SEQ ICID is asserted when reading the Job Descriptor from memory.<br>0b - Non-SEQ ICID<br>1b - SEQ ICID |
| 13-11 | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 10-8<br>SRC | Job Source. Source of the job. Determines which set of DMA configuration attributes (e.g. JRCFGR_JRa_MS and endian configuration bits ) the DMA should use for bus transactions. It is illegal for the SRC field to have a value other than that of a job ring when running descriptors via the IP bus (i.e. under the direct control of software).<br>000b - job ring 0<br>001b - job ring 1<br>010b - job ring 2<br>011b - job ring 3<br>100b - RTIC<br>101b - QI<br>110b - Reserved<br>111b - Reserved |
| 7-4<br>— | Reserved |
| 3-0<br>ID | Job ID. Unique tag given to each job by its source. Used to tell the source that the job has completed. |

## 13.97 Holding Tank 0 Job Queue Control, least-significant half (HT0_JQ_CTRL_LS)

### 13.97.1 Offset

| Register | Offset | Description |
|---|---|---|
| HT0_JQ_CTRL_LS | C14h | Note that the addresses of the two halves of this register are unaffected by the endianness configuration. |

### 13.97.2 Function

The HTa_JQ_CTRL register holds the control information for a descriptor that is in a "holding tank" waiting to be loaded into a DECO. The register is intended to be used when debugging descriptor execution. The most-significant half of HTa_JQ_CTRL is formatted the same as the DECO Job Queue Control Register, except that there is no STEP field or SING field as in the DECO Job Queue Control Register.

The Job Queue Debug Select Register (JQ_DEBUG_SEL) HT_SEL field controls which holding tank supplies the Job Queue control data to the HTa_JQ_CTRL_MS.

### 13.97.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | NON_SEQ_ICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | SEQ_ICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.97.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved. |
| 27-16<br>NON_SEQ_ICID | Non-SEQ_ICID<br>This field defines the ICID value asserted for DMA transactions associated with external memory accesses for non-sequence commands, such as KEY, LOAD, and STORE. By default the Job Descriptor is read using this ICID value, although that behavior can be changed by setting the JDIS bit in the corresponding Job Ring Configuration Register.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |
| 15-12<br>— | Reserved. |
| 11-0<br>SEQ_ICID | This field defines the ICID value asserted for DMA transactions associated with external memory accesses for sequence commands, such as SEQ_KEY, SEQ_LOAD, and SEQ_STORE. Setting the JDIS bit in the corresponding Job Ring Configuration Register will cause the Job Queue to use this ICID value for Job Descriptor reads.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

## 13.98 Holding Tank Status (HT0_STATUS)

### 13.98.1 Offset

| Register | Offset |
|---|---|
| HT0_STATUS | C1Ch |

### 13.98.2 Function

The HT0_STATUS register holds the status information for a Job Descriptor "holding tank". The register is intended to be used when debugging descriptor execution.

The HT_SEL field in the Job Queue Debug Select Register (JQ_DEBUG_SEL) controls which holding tank supplies the status information to the HT0_STATUS.

### 13.98.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | BC | IN_USE | BB_IN_USE | OLD | Reserved | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | PEND_2 | PEND_1 | PEND_0 |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.98.4 Fields

| Field | Function |
|---|---|
| 31<br>BC | Been Changed. When using the Holding Tank debug registers, the Holding Tank Job Descriptor Address register should be the first register that is read. The BC ("Been Changed") bit is cleared when the Holding Tank Job Descriptor Address register is read. If data in the holding tanks changes after that time but before the HT Status register is read, the "Been Changed" bit is set. This indicates that the data read |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | from some of the HT debug registers may be inconsistent with data read from other HT debug registers. In this case the HT debug registers should be reread, starting with the Holding Tank Job Descriptor Address register. |
| 30<br>IN_USE | In Use. The "In use" bit is set when the HT contains some or all of the information for a job that has not yet been sent or not yet completely sent to a DECO. |
| 29<br>BB_IN_USE | BB In Use. The "BB In use" bit is set when the burst buffer contains some or all of the input frame data for a job that has not yet been sent or not yet completely sent to a DECO. |
| 28<br>OLD | Old BB data. The "OLD" bit is set when the burst buffer contains input frame data for the job that was previously in the associated HT. This input frame data has not yet been completely sent to a DECO, so the burst buffer is not available for the job currently in the HT. |
| 27-3<br>— | Reserved. |
| 2<br>PEND_2 | Pending for DECO 2. The PEND_2 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 2. It is possible for more than one pending bit for the holding tank to be set at the same time. |
| 1<br>PEND_1 | Pending for DECO 1. The PEND_1 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 1. It is possible for more than one pending bit for the holding tank to be set at the same time. |
| 0<br>PEND_0 | Pending for DECO 0. The PEND_0 bit for this holding tank is set if the shared descriptor in this holding tank matches the shared descriptor currently in DECO 0. It is possible for more than one pending bit for the holding tank to be set at the same time. |

## 13.99 Job Queue Debug Select Register (JQ_DEBUG_SEL)

### 13.99.1 Offset

| Register | Offset |
|---|---|
| JQ_DEBUG_SEL | C24h |

### 13.99.2 Function

The Job Queue Debug Select register is used to select which holding tank is being accessed in the holding tank debug registers (HTa Job Descriptor Address, HTa Shared Descriptor Address, HTa JQ Control, and HTa Status registers). The Job Queue Debug Select register is also used to select the ID of the job that is being queried in the Job Ring Job-Done Source and Job Ring Job-Done Descriptor Address registers. Finally, it specifies which FIFO index to report in the Job Ring Job-Done Job ID FIFO register.

If the value written to the HT_SEL field is larger than the number of holding tanks in SEC, a value of 0 will be stored in the HT_SEL field and Holding Tank 0 will be used by the HTa Job Descriptor Address, HTa Shared Descriptor Address, HTa JQ Control, and HTa Status registers.

## 13.99.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | JOB_ID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | HT_SEL | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.99.4 Fields

| Field | Function |
|---|---|
| 31-20<br>— | Reserved |
| 19-16<br>JOB_ID | Job ID. Specifies a Job ID for which to return a Job Source in the Job Ring Job-Done Source FIFO register or Descriptor address in the Job Ring Job-Done Descriptor Address register. Specifies a FIFO index for the Job ID returned by the Job Ring Job-Done Job IDFIFO register, where a value of 0 indicates the oldest job in the FIFO. |
| 15-2<br>— | Reserved |
| 1-0<br>HT_SEL | Holding Tank Select. Selects which holding tank is being accessed in the holding tank debug registers (HTa Job Descriptor Address, HTa Shared Descriptor Address, HTa JQ Control, and HTa Status registers). |

# 13.100 Job Ring Job IDs in Use Register, least-significant half (JRJIDU_LS)

## 13.100.1 Offset

| Register | Offset |
|---|---|
| JRJIDU_LS | DBCh |

## 13.100.2 Function

The Job Ring Job IDs in Use register indicates which of the Job IDs tracked by the Job Controller are currently in use (i.e. identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring). The register is intended to be used when debugging descriptor execution via a job ring. The JRJIDU contains a bit for each of the Job IDs, indicating whether that Job ID is currently in use.

## 13.100.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | JID11 | JID10 | JID09 | JID08 | JID07 | JID06 | JID05 | JID04 | JID03 | JID02 | JID01 | JID00 |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.100.4 Fields

| Field | Function |
|---|---|
| 31-12<br>— | Reserved. |
| 11<br>JID11 | Job ID 11. Job ID 11 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 10 | Job ID 10. Job ID 10 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| JID10 | |
| 9<br>JID09 | Job ID 09. Job ID 09 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 8<br>JID08 | Job ID 08. Job ID 08 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 7<br>JID07 | Job ID 07. Job ID 07 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 6<br>JID06 | Job ID 06. Job ID 06 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 5<br>JID05 | Job ID 05. Job ID 05 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 4<br>JID04 | Job ID 04. Job ID 04 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 3<br>JID03 | Job ID 03. Job ID 03 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 2<br>JID02 | Job ID 02. Job ID 02 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 1<br>JID01 | Job ID 01. Job ID 01 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |
| 0<br>JID00 | Job ID 00. Job ID 00 is currently in use identifying a job that is present in a holding tank, in a DECO, or in the completed Job Queue waiting for the Job Completion status to be written to an output ring. |

## 13.101 Job Ring Job-Done Job ID FIFO BC (JRJDJIFBC)

### 13.101.1 Offset

| Register | Offset |
|---|---|
| JRJDJIFBC | DC0h |

### 13.101.2 Function

This register indicates whether consistent data has been read from the JRJDJI FIFO and JRJDS and JRJDDA registers.

The Job Queue maintains an ordered list of Job IDs for the completed jobs whose completion status is waiting to be written to a job ring output ring. The Job Ring Job-Done Job ID FIFO register returns the Job ID located at the index specified in the JOB_ID field of the Job Queue Debug Select Register (JQ_DEBUG_SEL). When the JOB_ID field is set to 0, the oldest JOB_ID in the Job-Done FIFO is returned. See Section Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR3). Note that these Job IDs are not reset as job status is written to output rings, so the completion status for some Job IDs that appear in these registers may already have been written to output rings.

### 13.101.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | BC | Reserved | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.101.4 Fields

| Field | Function |
|---|---|
| 31<br>BC | Been changed. The hardware sets BC to 0 when the job descriptor address for Job ID 0 is read from Job Ring Job-Done Descriptor Address (JRJDDA). The hardware sets BC to 1 when any job is added or removed from the Job-Done Job ID FIFO. After software reads the JRJDJIF, JRJIDU, JRJDS1, and JRJDDA registers software should read BC. If the BC bit is 1, the results read from the JRJDJIF, JRJIDU, JRJDS1, and JRJDDA may be inconsistent with each other. |
| 30-0<br>— | Reserved |

## 13.102 Job Ring Job-Done Job ID FIFO (JRJDJIF)

### 13.102.1 Offset

| Register | Offset |
|---|---|
| JRJDJIF | DC4h |

### 13.102.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | JOB_ID_ENTRY | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.102.3 Fields

| Field | Function |
|---|---|
| 31-4<br>— | Reserved |
| 3-0<br>JOB_ID_ENTRY | Job ID entry. This field contains the Job ID of a job whose completion status is located at the JQ_DEBUG_SEL[JOB-ID] index in the Job-Done FIFO. |

## 13.103 Job Ring Job-Done Source 1 (JRJDS1)

### 13.103.1 Offset

| Register | Offset | Description |
|---|---|---|
| JRJDS1 | DE4h | The source for the job with the Job ID specified in JQ_DEBUG_SEL[JOB-ID]. |

### 13.103.2 Function

The Job Queue keeps track of the job source (job ring numbers) for each Job ID, and values in this register are updated whenever a new job ring job starts in a holding tank. Each entry in this register is matched to corresponding entries in the JRJDV and JRDDAa registers.

### 13.103.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | VALID | Reserved | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | SRC | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.103.4 Fields

| Field | Function |
|---|---|
| 31<br>VALID | Valid. If this bit is 1, the job located at the index specified by the JOB_ID field in the Job Queue Debug Select register is complete, but its status has not yet been written to the output ring. |
| 30-2<br>— | Reserved |
| 1-0<br>SRC | Source. This field contains the number of the job ring that was the source of a job whose completion status is waiting to be written to an output ring. The job is located in the FIFO at the index specified by the JOB_ID field in the Job Queue Debug Select register. |

## 13.104 Job Ring Job-Done Descriptor Address 0 Register (JRJDDA)

## 13.104.1 Offset

| Register | Offset | Description |
|---|---|---|
| JRJDDA | E00h | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R). |

## 13.104.2 Function

The JRJDDA register is used to store the address of a job descriptor when the job is sent to a holding tank. The descriptor address read from JRJDDA is the one corresponding to the Job ID specified in the JOB_ID field of the Job Queue Debug Select Register (JQ_D EBUG_SEL). See Section Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR3). Because these addresses are updated only when a new job starts in a holding tank, some addresses read from this register may be for completed jobs that have already been written to an output ring. This register is intended to be used when debugging descriptor execution via a job ring.

## 13.104.3 Diagram

<table>
<tr><td>Bits</td><td>63</td><td>62</td><td>61</td><td>60</td><td>59</td><td>58</td><td>57</td><td>56</td><td>55</td><td>54</td><td>53</td><td>52</td><td>51</td><td>50</td><td>49</td><td>48</td></tr>
<tr><td>R</td><td colspan="16" rowspan="2">Reserved</td></tr>
<tr><td>W</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

<table>
<tr><td>Bits</td><td>47</td><td>46</td><td>45</td><td>44</td><td>43</td><td>42</td><td>41</td><td>40</td><td>39</td><td>38</td><td>37</td><td>36</td><td>35</td><td>34</td><td>33</td><td>32</td></tr>
<tr><td>R</td><td colspan="8" rowspan="2">Reserved</td><td colspan="8">JD_ADDR</td></tr>
<tr><td>W</td><td colspan="8"></td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

<table>
<tr><td>Bits</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td>R</td><td colspan="16">JD_ADDR</td></tr>
<tr><td>W</td><td colspan="16"></td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

<table>
<tr><td>Bits</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>R</td><td colspan="16">JD_ADDR</td></tr>
<tr><td>W</td><td colspan="16"></td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

### 13.104.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved |
| 39-0<br>JD_ADDR | Job Descriptor Address. |

## 13.105 Performance Counter, Number of Requests Dequeued (PC_REQ_DEQ)

### 13.105.1 Offset

| Register | Offset | Description |
|---|---|---|
| PC_REQ_DEQ | F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_REQ_DEQ | 1_0F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_REQ_DEQ | 2_0F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_REQ_DEQ | 3_0F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_REQ_DEQ | 4_0F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_REQ_DEQ | 6_0F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_REQ_DEQ | 7_0F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_REQ_DEQ | 8_0F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| PC_REQ_DEQ | 9_0F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_REQ_DEQ | A_0F00h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

## 13.105.2 Function

The PC_REQ_DEQ register counts the total number of DECO jobs that SEC has started. The counter is incremented whenever the job queue controller register in DECO is written to start a job. The job could originate from the Queue Manager Interface, one of the job rings, from RTIC or from the register interface when the DECO is under the direct control of software.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

## 13.105.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_REQ_DEQ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_REQ_DEQ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_REQ_DEQ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.105.4 Fields

| Field | Function |
|---|---|
| 63-48<br>— | Reserved |
| 47-0<br>PC_REQ_DEQ | Performance Counter Requests Dequeued. |

## 13.106 Performance Counter, Number of Outbound Encrypt Requests (PC_OB_ENC_REQ)

### 13.106.1 Offset

| Register | Offset | Description |
|---|---|---|
| PC_OB_ENC_REQ | F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENC_REQ | 1_0F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENC_REQ | 2_0F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENC_REQ | 3_0F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENC_REQ | 4_0F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENC_REQ | 6_0F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENC_REQ | 7_0F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENC_REQ | 8_0F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| PC_OB_ENC_REQ | 9_0F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENC_REQ | A_0F08h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

## 13.106.2 Function

The PC_OB_ENC_REQ register counts the total number of requests for symmetric encryption (excluding requests associated with blob encapsulations and encryption of Black Keys). If a descriptor specifies an encryption operation, the register is incremented at the time that the encryption operation completes. Note that a single descriptor containing multiple encryption commands could cause this register to increment more than once. The LSB of the Class 1 Mode register determines if this register or PC_IB_DEC_REQ is incremented.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

### 13.106.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_OB_ENC_REQ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_OB_ENC_REQ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_OB_ENC_REQ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.106.4 Fields

| Field | Function |
|---|---|
| 63-48<br>— | Reserved |
| 47-0<br>PC_OB_ENC_REQ | Performance Counter Outbound Encryption Requests. |

## 13.107 Performance Counter, Number of Inbound Decrypt Requests (PC_IB_DEC_REQ)

## 13.107.1 Offset

| Register | Offset | Description |
|---|---|---|
| PC_IB_DEC_REQ | F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DEC_REQ | 1_0F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DEC_REQ | 2_0F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DEC_REQ | 3_0F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DEC_REQ | 4_0F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DEC_REQ | 6_0F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DEC_REQ | 7_0F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DEC_REQ | 8_0F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DEC_REQ | 9_0F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DEC_REQ | A_0F10h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

## 13.107.2 Function

The PC_IB_DEC_REQ register counts the total number of requests for symmetric decryptions (excluding blob decapsulations and decryptions of Black Keys). If a descriptor specifies a decryption operation, the register is incremented at the time that the decryption operation completes. Note that a single descriptor containing multiple decryption commands could cause this register to increment more than once. The LSB of the Class 1 Mode register determines if this register or PC_OB_ENC_REQ is incremented.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

## 13.107.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_IB_DEC_REQ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_IB_DEC_REQ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_IB_DEC_REQ | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.107.4 Fields

| Field | Function |
|---|---|
| 63-48<br>— | Reserved |
| 47-0<br>PC_IB_DEC_REQ | Performance Counter Inbound Decryptions Requested |

# 13.108 Performance Counter, Number of Outbound Bytes Encrypted (PC_OB_ENCRYPT)

## 13.108.1 Offset

| Register | Offset | Description |
|---|---|---|
| PC_OB_ENCRYPT | F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENCRYPT | 1_0F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENCRYPT | 2_0F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENCRYPT | 3_0F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENCRYPT | 4_0F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENCRYPT | 6_0F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENCRYPT | 7_0F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENCRYPT | 8_0F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENCRYPT | 9_0F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_ENCRYPT | A_0F18h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

## 13.108.2 Function

The PC_OB_ENCRYPT register counts the total number of bytes encrypted with a symmetric key algorithm (excluding blob encapsulations and encryptions of Black Keys). PC_OB_ENCRYPT is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1, with the following exceptions. If the operation is AES-CMAC or AES-XCBC-MAC with the “no encryption” option, or the operation is Kasumi f9, the PC_OB_ENCRYPT register is not incremented but the

PC_OB_PROTECT register is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1. PC_OB_PROTECT is incremented by the value written to the “SAD Data Size” alias of the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

## 13.108.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_OB_ENCRYPT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_OB_ENCRYPT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_OB_ENCRYPT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.108.4 Fields

| Field | Function |
|---|---|
| 63-48<br>— | Reserved |
| 47-0<br>PC_OB_ENCRYPT | Performance Counter Outbound Bytes Encrypted. |

# 13.109 Performance Counter, Number of Outbound Bytes Protected (PC_OB_PROTECT)

## 13.109.1 Offset

| Register | Offset | Description |
|---|---|---|
| PC_OB_PROTECT | F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_PROTECT | 1_0F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_PROTECT | 2_0F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_PROTECT | 3_0F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_PROTECT | 4_0F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_PROTECT | 6_0F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_PROTECT | 7_0F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_PROTECT | 8_0F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_PROTECT | 9_0F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_OB_PROTECT | A_0F20h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

## 13.109.2 Function

The PC_OB_PROTECT register counts the total number of bytes protected—that is, the number of outbound bytes over which an integrity check value (ICV) was computed (for example, in HMAC and CMAC). (Note that this excludes blob encapsulations and CCM encryptions of Black Keys.) PC_OB_PROTECT is incremented by the value written to the Class 2 Data Size register if the AP bit in the Class 2 Mode register is set to 1. PC_OB_PROTECT is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1 and the operation is AES-CMAC or AES-XCBC-MAC with the “no encryption” option, or the operation is Kasumi f9. PC_OB_PROTECT is incremented by the value written to the “SAD Data Size” alias of the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1. For AES-GCM, AES-CCM, AES-CBC-XCBC, AES-CTR-XCBC, AES-CBC-CMAC and AES-CTR-CMAC operations both the PC_OB_PROTECT register and the PC_OB_ENCRYPT register will be incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 1.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

## 13.109.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_OB_PROTECT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_OB_PROTECT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_OB_PROTECT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.109.4 Fields

| Field | Function |
|---|---|
| 63-48<br>— | Reserved |
| 47-0<br>PC_OB_PROTECT | Performance Counter Outbound Bytes Encrypted. |

## 13.110 Performance Counter, Number of Inbound Bytes Decrypted (PC_IB_DECRYPT)

### 13.110.1 Offset

| Register | Offset | Description |
|---|---|---|
| PC_IB_DECRYPT | F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DECRYPT | 1_0F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DECRYPT | 2_0F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DECRYPT | 3_0F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DECRYPT | 4_0F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DECRYPT | 6_0F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DECRYPT | 7_0F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DECRYPT | 8_0F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_DECRYPT | 9_0F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| PC_IB_DECRYPT | A_0F28h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

## 13.110.2 Function

The PC_IB_DECRYPT register counts the total number of bytes decrypted with a symmetric key algorithm (excluding blob decapsulations and decryptions of Black Keys). PC_IB_DECRYPT is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 0, with the following exceptions. If the operation is AES-CMAC or AES-XCBC-MAC with the “no encryption” option, or the operation is Kasumi f9, the PC_IB_DECRYPT register is not incremented but the PC_IB_VALIDATED register is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 0. Note that writing to the “SAD Data Size” alias of the Class 1 Data Size register causes the PC_IB_VALIDATED register to be incremented rather than the PC_IB_DECRYPT register.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

## 13.110.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_IB_DECRYPT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_IB_DECRYPT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_IB_DECRYPT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.110.4 Fields

| Field | Function |
|---|---|
| 63-48<br>— | Reserved |
| 47-0<br>PC_IB_DECRYPT | Performance Counter Inbound Bytes Decrypted. |

## 13.111 Performance Counter, Number of Inbound Bytes Validated. (PC_IB_VALIDATED)

## 13.111.1 Offset

| Register | Offset | Description |
|---|---|---|
| PC_IB_VALIDATED | F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_VALIDATED | 1_0F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_VALIDATED | 2_0F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_VALIDATED | 3_0F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_VALIDATED | 4_0F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_VALIDATED | 6_0F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_VALIDATED | 7_0F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_VALIDATED | 8_0F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_VALIDATED | 9_0F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |
| PC_IB_VALIDATED | A_0F30h (alias) | These addresses are for the least-significant 32 bits; the most significant 32 bits can be accessed at these addresses +4h. |

## 13.111.2 Function

The PC_IB_VALIDATED register counts the total number of bytes validated (that is, the number of inbound bytes over which an ICV was computed for comparison purposes). PC_IB_VALIDATED is incremented by the value written to the Class 2 Data Size register if the AP bit in the Class 2 Mode register is set to 0. PC_IB_VALIDATED is incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 0 and the operation is AES-CMAC or AES-XCBC-MAC with the “no encryption” option, or the operation is Kasumi f9. PC_IB_VALIDATED is incremented by the value written to the “SAD Data Size” alias of the Class 1 Data Size register the ENC bit in the Class 1 Mode register is set to 0. For AES-GCM, AES-CCM, AES-CBC-XCBC, AES-CTR-XCBC, AES-CBC-CMAC and AES-CTR-CMAC operations both the PC_IB_VALIDATED register and the PC_IB_DECRYPT register will be incremented by the value written to the Class 1 Data Size register if the ENC bit in the Class 1 Mode register is set to 0.

Since this register is greater than 32 bits, it must be accessed as two 32-bit words. When reading or writing the register first access the lower address, then the higher address. This ensures that a consistent 48-bit value is read or written despite the fact that the register value may increment between accessing the two halves of the register.

**NOTE**

This counter does not include the number of bytes of the received ICV. Also, it increments whether the ICV comparison was successful or not.

## 13.111.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_IB_VALIDATED | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_IB_VALIDATED | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PC_IB_VALIDATED | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.111.4 Fields

| Field | Function |
|---|---|
| 63-48 | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 47-0<br>PC_IB_VALIDATED | Performance Counter Inbound Bytes Validated. |

## 13.112 CHA Revision Number Register, most-significant half (CRNR_MS)

### 13.112.1 Offset

| Register | Offset |
|---|---|
| CRNR_MS | FA0h (alias) |
| CRNR_MS | 1_0FA0h (alias) |
| CRNR_MS | 2_0FA0h (alias) |
| CRNR_MS | 3_0FA0h (alias) |
| CRNR_MS | 4_0FA0h (alias) |
| CRNR_MS | 6_0FA0h (alias) |
| CRNR_MS | 7_0FA0h (alias) |
| CRNR_MS | 8_0FA0h (alias) |
| CRNR_MS | 9_0FA0h (alias) |
| CRNR_MS | A_0FA0h (alias) |

### 13.112.2 Function

The CHA Revision Number register indicates the revision number of each CHA. The revisions are numbered independently for each version of a particular CHA (see CHA Version ID Register, most-significant half (CHAVID_MS)). Since the register is larger than 32 bits, the CRNR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple pages. The register format is shown in the figure and table below.

## 13.112.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JRRN | | | | DECORN | | | | Reserved | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ZARN | | | | ZERN | | | | SNW9RN | | | | CRCRN | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |

## 13.112.4 Fields

| Field | Function |
|---|---|
| 31-28<br>JRRN | Job Ring Revision Number |
| 27-24<br>DECORN | DECO Revision Number |
| 23-16<br>— | Reserved |
| 15-12<br>ZARN | ZUC Authentication Hardware Accelerator Revision Number |
| 11-8<br>ZERN | ZUC Encryption Hardware Accelerator Revision Number |
| 7-4<br>SNW9RN | SNOW-f9 Hardware Accelerator Revision Number |
| 3-0<br>CRCRN | CRC Hardware Accelerator Revision Number |

# 13.113 CHA Revision Number Register, least-significant half (CRNR_LS)

## 13.113.1 Offset

| Register | Offset |
|---|---|
| CRNR_LS | FA4h (alias) |
| CRNR_LS | 1_0FA4h (alias) |
| CRNR_LS | 2_0FA4h (alias) |
| CRNR_LS | 3_0FA4h (alias) |
| CRNR_LS | 4_0FA4h (alias) |
| CRNR_LS | 6_0FA4h (alias) |
| CRNR_LS | 7_0FA4h (alias) |
| CRNR_LS | 8_0FA4h (alias) |
| CRNR_LS | 9_0FA4h (alias) |
| CRNR_LS | A_0FA4h (alias) |

## 13.113.2 Function

The CHA Revision Number register indicates the revision number of each CHA. The revisions are numbered independently for each version of a particular CHA (see CHA Version ID Register, most-significant half (CHAVID_MS)). Since the register is larger than 32 bits, the CRNR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple pages. The register format is shown in the figure and table below.

## 13.113.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKRN | | | | KASRN | | | | SNW8RN | | | | RNGRN | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MDRN | | | | Reserved | | | | DESRN | | | | AESRN | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |

## 13.113.4 Fields

| Field | Function |
|---|---|
| 31-28<br>PKRN | Public Key Hardware Accelerator Revision Number<br>For PKHA-XT, PKRN=1.<br>For PKHA-SD, see below.<br>0000b - PKHA-SDv1<br>0001b - PKHA-SDv2<br>0010b - PKHA-SDv3<br>0011b - PKHA-SDv4 |
| 27-24<br>KASRN | Kasumi f8/f9 Hardware Accelerator Revision Number |
| 23-20<br>SNW8RN | SNOW-f8 Hardware Accelerator Revision Number |
| 19-16<br>RNGRN | Random Number Generator Revision Number. |
| 15-12<br>MDRN | Message Digest Hardware Accelerator module Revision Number. |
| 11-8<br>— | Reserved |
| 7-4<br>DESRN | DES Accelerator Revision Number. |
| 3-0<br>AESRN | AES Accelerator Revision Number.<br>0000 No Differential Power Analysis resistance implemented<br>0001 Differential Power Analysis resistance implemented<br>For all other values when AESVID = 4, Differential Power Analysis resistance is implemented. |

# 13.114 Compile Time Parameters Register, most-significant half (CTPR_MS)

## 13.114.1 Offset

| Register | Offset |
|---|---|
| CTPR_MS | FA8h (alias) |
| CTPR_MS | 1_0FA8h (alias) |
| CTPR_MS | 2_0FA8h (alias) |
| CTPR_MS | 3_0FA8h (alias) |

*Table continues on the next page...*

| Register | Offset |
|---|---|
| CTPR_MS | 4_0FA8h (alias) |
| CTPR_MS | 6_0FA8h (alias) |
| CTPR_MS | 7_0FA8h (alias) |
| CTPR_MS | 8_0FA8h (alias) |
| CTPR_MS | 9_0FA8h (alias) |
| CTPR_MS | A_0FA8h (alias) |

## 13.114.2 Function

The Compile Time Parameters register indicates the parameter settings at the time SEC was compiled. Since the register is larger than 32 bits, the CTPR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

## 13.114.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AXI_PIPE_DEPTH | | | | AXI_LIODN | AXI_PRI | QI | ACC_CTL | C1C2 | Reserved | PC | DECO_WD | PM_EVT_BUS | SG8 | MCFG_PS | MCFG_BURST |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | IP_CLK | DPAA2 | Reserved | AI_INCL | RNG_I | | | Reserved | | | REG_PG_SIZE | Reserved | | VIRT_EN_POR_VALUE | VIRT_EN_INCL |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

## 13.114.4 Fields

| Field | Function |
|---|---|
| 31-28<br>AXI_PIPE_DEPTH | AXI Pipeline Depth. A value of 0b0000 indicates maximum value. |
| 27<br>AXI_LIODN | LIODN logic included<br>1b - The logic to select LIODNs is included in SEC. Although this SoC uses ICIDs, the SEC manipulates the ICID values in the same manner as LIODNs in older QorIQ SoCs. |
| 26<br>AXI_PRI | AXI Master Priority<br>1b - The logic for the AXI Master Priority signals is included in SEC. Note that the presence of this logic will NOT have an effect in this SoC. |
| 25<br>QI | QMI included<br>1b - The Queue Manager Interface is included in SEC |
| 24<br>ACC_CTL | MID/DID-based access control<br>1b - SEC implements MID-based or DID-based access control for the IP Bus registers |
| 23<br>C1C2 | Separate C1 and C2 registers<br>1b - SEC implements Class 2 Key and Context registers that are separate from the Class 1 Key and Context registers |
| 22<br>— | Reserved |
| 21<br>PC | Performance Counter registers implemented<br>1b - SEC implements Performance Counter registers |
| 20<br>DECO_WD | DECO Watchdog Counter implemented<br>1b - SEC implements a DECO Watchdog Counter |
| 19<br>PM_EVT_BUS | Performance Monitor Event Bus.<br>1b - SEC implements a Performance Monitor Event Bus. |
| 18<br>SG8 | Eight Scatter-Gather Tables implemented<br>1b - SEC implements eight Scatter-Gather Tables, rather than just one. |
| 17<br>MCFG_PS | Pointer Size field implemented<br>1b - The Master Configuration Register contains a Pointer Size field |
| 16<br>MCFG_BURST | AXI Burst field implemented<br>1b - The Master Configuration Register contains an AXI Burst field. |
| 15<br>— | Reserved |
| 14<br>IP_CLK | IP Bus Slave Clock<br>0b - The frequency of SEC's IP Bus Slave Clock is the same as the frequency of SEC's AXI bus clock.<br>1b - The frequency of SEC's IP Bus Slave Clock is one-half the frequency of SEC's AXI bus clock. |
| 13<br>DPAA2 | This version of SEC supports version 2 of the Data Path Acceleration Architecture (DPAA2). |
| 12<br>— | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 11<br>AI_INCL | 0 : This version of SEC does **not** implement an AIOP interface.<br>1 : This version of SEC implements one or more AIOP interfaces. |
| 10-8<br>RNG_I | RNG Instantiations. This indicates the number of RNG instantiations that are implemented. Note that each instantiation is the data context for an independent RNG stream, and may share the same RNG hardware as other instantiations. The number of hardware RNGs is indicated in the RNGNUM field of the CHANUM register. |
| 7-5<br>— | Reserved. |
| 4<br>REG_PG_SIZE | SEC register page size.<br>0b - SEC uses 4Kbyte register pages.<br>1b - SEC uses 64Kbyte register pages. |
| 3-2<br>— | Reserved |
| 1<br>VIRT_EN_POR_VALUE | Job Ring Virtualization POR state. If VIRT_EN_POR_VALUE=1, this indicates that job ring virtualization is enabled at power up. |
| 0<br>VIRT_EN_INCL | Job Ring Virtualization programmable. If this bit is 1, this indicates that job ring virtualization can be programmed to be enabled or disabled by writing to the VIRT_EN bit in the Security Configuration register. If this bit is 0, job ring virtualization is always enabled and the Security Configuration register does not contain a VIRT_EN bit. |

## 13.115 Compile Time Parameters Register, least-significant half (CTPR_LS)

### 13.115.1 Offset

| Register | Offset |
|---|---|
| CTPR_LS | FACh (alias) |
| CTPR_LS | 1_0FACh (alias) |
| CTPR_LS | 2_0FACh (alias) |
| CTPR_LS | 3_0FACh (alias) |
| CTPR_LS | 4_0FACh (alias) |
| CTPR_LS | 6_0FACh (alias) |
| CTPR_LS | 7_0FACh (alias) |
| CTPR_LS | 8_0FACh (alias) |
| CTPR_LS | 9_0FACh (alias) |
| CTPR_LS | A_0FACh (alias) |

### 13.115.2 Function

The Compile Time Parameters register indicates the parameter settings at the time SEC was compiled. Since the register is larger than 32 bits, the CTPR fields are accessed as two 32-bit words. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

### 13.115.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | SPLIT_KEY | MAN_PROT | DBL_CRC | P3G_LTE | RSA | MACSEC | TLS_PRF | SSL_TLS | IKE | IPSEC | SRTP | WIMAX | WIFI | BLOB | KG_DS |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

### 13.115.4 Fields

| Field | Function |
|---|---|
| 31-15<br>— | Reserved. |
| 14<br>SPLIT_KEY | Split key protocol<br>1b - SEC implements the split-key protocol. |
| 13<br>MAN_PROT | Manufacturing Protection protocol<br>1b - SEC implements the two Manufacturing Protection functions. |
| 12<br>DBL_CRC | DOuble CRC protocol<br>1b - SEC implements specialized support for 3G Double CRC. |
| 11<br>P3G_LTE | 3GPP/LTE protocol<br>1b - SEC implements specialized support for 3G and LTE protocols. |
| 10 | RSA protocol |

*Table continues on the next page...*

| Field | Function |
|---|---|
| RSA | 1b - SEC implements specialized support for RSA encrypt and decrypt operations. |
| 9<br>MACSEC | MACSEC protocol<br>1b - SEC implements specialized support for the MACSEC protocol. |
| 8<br>TLS_PRF | TLS PRF protocol<br>1b - SEC implements specialized support for the TLS protocol pseudo-random function." |
| 7<br>SSL_TLS | SSL/TLS protocol<br>1b - SEC implements specialized support for the SSL and TLS protocols. |
| 6<br>IKE | IKE protocols<br>1b - SEC implements specialized support for the IKE protocol. |
| 5<br>IPSEC | IPSEC protocols<br>1b - SEC implements specialized support for the IPSEC protocol. |
| 4<br>SRTP | SRTP protocol<br>1b - SEC implements specialized support for the SRTP protocol. |
| 3<br>WIMAX | WiMax protocol<br>1b - SEC implements specialized support for the WIMAX protocol. |
| 2<br>WIFI | WiFi protocol<br>1b - SEC implements specialized support for the WIFI protocol. |
| 1<br>BLOB | Blob protocol<br>1b - SEC implements specialized support for encapsulating and decapsulating cryptographic blobs. |
| 0<br>KG_DS | PK generation and digital signature protcols<br>1b - SEC implements specialized support for Public Key Generation and Digital Signatures. |

## 13.116 Fault Address Register (FAR)

### 13.116.1 Offset

| Register | Offset | Description |
|---|---|---|
| FAR | FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |
| FAR | 1_0FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |
| FAR | 2_0FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit |

Table continues on the next page...

| Register | Offset | Description |
|---|---|---|
| | | description in Master Configuration Register (MCFGR). |
| FAR | 3_0FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |
| FAR | 4_0FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |
| FAR | 6_0FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |
| FAR | 7_0FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |
| FAR | 8_0FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |
| FAR | 9_0FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |
| FAR | A_0FC0h (alias) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |

## 13.116.2 Function

The Fault Address Register is used for software debugging of external memory access errors. This register will hold the value of the AXI address where a read or write error occurred. The read error address is aligned to the data bus address boundary of the data sample where the error occurred. The write error address is the starting address of the transaction, aligned to the data bus address boundary. Additional details concerning the bus transaction appear in the FADR (see Fault Address Detail Register (FADR)). The associated ICID is in the Fault Address ICID Register (see Section Fault Address ICID Register (FAICID)). Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. The values in the Fault Address Register, the Fault Address ICID Register and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers (including both halves of FAR) have been read, in any order, whereupon all these registers will be cleared.

## 13.116.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | FAR | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | FAR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | FAR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.116.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved |
| 39-0<br>FAR | Fault Address. This is the AXI address at which the error occurred. If multiple errors occur, this is the AXI address at which the first error occurred. This address will remain in the register until software has read both the upper and lower halves of the register. |

# 13.117 Fault Address ICID Register (FAICID)

## 13.117.1 Offset

| Register | Offset |
|---|---|
| FAICID | FC8h (alias) |
| FAICID | 1_0FC8h (alias) |
| FAICID | 2_0FC8h (alias) |
| FAICID | 3_0FC8h (alias) |
| FAICID | 4_0FC8h (alias) |
| FAICID | 6_0FC8h (alias) |
| FAICID | 7_0FC8h (alias) |
| FAICID | 8_0FC8h (alias) |
| FAICID | 9_0FC8h (alias) |
| FAICID | A_0FC8h (alias) |

## 13.117.2 Function

The Fault Address ICID Register is used by software for debugging external memory access errors. This register indicates the ICID associated with the AXI transaction where the error occurred. The associated AXI address is in the Fault Address Register (see Fault Address Register (FAR)) and additional details appear in the Fault Address Detail Register (see Fault Address Detail Register (FADR)). Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. The values in the Fault Address Register, the Fault Address ICID Register and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers (including both halves of FAR) have been read, in any order, whereupon all these registers will be cleared.

## 13.117.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | FICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.117.4 Fields

| Field | Function |
|---|---|
| 31-12<br>— | Reserved. |
| 11-0<br>FICID | DMA transaction ICID. This was the ICID associated with the DMA transaction that failed.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

## 13.118 Fault Address Detail Register (FADR)

### 13.118.1 Offset

| Register | Offset |
|---|---|
| FADR | FCCh (alias) |
| FADR | 1_0FCCh (alias) |
| FADR | 2_0FCCh (alias) |
| FADR | 3_0FCCh (alias) |
| FADR | 4_0FCCh (alias) |
| FADR | 6_0FCCh (alias) |
| FADR | 7_0FCCh (alias) |
| FADR | 8_0FCCh (alias) |
| FADR | 9_0FCCh (alias) |
| FADR | A_0FCCh (alias) |

### 13.118.2 Function

The Fault Address Detail Register is used by software for debugging external memory access errors. This register will hold details about the AXI transaction where the error occurred. The associated AXI address is in the Fault Address Register (FAR). The associated ICID is in the Fault Address ICID Register (FAICID). Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces. The values in the Fault Address Register, the Fault

Address ICID Register and the Fault Address Detail Register are stored (and no additional address fault data is recorded) until all these registers (including both halves of FAR) have been read, in any order, whereupon all these registers will be cleared.

## 13.118.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | FERR | | Reserved | | | | | | Reserved | | | | | FSZ_EXT | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | DTYP | JSRC | | | BLKID | | | | TYP | FSZ | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.118.4 Fields

| Field | Function |
|---|---|
| 31-30<br>FERR | Fault Error Code. This is the AXI Error Response Code.<br>00b - OKAY - Normal Access<br>01b - Reserved<br>10b - SLVERR - Slave Error<br>11b - DECERR - Decode Error |
| 29-24<br>— | Reserved. Always 0. |
| 23-19<br>— | Reserved. Always 0. |
| 18-16<br>FSZ_EXT | AXI Transaction Transfer Size - extended. This field holds the most significant bits of the transfer size, measured in bytes, of the DMA transaction that resulted in an error. |
| 15<br>DTYP | Data Type. The type of data being processed when the AXI transfer error occurred.<br>0b - message data<br>1b - control data |
| 14-12<br>JSRC | Job Source. The source of the job whose AXI transfer ended with an error:<br>000b - job ring 0<br>001b - job ring 1<br>010b - job ring 2<br>011b - job ring 3<br>100b - RTIC |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | 101b - Queue Interface<br>110b - reserved<br>111b - reserved |
| 11-8<br>BLKID | Block ID. The Block ID is the identifier of the block internal to SEC that initiated the DMA transfer that resulted in an error. BLKID is interpreted as follows:<br>0100b - job queue controller Burst Buffer<br>0101b - One of the job rings (see JSRC field)<br>0111b - Queue Interface<br>1000b - DECO0<br>1001b - DECO1<br>1010b - DECO2 |
| 7<br>TYP | AXI Transaction Type. This is the type, read or write, of the DMA transaction that resulted in an error.<br>0b - Read.<br>1b - Write. |
| 6-0<br>FSZ | AXI Transaction Transfer Size. This field holds the least-significant bits of the transfer size, measured in bytes, of the DMA transaction that resulted in an error. For large transfers the most-significant bits are held in field FSZ_EXT. |

# 13.119 SEC Status Register (SSTA)

## 13.119.1 Offset

| Register | Offset |
|---|---|
| SSTA | FD4h (alias) |
| SSTA | 1_0FD4h (alias) |
| SSTA | 2_0FD4h (alias) |
| SSTA | 3_0FD4h (alias) |
| SSTA | 4_0FD4h (alias) |
| SSTA | 6_0FD4h (alias) |
| SSTA | 7_0FD4h (alias) |
| SSTA | 8_0FD4h (alias) |
| SSTA | 9_0FD4h (alias) |
| SSTA | A_0FD4h (alias) |

## 13.119.2 Function

The SEC Status Register indicates some status information that is relevant to the entire SEC block. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

## 13.119.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | PLEND | MOO | | Reserved | | | | | TRNG_IDLE | IDLE | BSY |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

## 13.119.4 Fields

| Field | Function |
|---|---|
| 31-11<br>— | Reserved. |
| 10<br>PLEND | Platform Endianness. This is a hardwired SOC-specific configuration. PLEND indicates whether the SEC bus master views memory by default as big endian or little endian. Software can override the default for particular data by setting bits in the Job Ring Configuration Register, the RTIC Endian Register, the Queue Interface Control Register and the DECO Job Queue Control Register.<br>0b - Platform default is Little Endian<br>1b - Platform default is Big Endian |
| 9-8<br>MOO | Mode of Operation. These bits indicate the Security Mode that SEC is currently working in. The Security Mode is determined by the Security State Machine (see Security Monitor (SecMon) security states located in the Security Monitor. The modes are defined in SEC modes of operation.<br>00b - Non-Secure<br>01b - Secure<br>10b - Trusted<br>11b - Fail |
| 7-3<br>— | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 2<br>TRNG_IDLE | If TRNG_IDLE=1, the TRNG portion of the RNG is idle. The free-running oscillator is stopped, so no entropy is being generated. |
| 1<br>IDLE | SEC is idle. IDLE=1 indicates that SEC is not currently processing any jobs from QI, from job rings, or from RTIC and there are no pending interrupts (or the interrupts are masked) and the output job-ring timers are not counting. There may still be results in the output rings that have not been removed. If the RIDLE field in the RTIC Control Register=1, IDLE will be 0 if RTIC is in Run-Time Mode and one or more Memory Blocks are enabled for Run-Time Mode (i.e. one or more of the RTME bits is 1). If RIDLE=0 and SEC is otherwise idle, the IDLE will still occasionally be 0 while RTIC is actually hashing a chunk of memory. That is, if RTIC is in Run-Time Mode and one or more memory blocks is enabled, RTIC's Programmable DMA Throttle Timer may time out periodically and RTIC will launch a hashing job, which will cause IDLE to briefly go to 0. Software should always check the output rings and Output FQs for results prior to checking for IDLE. |
| 0<br>BSY | SEC Busy. BSY=1 indicates that SEC is processing at least one Descriptor. |

## 13.120 RTIC Version ID Register (RVID)

### 13.120.1 Offset

| Register | Offset |
|---|---|
| RVID | FE0h (alias) |
| RVID | 1_0FE0h (alias) |
| RVID | 2_0FE0h (alias) |
| RVID | 3_0FE0h (alias) |
| RVID | 4_0FE0h (alias) |
| RVID | 6_0FE0h (alias) |
| RVID | 7_0FE0h (alias) |
| RVID | 8_0FE0h (alias) |
| RVID | 9_0FE0h (alias) |
| RVID | A_0FE0h (alias) |

### 13.120.2 Function

The Run Time Integrity Checking Version ID register can be used by software to differentiate between different versions of the RTIC. Field RMJV is used for major revisions, field RMNV is used for minor revisions and the remaining fields are used for other revision information about the hardware. The bit assignments of this register appear below. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

## 13.120.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | MD | MC | MB | MA | Reserved | | | | SHA_512 | Reserved | SHA_256 | Reserved |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | RMJV | | | | | | | | RMNV | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

## 13.120.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved |
| 27<br>MD | Memory Block D Available. This bit indicates that Memory Block D is available for Hash Once and Run Time Checking. |
| 26<br>MC | Memory Block C Available. This bit indicates that Memory Block C is available for Hash Once and Run Time Checking. |
| 25<br>MB | Memory Block B Available. This bit indicates that Memory Block B is available for Hash Once and Run Time Checking. |
| 24<br>MA | Memory Block A Available. This bit indicates that Memory Block A is available for Hash Once and Run Time Checking. |
| 23-20<br>— | Reserved |
| 19<br>SHA_512 | SHA-512.<br>0b - RTIC cannot use the SHA-512 hashing algorithm.<br>1b - RTIC can use the SHA-512 hashing algorithm. |
| 18<br>— | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 17<br>SHA_256 | SHA-256.<br>0b - RTIC cannot use the SHA-256 hashing algorithm.<br>1b - RTIC can use the SHA-256 hashing algorithm. |
| 16<br>— | Reserved |
| 15-8<br>RMJV | RTIC Major Version. Represents major revision number of RTIC. This value is incremented when major functional changes are introduced or the programming model has changed. |
| 7-0<br>RMNV | RTIC Minor Version. Represents minor revision number of RTIC. This value is incremented when minor functional changes are made that do not change the programming model. Corrections that require changes to the design are the typical reason for incrementing these bits. |

## 13.121 CHA Cluster Block Version ID Register (CCBVID)

### 13.121.1 Offset

| Register | Offset |
|---|---|
| CCBVID | FE4h (alias) |
| CCBVID | 1_0FE4h (alias) |
| CCBVID | 2_0FE4h (alias) |
| CCBVID | 3_0FE4h (alias) |
| CCBVID | 4_0FE4h (alias) |
| CCBVID | 6_0FE4h (alias) |
| CCBVID | 7_0FE4h (alias) |
| CCBVID | 8_0FE4h (alias) |
| CCBVID | 9_0FE4h (alias) |
| CCBVID | A_0FE4h (alias) |

### 13.121.2 Function

The CHA Cluster Block Version ID register can be used by software to differentiate between different versions of the CCB. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

### 13.121.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SEC_ERA | | | | | | | | Reserved | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AMJV | | | | | | | | AMNV | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

### 13.121.4 Fields

| Field | Function |
|---|---|
| 31-24<br>SEC_ERA | SEC Era.<br>00000000b - This version of SEC is based on Era 5 or earlier RTL.<br>00000110b - This version of SEC is based on Era 6 RTL.<br>00000111b - This version of SEC is based on Era 7 RTL.<br>00001000b - This version of SEC is based on Era 8 RTL. |
| 23-16<br>— | Reserved |
| 15-8<br>AMJV | Accelerator Major Revision Number. This value will be incremented every time there is a major architectural change to the CCB design. Incrementing this results in the AMNV being set back to 0. |
| 7-0<br>AMNV | Accelerator Minor Revision Number. This value will be incremented every time an RTL change has been made to the CCB module. |

## 13.122 CHA Version ID Register, most-significant half (CHAVID_MS)

### 13.122.1 Offset

| Register | Offset |
|---|---|
| CHAVID_MS | FE8h (alias) |
| CHAVID_MS | 1_0FE8h (alias) |

*Table continues on the next page...*

| Register | Offset |
|---|---|
| CHAVID_MS | 2_0FE8h (alias) |
| CHAVID_MS | 3_0FE8h (alias) |
| CHAVID_MS | 4_0FE8h (alias) |
| CHAVID_MS | 6_0FE8h (alias) |
| CHAVID_MS | 7_0FE8h (alias) |
| CHAVID_MS | 8_0FE8h (alias) |
| CHAVID_MS | 9_0FE8h (alias) |
| CHAVID_MS | A_0FE8h (alias) |

## 13.122.2 Function

The CHA Version ID register can be used, along with the CCB Version ID, by software to differentiate between different versions of the cryptographic hardware accelerators. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

## 13.122.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JRVID | | | | DECOVID | | | | Reserved | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ZAVID | | | | ZEVID | | | | SNW9VID | | | | CRCVID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

## 13.122.4 Fields

| Field | Function |
|---|---|
| 31-28<br>JRVID | Job Ring Version ID |
| 27-24 | DECO Version ID |

*Table continues on the next page...*

| Field | Function |
|---|---|
| DECOVID | |
| 23-16<br>— | Reserved |
| 15-12<br>ZAVID | ZUC Authentication Hardware Accelerator Version ID |
| 11-8<br>ZEVID | ZUC Encryption Hardware Accelerator Version ID |
| 7-4<br>SNW9VID | SNOW-f9 Hardware Accelerator Version ID |
| 3-0<br>CRCVID | CRC Hardware Accelerator Version ID |

## 13.123 CHA Version ID Register, least-significant half (CHAVID_LS)

### 13.123.1 Offset

| Register | Offset |
|---|---|
| CHAVID_LS | FECh (alias) |
| CHAVID_LS | 1_0FECh (alias) |
| CHAVID_LS | 2_0FECh (alias) |
| CHAVID_LS | 3_0FECh (alias) |
| CHAVID_LS | 4_0FECh (alias) |
| CHAVID_LS | 6_0FECh (alias) |
| CHAVID_LS | 7_0FECh (alias) |
| CHAVID_LS | 8_0FECh (alias) |
| CHAVID_LS | 9_0FECh (alias) |
| CHAVID_LS | A_0FECh (alias) |

## 13.123.2 Function

The CHA Version ID register can be used, along with the CCB Version ID, by software to differentiate between different versions of the cryptographic hardware accelerators. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

## 13.123.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKVID | | | | KASVID | | | | SNW8VID | | | | RNGVID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MDVID | | | | Reserved | | | | DESVID | | | | AESVID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

## 13.123.4 Fields

| Field | Function |
|---|---|
| 31-28<br>PKVID | Public Key Hardware Accelerator Version ID<br>The bit count is the size of the digit used during computation. The single-digit ("SD") versions allow a minimum modulus size of one byte.<br>0000b - PKHA-XT (32-bit); minimum modulus five bytes<br>0001b - PKHA-SD (32-bit)<br>0010b - PKHA-SD (64-bit)<br>0011b - PKHA-SD (128-bit) |
| 27-24<br>KASVID | Kasumi f8/f9 Hardware Accelerator Version ID |
| 23-20<br>SNW8VID | SNOW-f8 Hardware Accelerator Version ID |
| 19-16<br>RNGVID | Random Number Generator Version ID.<br>0010b - RNGB<br>0100b - RNG4 |
| 15-12<br>MDVID | Message Digest Hardware Accelerator Version ID.<br>0000b - low-power MDHA, with SHA-1, SHA-256, SHA 224, MD5 and HMAC |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | 0001b - low-power MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-512/224, SHA-512/256, SHA-384, MD5 and HMAC<br>0010b - medium-performance MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-512/224, SHA-512/256, SHA-384, MD5, HMAC & SMAC<br>0011b - high-performance MDHA, with SHA-1, SHA-256, SHA 224, SHA-512, SHA-512/224, SHA-512/256, SHA-384, MD5, HMAC & SMAC |
| 11-8<br>— | Reserved |
| 7-4<br>DESVID | DES Accelerator Version ID. |
| 3-0<br>AESVID | AES Accelerator Version ID<br>0011b - low-power AESA, implementing ECB, CBC, CFB128, OFB, CTR, CCM, CMAC, XCBC-MAC, and GCM modes<br>0100b - high-performance AESA, implementing ECB, CBC, CFB128, OFB, CTR,CCM, CMAC, XCBC-MAC, CBCXCBC, CTRXCBC, XTS, and GCM modes |

# 13.124 CHA Number Register, most-significant half (CHANUM_MS)

## 13.124.1 Offset

| Register | Offset |
|---|---|
| CHANUM_MS | FF0h (alias) |
| CHANUM_MS | 1_0FF0h (alias) |
| CHANUM_MS | 2_0FF0h (alias) |
| CHANUM_MS | 3_0FF0h (alias) |
| CHANUM_MS | 4_0FF0h (alias) |
| CHANUM_MS | 6_0FF0h (alias) |
| CHANUM_MS | 7_0FF0h (alias) |
| CHANUM_MS | 8_0FF0h (alias) |
| CHANUM_MS | 9_0FF0h (alias) |
| CHANUM_MS | A_0FF0h (alias) |

### 13.124.2 Function

The CHA Number register can be used by software to determine how many copies of each type of cryptographic hardware accelerator are implemented in this version of SEC. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

### 13.124.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JRNUM | | | | DECONUM | | | | Reserved | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ZANUM | | | | ZENUM | | | | SNW9NUM | | | | CRCNUM | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

### 13.124.4 Fields

| Field | Function |
|---|---|
| 31-28<br>JRNUM | The number of copies of the job ring that are implemented in this version of SEC |
| 27-24<br>DECONUM | The number of copies of the DECO that are implemented in this version of SEC |
| 23-16<br>— | Reserved |
| 15-12<br>ZANUM | The number of copies of ZUCA that are implemented in this version of SEC |
| 11-8<br>ZENUM | The number of copies of ZUCE that are implemented in this version of SEC |
| 7-4<br>SNW9NUM | The number of copies of the SNOW-f9 module that are implemented in this version of SEC |
| 3-0<br>CRCNUM | The number of copies of the CRC module that are implemented in this version of SEC |

# 13.125 CHA Number Register, least-significant half (CHANUM_LS)

## 13.125.1 Offset

| Register | Offset |
|---|---|
| CHANUM_LS | FF4h (alias) |
| CHANUM_LS | 1_0FF4h (alias) |
| CHANUM_LS | 2_0FF4h (alias) |
| CHANUM_LS | 3_0FF4h (alias) |
| CHANUM_LS | 4_0FF4h (alias) |
| CHANUM_LS | 6_0FF4h (alias) |
| CHANUM_LS | 7_0FF4h (alias) |
| CHANUM_LS | 8_0FF4h (alias) |
| CHANUM_LS | 9_0FF4h (alias) |
| CHANUM_LS | A_0FF4h (alias) |

## 13.125.2 Function

The CHA Number register can be used by software to determine how many copies of each type of cryptographic hardware accelerator are implemented in this version of SEC. Since this register holds more than 32 bits, it is accessed as two 32-bit registers. Because this register may be of interest to multiple software entities, this register is aliased to addresses in multiple 64kbyte address spaces.

## 13.125.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PKNUM | | | | KASNUM | | | | SNW8NUM | | | | RNGNUM | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MDNUM | | | | ARC4NUM | | | | DESNUM | | | | AESNUM | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

### 13.125.4 Fields

| Field | Function |
|---|---|
| 31-28<br>PKNUM | The number of copies of the Public Key module that are implemented in this version of SEC |
| 27-24<br>KASNUM | The number of copies of the Kasumi module that are implemented in this version of SEC |
| 23-20<br>SNW8NUM | The number of copies of the SNOW-f8 module that are implemented in this version of SEC |
| 19-16<br>RNGNUM | The number of copies of the Random Number Generator that are implemented in this version of SEC. |
| 15-12<br>MDNUM | The number of copies of the MDHA (Hashing module) that are implemented in this version of SEC. |
| 11-8<br>ARC4NUM | The number of copies of the ARC4 module that are implemented in this version of SEC. |
| 7-4<br>DESNUM | The number of copies of the DES module that are implemented in this version of SEC. |
| 3-0<br>AESNUM | The number of copies of the AES module that are implemented in this version of SEC. |

## 13.126 Input Ring Base Address Register for Job Ring a (IRBAR_JR0 - IRBAR_JR3)

### 13.126.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| IRBAR_JRa | 1_0000h + (a × 1_0000h) | Used by JRa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R). |

## 13.126.2 Function

The Input Ring Base Address register holds the physical address of the input ring in memory (see Address pointers. SEC will use the number of address bits configured by the Pointer Size bit of the Master Configuration Register (MCFGR). Because there are 4 job rings, there are 4 copies of this register.

When the job ring is allocated to TrustZone SecureWorld, IRBAR may only be written with a transaction with ns=0. Also, the job ring must be started, if virtualization is enabled. The IRBAR register can be written only when there are no jobs in the input ring or when the job ring is halted, else an input ring base address or size invalid write error will result and a job ring reset or a power on reset will be required. Writing this register resets the Input Ring Read Index register, therefore following a write to the IRBAR the new head of the queue within the input ring will be located at the value just written to the IRBAR. Note that if the input ring was not empty, software must relocate the queue entries and write the number of these relocated entries to the Input Ring Jobs Added Register or these jobs will be lost. The address written to the Input Ring Base Address register must be 4-byte aligned, else an error will result and the job ring will not process jobs until a valid address is written and the error is cleared. More information on job management can be found in Job Ring interface.

## 13.126.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | IRBA | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | IRBA | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | IRBA | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.126.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved |
| 39-0<br>IRBA | Input Ring Base Address. |

## 13.127 Input Ring Size Register for Job Ring a (IRSR_JR0 - IRSR_JR3)

### 13.127.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| IRSR_JRa | 1_000Ch + (a × 1_0000h) | (used by JR a) |

### 13.127.2 Function

The Input Ring Size register holds the current size of the input ring, measured in number of entries. Note that each entry will be one word if 32-bit pointers are in use, but will be two words if pointers larger than 32-bits are in use. Because there are 4 job rings, there are 4 copies of this register.

When the job ring is allocated to TrustZone SecureWorld, IRSR may only be written with a transaction with ns=0. Also, the job ring must be started, if virtualization is enabled, in order to write the register. See Section Job Ring Registers. This register can be written only when there are no jobs in the input ring or when the job ring is halted, else an input ring base address or size invalid write error (type 5h) will result and a Job Ring reset or a power on reset will be required. Writing this register resets the Input Ring Read Index register, therefore following a write to the IRSR the new head of the queue within the input ring will be located at the value stored in the IRBAR. Note that if the input ring was not empty, software must relocate the queue entries and write the number of these relocated entries to the Input Ring Jobs Added Register or these jobs will be lost.

The size of the pointer entries in the ring is defined by the Pointer Size field of the (see Master Configuration Register (MCFGR). See Address pointers for a discussion of address pointers. More information on job management can be found in Job Ring interface.

### 13.127.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | IRS | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.127.4 Fields

| Field | Function |
|---|---|
| 31-10<br>— | Reserved. Must be 0. |
| 9-0<br>IRS | Input Ring Size. (measured in number of entries) |

## 13.128 Input Ring Slots Available Register for Job Ring a (IRSAR_JR0 - IRSAR_JR3)

### 13.128.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| IRSAR_JRa | 1_0014h + (a × 1_0000h) | (used by JR a) |

## 13.128.2 Function

The Input Ring Slots Available Register gives the number of empty slots for jobs in the input ring. Each slot is one word long if 32-bit pointers are in use, but is two words long if pointers larger than 32 bits are in use. Because there are 4 job rings, there are 4 copies of this register. This tells software how many more jobs it can submit to SEC before the input ring would be full. SEC increments this register when it removes a job from the input ring for processing. SEC decrements this register by the value in the Input Ring Jobs Added Register (see Section Input Ring Jobs Added Register for Job Ringa (IRJAR_JR0 - IRJAR_JR3)) when that register is updated. The value of the Input Ring Slots Available Register will never be larger than the Input Ring Size Register (see Section Input Ring Size Register for Job Ring a (IRSR_JR0 - IRSR_JR3)). More information on job management can be found in Job Ring interface.

The job ring must be started in order to write the IRSAR register. This register is read-only when virtualization is disabled. When the job ring is allocated to TrustZone SecureWorld, IRSAR may only be written with a transaction with ns=0. See Section Job Ring Registers.

## 13.128.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | IRSA | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.128.4 Fields

| Field | Function |
|---|---|
| 31-10<br>— | Reserved. Must be 0. |
| 9-0<br>IRSA | Input Ring Slots Available. (measured in number of available job slots) |

## 13.129 Input Ring Jobs Added Register for Job Ringa (IRJAR_JR0 - IRJAR_JR3)

### 13.129.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| IRJAR_JRa | 1_001Ch + (a × 1_0000h) | (used by JR a) |

### 13.129.2 Function

The Input Ring Jobs Added Register tells SEC how many new jobs were added to the input ring. Because there are 4 job rings, there are 4 copies of this register. Software must write into this register the number of Job Descriptor addresses that software has added to the ring. When the Input Ring Jobs Added Register is written, SEC adds that new value to its count of the jobs available for processing and decrements the Input Ring Slots Available Register. The value in the Input Ring Jobs Added Register must not be larger than the value of the Input Ring Slots Available Register (see Section Input Ring Slots Available Register for Job Ring a (IRSAR_JR0 - IRSAR_JR3)). If more jobs are added than the value in the Input Ring Slots Available Register an "Added too many jobs" error (type 9h) will occur. This is a fatal error and will require a job ring reset or power on reset to correct. More information on job management can be found in Job Ring interface.

When the job ring is allocated to TrustZone SecureWorld, IRJAR may only be written with a transaction with ns=0. If virtualization is enabled, the job ring must be started in order to write the register. See Section Job Ring Registers.

### 13.129.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | IRJA | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.129.4 Fields

| Field | Function |
|---|---|
| 31-10<br>— | Reserved. Must be 0. |
| 9-0<br>IRJA | Input Ring Jobs Added. (measured in number of entries) |

## 13.130 Output Ring Base Address Register for Job Ring a (ORBAR_JR0 - ORBAR_JR3)

### 13.130.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| ORBAR_JRa | 1_0020h + (a × 1_0000h) | Used by JRa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |

### 13.130.2 Function

The Output Ring Base Address Register holds the address of the output ring in memory (see Job Ring interface). SEC will use the number of bits configured by the Pointer Size bit of the Master Configuration Register (MCFGR). Because there are 4 job rings, there are 4 copies of this register. When the job ring is allocated to TrustZone SecureWorld, ORBAR may only be written with a transaction with ns=0. If virtualization is enabled, the job ring must be started in order to write the register. See Section Job Ring Registers. This register can be written only when the job ring is halted or when there are no jobs from this ring in progress within SEC or in the input ring or output ring, else an output ring base address or size invalid write error will result and a job ring reset, software SEC reset or a power on reset will be required.

Writing this register resets the Output Ring Write Index register, therefore following a write to the ORBAR the new tail of the queue within the output ring will be located at the value just written to the ORBAR. If the JR was halted before writing to the ORBAR, all jobs from that job ring will either still be in the input ring or will be completed and written to the output ring. This gives software a chance to process all completed jobs from the selected JR, and to query to see how many jobs are still in the input ring before writing the new output ring base address. This would allow for a clean start with a new empty output ring. Note that if the output ring was not empty at the time the ORBAR was written, those old results entries will not be in the new output ring. The address written to the Output Ring Base Address register must be 4-byte aligned, else an error will result and the job ring will not process jobs until a valid address is written and the error is cleared. More information on job management can be found in Job Ring interface.

### 13.130.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | ORBA | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ORBA | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ORBA | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.130.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved |
| 39-0<br>ORBA | Output Ring Base Address. |

## 13.131 Output Ring Size Register for Job Ring a (ORSR_JR0 - ORSR_JR3)

### 13.131.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| ORSR_JRa | 1_002Ch + (a × 1_0000h) | (used by JR a) |

## 13.131.2 Function

The Output Ring Size Register holds the current size of the output ring, measured in number of entries. Each entry in the output ring consists of one descriptor address pointer plus one 32-bit results status word, plus an optional word indicating the length of the SEQ sequence, if any, associated with this job (see INCL_SEQ_OUT field in the section Job Ring Configuration Register for Job Ring a, most-significant half (JRCFGR_JR0_MS - JRCFGR_JR3_MS)).The size of a descriptor pointer is defined by the Pointer Size bit of the Master Configuration Register (MCFGR). If PS=0, pointers are 32 bits. If PS=1, pointers are 40 bits. See Address pointers for a discussion of address pointers. Because there are 4 job rings, there are 4 copies of this register. If virtualization is enabled, the job ring must be started in order to write the register. See Section Job Ring Registers. This register can be written only when the job ring is halted or when there are no jobs from this ring in the input ring or output ring or in progress within SEC, else an *output ring base address or size invalid write error* will result and a job ring reset, software SEC reset or a power on reset will be required.

Writing this register resets the Output Ring Write Index register, therefore following a write to the ORSR the new tail of the queue within the output ring will be located at the value stored in the ORBAR. If the JR was halted before writing to the ORBAR, all jobs from that job ring will either still be in the input ring or will be completed and written to the output ring. This gives software a chance to process all completed jobs from the selected JR, and to query to see how many jobs are still in the input ring before writing the new output ring base address. This would allow for a clean start with a new empty output ring. Note that if the output ring was not empty at the time the ORSR was written, those old results entries will not be in the new output ring. If the output ring is not empty when the ORSR is written, software may need to process or relocate those entries to avoid losing job results.

More information on job management can be found in Job Ring interface.

### 13.131.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | ORS | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.131.4 Fields

| Field | Function |
|---|---|
| 31-10<br>— | Reserved. Must be 0. |
| 9-0<br>ORS | Output Ring Size. (measured in number of entries) |

## 13.132 Output Ring Jobs Removed Register for Job Ring a (ORJRR_JR0 - ORJRR_JR3)

### 13.132.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| ORJRR_JRa | 1_0034h + (a × 1_0000h) | (used by JR a) |

## 13.132.2 Function

The Output Ring Jobs Removed Register tells SEC how many jobs were removed from the output ring for processing by software. Because there are 4 job rings, there are 4 copies of this register. Software must write into this register the number of entries that software has removed from the ring. When the Output Ring Jobs Removed Register is written, SEC will subtract this amount from the Output Ring Slots Full Register (see Section Output Ring Slots Full Register for Job Ring a (ORSFR_JR0 - ORSFR_JR3)). The value of the Output Ring Jobs Removed Register must not be larger than the value in the Output Ring Slots Full Register. If a value larger than the Output Ring Slots Full Register is written to the ORJRR, a removed too many jobs error will occur and a job ring reset, software SEC reset or a power on reset will be required.

When the job ring is allocated to TrustZone SecureWorld, ORJRR may only be written with a transaction with ns=0. If virtualization is enabled, the job ring must be started in order to write the register. See Section Job Ring Registers.

More information on job management can be found in Job Ring interface.

## 13.132.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | ORJR | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.132.4 Fields

| Field | Function |
|---|---|
| 31-10<br>— | Reserved. Must be 0. |
| 9-0<br>ORJR | Output Ring Jobs Removed. (measured in number of entries) |

# 13.133 Output Ring Slots Full Register for Job Ring a (ORSFR_JR0 - ORSFR_JR3)

## 13.133.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| ORSFR_JRa | 1_003Ch + (a × 1_0000h) | (used by JR a) |

## 13.133.2 Function

The Output Ring Slots Full Register tells the software how many completed jobs SEC has placed in the output ring. Because there are 4 job rings, there are 4 copies of this register. SEC will increment this register as it completes a Descriptor and adds it to the output ring. SEC will decrement this register when software writes a new value to the Output Ring Jobs Removed Register (see Section Output Ring Jobs Removed Register for Job Ring a (ORJRR_JR0 - ORJRR_JR3)). The value in the Output Ring Slots Full Register cannot be larger than the value in the Output Ring Size Register (see Section Output Ring Size Register for Job Ring a (ORSR_JR0 - ORSR_JR3)).

The job ring must be started in order to write the IRSAR register. This register is read-only when virtualization is disabled. When the job ring is allocated to TrustZone SecureWorld, ORSFR may only be written with a transaction with ns=0. See Section Job Ring Registers.

More information on job management can be found in Job Ring interface.

### 13.133.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | ORSF | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.133.4 Fields

| Field | Function |
|---|---|
| 31-10<br>— | Reserved. Must be 0. |
| 9-0<br>ORSF | Output Ring Slots Full. (measured in number of entries) |

## 13.134 Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR3)

### 13.134.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| JRSTAR_JRa | 1_0044h + (a × 1_0000h) | (used by JR a) |

## 13.134.2 Function

This register is used to show the status of the last job that was completed. Because there are 4 job rings, there are 4 copies of this register. Although it is possible to read the job completion status directly from this register, in normal circumstances this is not useful because the status value will quickly be overwritten when the next job completes. Bits 0-31 of this register are written into the output ring after the completion of a job, and software should read the status from there. More information on job ring management can be found in Section Job Ring interface. Only one type of error will be valid at a time. The status code and various other information related to the status are given in the SSED field.

## 13.134.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SSRC | | | | SSED | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SSED | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.134.4 Fields

| Field | Function |
|---|---|
| 31-28<br>SSRC | Status source. These bits define which source is reporting the status.<br>All other values - reserved<br>0000b - No Status Source (No Error or Status Reported)<br>0001b - Reserved<br>0010b - CCB Status Source (CCB Error Reported)<br>0011b - Jump Halt User Status Source (User-Provided Status Reported)<br>0100b - DECO Status Source (DECO Error Reported)<br>0101b - QI Status Source (Queue Manager Interface Error Reported)<br>0110b - Job Ring Status Source (Job Ring Error Reported)<br>0111b - Jump Halt Condition Codes (Condition Code Status Reported) |
| 27-0<br>SSED | Source-specific error details. The format of this field depends on the status source specified in the SSRC field. The interpretation of the SSED field for all status sources is shown in Job termination status/error codes. |

# 13.135 Job Ring Interrupt Status Register for Job Ring a (JRINTR_JR0 - JRINTR_JR3)

## 13.135.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| JRINTR_JRa | 1_004Ch + (a × 1_0000h) | (used by JR a) |

## 13.135.2 Function

The Job Ring Interrupt Status Register indicates whether SEC has asserted an interrupt for a particular job ring, whether software has requested that the Job Ring be halted, whether the job ring is now halted, and whether there is an error in this job ring. If there was an error, the type of error is indicated. The error bit in the JRINT Register doesn't assert when there is a non-zero job completion status. It only asserts for the types of errors reported in the ERR_TYPE field in this register. Because there are 4 job rings, there are 4 copies of this register.

## 13.135.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | ERR_ORWI | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | ERR_TYPE | | | | | Reserved | | EXIT_FAIL | ENTER_FAIL | HALT | | JRE | JRI |
| W | | | | | | | | | | | W1C | W1C | W1C | | W1C | W1C |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.135.4 Fields

| Field | Function |
|---|---|
| 31-30<br>— | Reserved |
| 29-16<br>ERR_ORWI | Output ring write index with error. Set only when ERR_TYPE=0001. This indicates the location in the output ring that was being written when the error occurred. It is the offset in bytes from the Output Ring Base Address (see Section Output Ring Base Address Register for Job Ring a (ORBAR_JR0 - ORBAR_JR3)). |
| 15-13<br>— | Reserved |
| 12-8<br>ERR_TYPE | Error type. Set only when JRE bit is also set. Indicates the type of error when it cannot be reported in the Job Ring Status Register (see Section Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR3).)<br>00001b - Error writing status to Output Ring<br>00011b - Bad input ring base address (not on a 4-byte boundary).<br>00100b - Bad output ring base address (not on a 4-byte boundary).<br>00101b - Invalid write to Input Ring Base Address Register or Input Ring Size Register. Can be written when there are no jobs in the input ring or when the job ring is halted. These are fatal and will likely result in not being able to get all jobs out into the output ring for processing by software. Resetting the job ring will almost certainly be necessary.<br>00110b - Invalid write to Output Ring Base Address Register or Output Ring Size Register. Can be written when there are no jobs in the output ring and no jobs from this queue are already processing in SEC (in the holding tanks or DECOs), or when the job ring is halted.<br>00111b - Job ring reset released before job ring is halted.<br>01000b - Removed too many jobs (ORJRR larger than ORSFR). |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | 01001b - Added too many jobs (IRJAR larger than IRSAR).<br>01010b - Writing ORSF > ORS In these error cases the write is ignored, the interrupt is asserted (unless masked) and the error bit and error_type fields are set in the Job Ring Interrupt Status Register.<br>01011b - Writing IRSA > IRS<br>01100b - Writing ORWI > ORS in bytes<br>01101b - Writing IRRI > IRS in bytes<br>01110b - Writing IRSA when ring is active<br>01111b - Writing IRRI when ring is active<br>10000b - Writing ORSF when ring is active<br>10001b - Writing ORWI when ring is active |
| 7-6<br>— | Reserved |
| 5<br>EXIT_FAIL | Exit SecMon Fail State. This bit asserts when SecMon Fail State is exited. If the FAIL_MODE bit is set in the Job Ring Configuration register, the interrupt will also assert. Writing a 1 to the EXIT_FAIL bit will clear it. |
| 4<br>ENTER_FAIL | Enter SecMon Fail State. This bit asserts when SecMon Fail State is entered. If the FAIL_MODE bit is set in the Job Ring Configuration register, the interrupt will also assert. Writing a 1 to the ENTER_FAIL bit will clear it. |
| 3-2<br>HALT | Halt the job ring.<br>If reading HALT returns 01:<br>Software has requested that SEC flush the jobs in this job ring and halt processing jobs in this job ring (by writing to the RESET bit in the Job Ring Command register.).<br>If reading HALT returns 10:<br>SEC has flushed all jobs from this job ring and has halted processing jobs in this job ring. If there is not enough room in the output ring for all the flushed jobs, HALT will continue to return 01 until software has removed enough jobs so that all the flushed jobs can be written to the output ring.<br>Software writes a "1" to the MSB of HALT (bit 3) to clear the HALT field and resume processing jobs in this job ring. An error will occur if 1 is written to the MSB of the HALT field before the HALT field indicates that SEC has flushed all jobs from this job ring.<br>If SecMon indicates a FAIL MODE and the FAIL MODE bit is not set in the Job Ring Configuration register, a job ring halt will be initiated and the HALT status will return 01. When the halt process is complete, the HALT status will be 10. The HALT status cannot be cleared until SecMon transitions out of FAIL MODE. If the FAIL MODE bit is set in the Job Ring Configuration register, the job ring is not halted in FAIL MODE. |
| 1<br>JRE | Job Ring Error. A Job Ring error occurred. The error code is indicated in the ERR_TYPE field in this register. Write a 1 to this bit to clear the error indication. |
| 0<br>JRI | Job Ring Interrupt. SEC has asserted the interrupt request signal for this job ring. Write a 1 to this bit to clear the interrupt request. |

## 13.136 Job Ring Configuration Register for Job Ring a, most-significant half (JRCFGR_JR0_MS - JRCFGR_JR3_MS)

### 13.136.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| JRCFGR_JRa_MS | 1_0050h + (a × 1_0000h) | (used by JR a) |

### 13.136.2 Function

Software uses the Job Ring Configuration Register to configure the interrupt handling, error handling, and data endianness specific to a job ring. Because there are 4 job rings, there are 4 copies of this register. Since there are more than 32 bits in the JRCFG Register, it is accessed as two 32-bit words.

Note that many of the bits of this register are used to configure how data is rearranged when it is read from or written to memory. This is intended primarily to facilitate data handling in SoCs in which different processors use different data endianness. Because data may have to be rearranged differently depending upon the type of data, this register provides separate configuration bits for "control data" and for "message data". These are defined as shown below:

**Table 13-2. Control Data vs. Message Data**

| Control Data | Message Data |
|---|---|
| Control data read by SEC DMA:<br>• Descriptors or other data loaded into the Descriptor Buffer<br>• Job ring input ring entries<br>• Address pointers<br>• Scatter/Gather Tables<br>• Data loaded into the Class 1 or Class 2 Key Size registers<br>• Data loaded into the Class 1 or Class 2 Data Size registers<br>• Data loaded into the Class 1 or Class 2 ICV Size registers<br>• Data loaded into the DECO ICID register<br>• Data loaded into the CHA Control register<br>• Data loaded into the DECO Control register<br>• Data loaded into the IRQ Control register<br>• Data loaded into the DECO Protocol Override register<br>• Data loaded into the Clear Written register<br>• Data loaded into the Math registers<br>• Data loaded into the AAD Size register<br>• Data loaded into the Class 1 IV Size register<br>• Data loaded into the Alternate Data Size Class 1 register | Message data read by SEC DMA:<br>• Data read into the Input Data FIFO<br>• Data loaded into the Output Data FIFO<br>• Data loaded into the Class 1 or Class 2 Context registers<br>• Data loaded into the Class 1 or Class 2 Key registers<br>• Data loaded into the Input or Output Data FIFO Nibble Shift registers<br>• Data put into the Auxiliary Data FIFO |

*Table continues on the next page...*

**Table 13-2. Control Data vs. Message Data (continued)**

| Control Data | Message Data |
|---|---|
| • Data loaded into the PKHA Size registers<br>• Data loaded into the iNformation FIFO (NFIFO) | |
| Control data written by SEC DMA:<br>• Descriptors or other data stored from the Descriptor Buffer<br>• Job ring output ring entries<br>• Address pointers<br>• Scatter/Gather Tables<br>• Data stored from the Class 1 or Class 2 Mode registers<br>• Data stored from the DECO Job Queue Control register<br>• Data stored from the Class 1 or Class 2 Key Size registers<br>• Data stored from the DECO Descriptor Address Register<br>• Data stored from the Class 1 or Class 2 Data Size registers<br>• Data stored from the DECO Status register<br>• Data stored from the Class 1 or Class 2 ICV Size registers<br>• Data stored from the CHA Control register<br>• Data stored from the IRQ Control register<br>• Data stored from the Clear Written register<br>• Data stored from the Math registers<br>• Data stored from the CCB Status register<br>• Data stored from the AAD Size register<br>• Data stored from the Class 1 IV Size register<br>• Data stored from the PKHA Size registers | Message data written by SEC DMA:<br>• Data output via the Output Data FIFO<br>• Data stored from the Class 1 or Class 2 Context registers |

## 13.136.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | JDIS | INCL_SEQ_OUT | FAIL_MODE | Reserved | | | | | | | | | | DWSO | PEO | DMBS |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | CDWSO | CWSO | CHWSO | CBSO | MDWSO | MWSO | MHWSO | MBSO | CDWSI | CWSI | CHWSI | CBSI | MDWSI | MWSI | MHWSI | MBSI |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.136.4 Fields

| Field | Function |
|---|---|
| 31<br>JDIS | Job Descriptor ICID Select. Determines whether the SEQ ICID or the Non-SEQ ICID is asserted when reading the Job Descriptors addressed by the pointers stored in the job ring's input ring. The JDIS bit is also used to select the ICID when updating the same Job Descriptors with a STORE of type 41h. The JDIS bit does not apply to any Job Descriptors loaded during the operation of another Job Descriptor (e.g. loaded with a non-local JUMP command).<br>0b - Non-SEQ ICID<br>1b - SEQ ICID |
| 30<br>INCL_SEQ_OUT | Include Sequence Out Length. If this bit is set to 1, entries in the job ring's output ring will include a 32-bit word indicating the number of bytes written out via SEQ STORE and SEQ FIFO STORE commands in this job. If this bit is set to 0, the SEQ OUT Length is omitted from the entries. The setting of this bit can be changed only during ring configuration, when no jobs are running in SEC, else an error will be flagged. |
| 29<br>FAIL_MODE | Fail mode control. If this bit is set to 1 and SecMon indicates a FAIL MODE, the job ring will assert its interrupt and set the ENTER_FAIL bit in the Job Ring Interrupt Status register. The job ring will not halt, but will continue to process any available jobs. DECO will return these jobs will a FAIL MODE error. If SecMon transitions out of FAIL MODE, the job ring will assert its interrupt and set the EXIT_FAIL bit in the Job Ring Interrupt Status register.<br>If this bit is set to 0 and SecMon indicates a FAIL MODE, the job ring will set the ENTER_FAIL bit in the Job Ring Interrupt Status register. The job ring will halt until SecMon transitions out of FAIL MODE. When the job ring has halted, it will assert its interrupt. If SecMon transitions out of FAIL MODE, the job ring will set the EXIT_FAIL bit in the Job Ring Interrupt Status register. |
| 28-19<br>— | Reserved. Must be 0. |
| 18<br>DWSO | Double Word Swap Override. Setting DWSO=1 complements the swap control determined by MCFGR[DWT] and JRCFGR_JR[PEO] |
| 17<br>PEO | Platform Endian Override - The bit is XORed with the PLEND bit in the CaCSTA Register and the other "swap" bits in the Job Ring Configuration Register to determine the AXI Master's view of memory endianness when executing Job Descriptors from this job ring. Note that the swap bits can be used in combination to achieve multiple swaps simultaneously. |
| 16<br>DMBS | Descriptor Message Data Byte Swap (this applies only to internal message data transfers to/from DECO Descriptor Buffers). An example is shown below:<br>**Data as it is read from the source** — **0123456789abcdefh**<br>Data as it is written to the destination when DBMS = 0 — 0123456789abcdefh<br>Data as it is written to the destination when DBMS = 1 — 23016745ab89efcdh |
| 15<br>CDWSO | To assist with mixed Endianness platforms, this bit configures a doubleword swap of control data written by SEC DMA. An example is shown below:<br>**Data as interpreted within SEC** — **first word: 0001020304050607h** **second word: 08090a0b0c0d0e0fh**<br>Data as written to memory when PEO XOR PLEND XOR CDWSO = 0 — 000102030405060708090a0b0c0d0e0fh<br>Data as written to memory when PEO XOR PLEND XOR CDWSO = 1 — 08090a0b0c0d0e0f0001020304050607h |

*Table continues on the next page...*

**Job Ring Configuration Register for Job Ring a, most-significant half (JRCFGR_JR0_MS - JRCFGR_JR3_MS)**

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>14<br>CWSO</td><td>To assist with mixed Endianness platforms, this bit configures a fullword swap of control data written by SEC DMA. An example is shown below:
<table>
<tr><th>Data as interpreted within SEC</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR CWSO = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR CWSO = 1</td><td>89abcdef01234567h</td></tr>
</table></td></tr>
<tr><td>13<br>CHWSO</td><td>To assist with mixed Endianness platforms, this bit configures a halfword swap of control data written by SEC DMA. An example is shown below:
<table>
<tr><th>Data as interpreted within SEC</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR CHWSO = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR CHWSO = 1</td><td>45670123cdef89abh</td></tr>
</table></td></tr>
<tr><td>12<br>CBSO</td><td>To assist with mixed Endianness platforms, this bit configures a byte swap of control data written by SEC DMA. An example is shown below:
<table>
<tr><th>Data as interpreted within SEC</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR CBSO = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR CBSO = 1</td><td>23016745ab89efcdh</td></tr>
</table></td></tr>
<tr><td>11<br>MDWSO</td><td>To assist with mixed Endianness platforms, this bit configures a doubleword swap of message data written by SEC DMA. An example is shown below:
<table>
<tr><th>Data as interpreted within SEC</th><th>first word: 0001020304050607h<br>second word: 08090a0b0c0d0e0fh</th></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MDWSO = 0</td><td>000102030405060708090a0b0c0d0e0fh</td></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MDWSO = 1</td><td>08090a0b0c0d0e0f0001020304050607h</td></tr>
</table></td></tr>
<tr><td>10<br>MWSO</td><td>To assist with mixed Endianness platforms, this bit configures a fullword swap of message data written by SEC DMA. An example is shown below:
<table>
<tr><th>Data as interpreted within SEC</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MWSO = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MWSO = 1</td><td>89abcdef01234567h</td></tr>
</table></td></tr>
</table>

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>9<br>MHWSO</td><td>To assist with mixed Endianness platforms, this bit configures a halfword swap of message data written by SEC DMA. An example is shown below:
<table>
<tr><th>Data as interpreted within SEC</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MHWSO = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MHWSO = 1</td><td>45670123cdef89abh</td></tr>
</table></td></tr>
<tr><td>8<br>MBSO</td><td>To assist with mixed Endianness platforms, this bit configures a byte swap of message data written by SEC DMA. An example is shown below:
<table>
<tr><th>Data as interpreted within SEC</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MBSO = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MBSO = 1</td><td>23016745ab89efcdh</td></tr>
</table></td></tr>
<tr><td>7<br>CDWSI</td><td>To assist with mixed Endianness platforms, this bit configures a doubleword swap of control data read by SEC DMA. An example is shown below:
<table>
<tr><th>Data as stored in memory</th><th>000102030405060708090a0b0c0d0e0fh</th></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR CDWSI = 0</td><td>first word: 0001020304050607h<br>second word: 08090a0b0c0d0e0fh</td></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR CDWSI = 1</td><td>first word: 08090a0b0c0d0e0fh<br>second word: 0001020304050607h</td></tr>
</table></td></tr>
<tr><td>6<br>CWSI</td><td>To assist with mixed Endianness platforms, this bit configures a fullword swap of control data read by SEC DMA. An example is shown below:
<table>
<tr><th>Data as stored in memory</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR CWSI = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR CWSI = 1</td><td>89abcdef01234567h</td></tr>
</table></td></tr>
<tr><td>5<br>CHWSI</td><td>To assist with mixed Endianness platforms, this bit configures a halfword swap of control data read by SEC DMA. An example is shown below:
<table>
<tr><th>Data as stored in memory</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR CHWSI = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR CHWSI = 1</td><td>45670123cdef89abh</td></tr>
</table></td></tr>
</table>

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>4<br>CBSI</td><td>To assist with mixed Endianness platforms, this bit configures a byte swap of control data read by SEC DMA. An example is shown below:
<table>
<tr><th>Data as stored in memory</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR CBSI = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR CBSI = 1</td><td>23016745ab89efcdh</td></tr>
</table></td></tr>
<tr><td>3<br>MDWSI</td><td>To assist with mixed Endianness platforms, this bit configures a doubleword swap of message data read by SEC DMA. An example is shown below:
<table>
<tr><th>Data as stored in memory</th><th>000102030405060708090a0b0c0d0e0fh</th></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR MDWSI = 0</td><td>first word:<br>0001020304050607h<br>second word:<br>08090a0b0c0d0e0fh</td></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR MDWSI = 1</td><td>first word: 08090a0b0c0d0e0fh<br>second word: 0001020304050607h</td></tr>
</table></td></tr>
<tr><td>2<br>MWSI</td><td>To assist with mixed Endianness platforms, this bit configures a fullword swap of message data read by SEC DMA. An example is shown below:
<table>
<tr><th>Data as interpreted within SEC</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MWSI = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as written to memory when PEO XOR PLEND XOR MWSI = 1</td><td>89abcdef01234567h</td></tr>
</table></td></tr>
<tr><td>1<br>MHWSI</td><td>To assist with mixed Endianness platforms, this bit configures a halfword swap of message data read by SEC DMA. An example is shown below:
<table>
<tr><th>Data as stored in memory</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR MHWSI = 0</td><td>0123456789abcdefh</td></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR MHWSI = 1</td><td>45670123cdef89abh</td></tr>
</table></td></tr>
<tr><td>0<br>MBSI</td><td>To assist with mixed Endianness platforms, this bit configures a byte swap of message data read by SEC DMA. An example is shown below:
<table>
<tr><th>Data as stored in memory</th><th>0123456789abcdefh</th></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR MBSI = 0</td><td>01234567ababcdefh</td></tr>
<tr><td>Data as interpreted by SEC when PEO XOR PLEND XOR MBSI = 1</td><td>23016745ab89efcdh</td></tr>
</table></td></tr>
</table>

# 13.137 Job Ring Configuration Register for Job Ring a, least-significant half (JRCFGR_JR0_LS - JRCFGR_JR3_LS)

## 13.137.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| JRCFGR_JRa_LS | 1_0054h + (a × 1_0000h) | (used by JR a) |

## 13.137.2 Function

See description of JRCFGR_JR_MS

## 13.137.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ICTT | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ICDCT | | | | | | | | Reserved | | | | | | ICEN | IMSK |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.137.4 Fields

| Field | Function |
|---|---|
| 31-16<br>ICTT | Interrupt Coalescing Timer Threshold. While interrupt coalescing is enabled (ICEN=1), this value determines the maximum amount of time after processing a Descriptor before raising an interrupt. If Descriptors have been processed but the Descriptor count threshold has not been met, an interrupt is raised when the interrupt coalescing timer expires. The interrupt coalescing timer is stopped when the Output Ring Slots Full Register is 0. The timer is reset and stopped once an interrupt has been asserted |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | or whenever the Output Ring Jobs Removed Register is written by software. Counting resumes from zero after a reset if the counter is still enabled. The timer begins counting once the next Descriptor is moved to the output ring. Note that it is possible for one or more Descriptors to be moved to the output ring after software has read the Output Ring Slots Full Register and before software has written the Output Ring Jobs Removed Register. This would cause the timer to be reset to 0, but still counting. In this situation an interrupt would be raised when the timer expires at the full threshold value (unless the interrupt was raised earlier due to the Descriptor Count Threshold). The threshold value is represented in units equal to 64 SEC interface clocks. Valid values for this field are from 1 to 65535. A value of 0 results in behavior identical to that when interrupt coalescing is disabled. |
| 15-8<br>ICDCT | Interrupt Coalescing Descriptor Count Threshold. While interrupt coalescing is enabled (ICEN=1), this value determines how many Descriptors are completed before raising an interrupt. Valid values for this field are from 0 to 255. Note that a value of 1 functionally defeats the advantages of interrupt coalescing since the threshold value is reached each time that a Job Descriptor is completed. A value of 0 is treated in the same manner as a value of 1. The value of ICDCT is ignored if ICEN=0. |
| 7-2<br>— | Reserved |
| 1<br>ICEN | Interrupt Coalescing Enable.<br>0b - Interrupt coalescing is disabled. If the IMSK bit is cleared, an interrupt is asserted whenever a job is written to the output ring. ICDCT is ignored. Note that if software removes one or more jobs and clears the interrupt but the output rings slots full is still greater than 0 (ORSF > 0), then the interrupt will clear but reassert on the next clock cycle.<br>1b - Interrupt coalescing is enabled. If the IMSK bit is cleared, an interrupt is asserted whenever the threshold number of frames is reached (ICDCT) or when the threshold timer expires (ICTT). Note that if software removes one or more jobs and clears the interrupt but the interrupt coalescing threshold is still met (ORSF >= ICDCT), then the interrupt will clear but reassert on the next clock cycle. |
| 0<br>IMSK | Interrupt Mask. Mask the interrupt that is associated with the particular processor.<br>0b - Interrupt enabled.<br>1b - Interrupt masked. |

# 13.138 Input Ring Read Index Register for Job Ring a (IRRIR_JR0 - IRRIR_JR3)

## 13.138.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| IRRIR_JRa | 1_005Ch + (a × 1_0000h) | (used by JR a) |

## 13.138.2 Function

The Input Ring Read Index Register points to the head of the queue within the Input Ring. At this address there will be a pointer to the next Job Descriptor that SEC will fetch from this job ring. After SEC reads a Job Descriptor from the job ring SEC increments this register based on the Pointer Size bit of the Master Configuration Register described in Section Master Configuration Register (MCFGR). If PS=0 (pointers are 32 bits), the increment is 4. If PS=1 (pointers are 40 bits), the increment is 8. The index will be added to the Input Ring Base Address to get the physical address. Because there are 4 job rings, there are 4 copies of this register. More information on job management can be found in Job Ring interface.

The job ring must be started in order to write the IRRIR register. This register is read-only when virtualization is disabled. When the job ring is allocated to TrustZone SecureWorld, IRRIR may only be written with a transaction with ns=0. See Section Job Ring Registers.

## 13.138.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | IRRI | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.138.4 Fields

| Field | Function |
|---|---|
| 31-13<br>— | Reserved |
| 12-0<br>IRRI | Input Ring Read Index. |

# 13.139 Output Ring Write Index Register for Job Ring a (ORWIR_JR0 - ORWIR_JR3)

## 13.139.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| ORWIR_JRa | 1_0064h + (a × 1_0000h) | (used by JR a) |

## 13.139.2 Function

The Output Ring Write Index Register points to the tail of the queue within the output ring. Because there are 4 Job Rings, there are 4 copies of this register. The Output Ring Write Index Register is added to the Output Ring Base Address Register to get the physical address. At this address SEC writes a pointer to the last Descriptor that SEC has processed. At the next entry in the ring SEC writes the completion status of that Descriptor. Every time that a Descriptor has been processed SEC increments the value in the Output Ring Write Index Register by the size of the pointer, [1] plus the size of the 4-byte completion status word plus an additional 4 bytes if the INCL_SEQ_OUT bit in the JRCRGR is 1. So if INCL_SEQ_OUT=0 the increment will be 8 (if PS=0, i.e. 32-bit addresses) or 12 (if PS=1. i.e. 40-bit addresses). If INCL_SEQ_OUT=1, the increment will be 12 (if PS=0, i.e. 32-bit addresses) or 16 (if PS=1, i.e. 40-bit addresses). For a discussion of address pointers see Address pointers.

The job ring must be started in order to write the ORWIR register. This register is read-only when virtualization is disabled. When the job ring is allocated to TrustZone SecureWorld, ORWIR may only be written with a transaction with ns=0. See Section Job Ring Registers.

More information on job management can be found in Job Ring interface.

[1] The size of the pointer is defined by the Pointer Size bit of the Master Configuration Register described in Section Master Configuration Register (MCFGR).

### 13.139.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | ORWI | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.139.4 Fields

| Field | Function |
|---|---|
| 31-14<br>— | Reserved |
| 13-0<br>ORWI | Output Ring Write Index. The pointer to the next entry in the output ring. |

## 13.140 Job Ring Command Register for Job Ring a (JRCR_JR0 - JRCR_JR3)

### 13.140.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| JRCR_JRa | 1_006Ch + (a × 1_0000h) | (used by JR a) |

### 13.140.2 Function

Software can use this register to issue a park, flush or reset command to a job ring. A flush command is issued by writing a 1 to JRCR[RESET] when JRCR[RESET]=0. A flush is defined as stalling any jobs currently in the input ring and terminating (with an error code) any jobs currently in progress in the holding tanks or DECOs. The terminated jobs will be written to the output ring with a status indicating that they were terminated by a flush request. Note that these flushed jobs will count towards the Interrupt Coalescing Descriptor Count. If there is not sufficient space in the output ring for all the flushed jobs, job ring flushing will be paused until software has made enough space in the Output Ring. After a flush completes, the halt can be cleared and job processing will resume, or a reset can be requested.

A park command will stall any jobs in the job ring that have not yet been fetched, but will allow all the jobs in progress to complete normally. A park command may be issued only if virtualization is enabled. If virtualization is disabled, any writes to the PARK bit are ignored.

During the time between the write to PARK and all of the in-progress jobs completing, the HALT field in the Job Ring Interrupt Status register will return 01b indicating that the job ring was asked to stop processing jobs. When all jobs are complete and the job ring has halted, the Job Ring Interrupt Status register will indicate this by setting the HALT field to 10b. Once the job ring indicates that it has halted, it is safe to read the values of the job ring registers to save the job ring state. The following register values should be saved:

- JRCFGR_JR -Job Ring Configuration Register for the job ring
- IRBAR_JR - Input Ring Base Address Register for the job ring
- IRSR_JR - Input Ring Size Register for the job ring
- ORBAR_JR - Output Ring Base Address Register for the job ring
- ORSR_JR - Output Ring Size Register for the job ring
- IRSAR_JR - Input Ring Slots Available Register for the job ring
- ORSFR_JR - Output Ring Slots Full Register for the job ring
- IRRIR_JR - Input Ring Read Index Register for the job ring
- ORWIR_JR - Output Ring Write Index Register for the job ring

Once the state is saved, the job ring may be reassigned. To reassign the job ring, the registers that were saved should be rewritten with new values. When reprogramming, note that IRS must be written before IRSA or IRRI, and ORS must be written before ORSF or ORWI. IRSA should be written last because this is the register that indicates to the job ring that it has jobs to process. Failure to write the registers in the correct order may result in one of the following errors: IRSA>IRS, IRRI>IRS, ORSF>ORS, or ORWI>ORS. Once the job ring is reprogrammed, park status can be released so that the job ring can start running again. To do this, write a "1" to the MSB of the HALT field in the job ring Interrupt Status register. Note that if software tries to release parking status before the Job Ring has halted, a fatal error will occur (type 00111). This is the same error type as releasing the job ring from reset status before the ring has halted.

A reset command is issued by writing a 1 to JRCR[RESET] when JRCR[RESET]=1. A reset command will clear all registers in the job ring except the following:

- Input Ring Base Address
- Input Ring Size
- Output Ring Base Address
- Output Ring Size
- Job Ring Configuration.

A reset can be initiated only after a flush has been requested and completed as indicated by the HALT field in the Job Ring Interrupt Status Register. After a reset, job processing will resume when the Input Ring Jobs Added Register is written to indicate that new jobs are available. If both PARK and RESET are written to 1, the PARK is ignored and the job ring is reset.

Because there are 4 job rings, there are 4 copies of this register.

## 13.140.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | PARK | RESET |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.140.4 Fields

| Field | Function |
|---|---|
| 31-2 | Reserved. Always 0. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 1<br>PARK | Park - When PARK is 0, software writes a 1 to PARK to request that the job ring be "parked", i.e. quiesced. All jobs currently "in flight" (in holding tanks, DECOs or waiting for status results to be written out) are completed, but no new jobs are fetched from the input ring. When the job ring has completed parking, the HALT field in the Job Ring Interrupt Status register will change from 01b to 10b. |
| 0<br>RESET | Reset - When RESET is 0, software writes a 1 to RESET to request a flush of the Job Ring. If software wants to initiate a reset of the job ring, software writes a 1 to the RESET bit after a flush (when RESET is already 1). The reset will clear the RESET bit and other registers in the job ring. If no reset is required, software writes a 1 to the MSB of the HALT field in the Job Ring Interrupt Status Register to cause the job ring to resume processing jobs. An error will occur if 1 is written to the MSB of the HALT field before the HALT field indicates that the SEC has flushed all jobs from this job ring. |

## 13.141 Job Ring a Address-Array Valid Register (JR0AAV - JR3AAV)

### 13.141.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| JRaAAV | 1_0704h + (a × 1_0000h) | Used with Job Ring a |

### 13.141.2 Function

The Job Ring Address-Array Valid register indicates stored in the Job Ring Address-Array Address Registers. The register is intended to be used when debugging descriptor execution via a Job Ring. The Debug Control Register can be used to stop SEC processing before reading the job ring debug registers so that a consistent set of values can be read.

## 13.141.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | BC | Reserved | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | V7 | V6 | V5 | V4 | V3 | V2 | V1 | V0 |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.141.4 Fields

| Field | Function |
|---|---|
| 31<br>BC | Been Changed. The BC bit is used to verify that consistent data has been read from the Address Array Registers. BC is set to 0 when JR0AAA0 is read, and BC is then set to 1 if the content of any of the JRAAAx registers or the JRaAAVS register changes (due to new addresses being loaded into AA or existing addresses being sent to a holding tank) before the JRaAAVS is read. So if BC is 1 after this sequence of register reads, some of the data that was read may be inconsistent with other data that was read. In this case the address Array registers should be read again. |
| 30-8<br>— | Reserved |
| 7<br>V7 | Valid 7. When V7=1, Job Ring Address-Array Address Register 7 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers. |
| 6<br>V6 | Valid 6. When V6=1, Job Ring Address-Array Address Register 6 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers. |
| 5<br>V5 | Valid 5. When V5=1, Job Ring Address-Array Address Register 5 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers. |
| 4<br>V4 | Valid 4. When V4=1, Job Ring Address-Array Address Register 4 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers. |
| 3<br>V3 | Valid 3. When V3=1, Job Ring Address-Array Address Register 3 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 2<br>V2 | Valid 2. When V2=1, Job Ring Address-Array Address Register 2 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers. |
| 1<br>V1 | Valid 1. When V1=1, Job Ring Address-Array Address Register 1 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers. |
| 0<br>V0 | Valid 0. When V0=1, Job Ring Address-Array Address Register 0 contains a valid Job Descriptor address read from one of the Job Ring input rings. The valid bit is set when a Job Descriptor is read, and is cleared when the Job Descriptor is sent to a Holding Tank. Note that this version of SEC implements four Job Ring Address-Array Registers. |

## 13.142 Job Ring a Address-Array Address b Register (JR0AAA0 - JR3AAA7)

### 13.142.1 Offset

For a = 0 to 3; b = 0 to 7:

| Register | Offset | Description |
|---|---|---|
| JRaAAAb | 1_0800h + (a × 1_0000h) + (b × 8h) | Used with Job Ring a. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |

### 13.142.2 Function

The JRAAA registers are intended to be used when debugging descriptor execution via a job ring. The Debug Control Register can be used to stop SEC processing before reading the job ring debug registers so that a consistent set of values can be read. As discussed in Job scheduling, the job Queue Controller buffers up to four Job Descriptors from one job ring before servicing the next Job Ring in round-robin fashion. For performance reasons SEC reads multiple input ring entries whenever possible, so SEC may read up to four job descriptor addresses in a single bus burst. These registers store the job descriptor addresses after the job queue controller fetches the descriptor address from the input ring and before assigning the descriptor to a Holding Tank.

### 13.142.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | JD_ADDR | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JD_ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JD_ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.142.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved. |
| 39-0<br>JD_ADDR | Job Descriptor Address. |

## 13.143 Recoverable Error Indication Record 0 for Job Ring a (REIR0JR0 - REIR0JR3)

### 13.143.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| REIR0JRa | 1_0E00h + (a × 1_0000h) | (used by JR a) |

## 13.143.2 Function

If a recoverable error occurs related to execution of a job from a job ring, error information will be captured in the JR's REIR registers. Data for a second recoverable error related to jobs from JR will not be captured until the REIR0JR is written. If another bus error from JR occurs before then, the double error status bit (MISS) in REIR0JR will be set. When REIR0JR is written, all of JR's REIRJR registers are cleared and error capture is re-enabled.

## 13.143.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MISS | Reserved | | | | | TYPE | | Reserved | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.143.4 Fields

| Field | Function |
|---|---|
| 31<br>MISS | If MISS=1, a second recoverable error associated with JR occurred before REIR0JR was written following a previous JR recoverable error. |
| 30-26<br>— | Reserved |
| 25-24<br>TYPE | This field indicates the type of the recoverable error.<br>If TYPE = 0 : reserved<br>If TYPE = 1 : memory access error<br>If TYPE = 2 : reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | If TYPE = 3 : reserved |
| 23-0<br>— | Reserved |

# 13.144 Recoverable Error Indication Record 2 for Job Ring a (REIR2JR0 - REIR2JR3)

## 13.144.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| REIR2JRa | 1_0E08h + (a × 1_0000h) | Used by JRa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R). |

## 13.144.2 Function

See the description for Recoverable Error Indication Record 0 for Job Ring a (REIR0JR0 - REIR0JR3).

### 13.144.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.144.4 Fields

| Field | Function |
|---|---|
| 63-0<br>ADDR | Address associated with the recoverable JR error. |

## 13.145 Recoverable Error Indication Record 4 for Job Ring a (REIR4JR0 - REIR4JR3)

### 13.145.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| REIR4JRa | 1_0E10h + (a × 1_0000h) | (used by JR a) |

## 13.145.2 Function

See the description for Recoverable Error Indication Record 0 for Job Ring a (REIR0JR0 - REIR0JR3).

## 13.145.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MIX | | ERR | | Reserved | | | | RWB | AXPROT | | | AXCACHE | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | ICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.145.4 Fields

| Field | Function |
|---|---|
| 31-30<br>MIX | This field holds the memory interface index associated with the recoverable error. |
| 29-28<br>ERR | This field holds the AXI error response associated with the recoverable error. |
| 27-24<br>— | Reserved |
| 23<br>RWB | This field specifies whether the memory access was a read or write. |
| 22-20<br>AXPROT | This field holds the AXI protection transaction attribute used for the memory access. |
| 19-16<br>AXCACHE | This field holds the AXI cache control transaction attribute used for the memory access. |
| 15-12 | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 11-0<br>ICID | This field holds the ICID associated with the recoverable error.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

# 13.146 Recoverable Error Indication Record 5 for Job Ring a (REIR5JR0 - REIR5JR3)

## 13.146.1 Offset

For a = 0 to 3:

| Register | Offset | Description |
|---|---|---|
| REIR5JRa | 1_0E14h + (a × 1_0000h) | (used by JR a) |

## 13.146.2 Function

See the description for Recoverable Error Indication Record 0 for Job Ring a (REIR0JR0 - REIR0JR3).

## 13.146.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | SAFE | Reserved | | | | BID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.146.4 Fields

| Field | Function |
|---|---|
| 31-25<br>— | Reserved |
| 24<br>SAFE | For errors of REIR0JR[TYPE] = 00b SAFE indicates whether the AXI transaction associated with the recoverable error was a "safe" transaction. |
| 23-20<br>— | Reserved |
| 19-16<br>BID | This field holds the block identifier (see Table 13-1) of the source of the AXI transaction associated with the recoverable error. |
| 15-0<br>— | Reserved |

## 13.147 RTIC Status Register (RSTA)

### 13.147.1 Offset

| Register | Offset |
|---|---|
| RSTA | 6_0004h |

### 13.147.2 Function

This section describes the registers of the Run Time Integrity Checker (RTIC). A functional description of the RTIC can be found in Run-time integrity checker (RTIC). Note the use of the RTIC is optional, to support platform assurance.

The Run Time Integrity Checking Status Register is a read-only register that gives software information about the internal states of RTIC. Reading the RTIC Status Register will clear all errors and the RTIC interrupt. Due to timing issues, instead of polling this register software should read the RTIC Status Register after an RTIC done interrupt.

## 13.147.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | CS | | Reserved | | | | | RTD | HOD | ABH | WE |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | AE | | | | MIS | | | | HE | SV | HD | BSY |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.147.4 Fields

| Field | Function |
|---|---|
| 31-27<br>— | Reserved |
| 26-25<br>CS | RTIC Current State. Indicates the current state of the RTIC controller.<br>00b - Idle State<br>01b - Single Hash State<br>10b - Run-time State<br>11b - Error State |
| 24-20<br>— | Reserved |
| 19<br>RTD | Run Time Blocks Disabled. When RTIC is in Run Time mode, a 1 in the RTD field indicates that all the Memory Blocks are Disabled for Run Time Operation. |
| 18<br>HOD | Hash Once Blocks Disabled. All the Memory Blocks are Disabled for Hash Once Operation. This bit is set when RTIC is either in an Idle State or Hash Once State and none of the Memory Blocks have been enabled for Hash Once Operation. |
| 17<br>ABH | All Blocks Hashed. This is a bit that is used for debugging. This bit toggles during run-time mode every time RTIC completes hashing A-D memory blocks and starts over at the beginning again. |
| 16<br>WE | RTIC Watchdog Error. RTIC Watchdog timer has tripped during run-time hashing. This indicates that all enabled memory segments did not finish a round of hashing prior to the RTIC watchdog timer completing.<br>0b - No RTIC Watchdog timer error has occurred.<br>1b - RTIC Watchdog timer has expired prior to completing a round of hashing. |
| 15-12<br>— | Reserved |
| 11-8<br>AE | Address Error. Indicates an illegal address was read from a peripheral memory block. This is caused by an invalid start address in the Address 1/2 fields or a value in the Length 1/2 fields that caused the RTIC to read outside a peripheral memory's valid address space. If an address error occurs, the illegal address will be captured in the SEC Fault Address Register (Section Fault Address Register (FAR)). |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | Each bit in the field represents the status of an individual memory block. The following bit positions in the field indicates which memory block has the error:<br>xxx1 - Memory Block A Address Error<br>xx1x - Memory Block B Address Error<br>x1xx - Memory Block C Address Error<br>1xxx - Memory Block D Address Error<br>The settings for each bit are as follows:<br>0000b - All reads by RTIC were valid.<br>0001b - An illegal address was accessed by the RTIC |
| 7-4<br>MIS | Memory Integrity Status. Indicates memory block(s) with error. Each bit in the field represents the status of an individual memory block. The following bit positions in the field indicates which memory block has the error:<br>xxx1 - Memory Block A Hash Error<br>xx1x - Memory Block B Hash Error<br>x1xx - Memory Block C Hash Error<br>1xxx - Memory Block D Hash Error<br>The settings for each bit are as follows:<br>0000b - Memory Block X is valid or state unknown<br>0001b - Memory Block X has been corrupted |
| 3<br>HE | Hashing Error. Indicates that a unlocked memory block has been corrupted during run time or that an address error has occurred. The unlocked memory block(s) in error are indicated in the MIS field. If a memory addressing error occurred, the memory block(s) in error are indicated in the AE field. The security violation signal will be asserted. RTIC will generate a done interrupt and disable checking the memory block that caused the failure.<br>Unlocked memory blocks can be determined by reading the RTIC Control Register (see Section RTIC Control Register (RCTL)).<br>0b - Memory block contents authenticated.<br>1b - Memory block hash doesn't match reference value. |
| 2<br>SV | Security Violation. Indicates that a locked RTIC memory block has been corrupted during run-time, an address error has occurred, or an RTIC Watchdog timeout has occurred. The memory block(s) in error are indicated in the MIS field. If a memory addressing error occurred, the memory block(s) in error are indicated in the AE field. If an RTIC Watchdog timeout error occurred then the WE bit will be set. A security violation can only be cleared by a hardware reset.<br>Locked memory blocks can be determined by reading the RTIC Control Register (see Section RTIC Control Register (RCTL)).<br>0b - Memory block contents authenticated.<br>1b - Memory block hash doesn't match reference value. |
| 1<br>HD | Hash Once Operation Completed (Hash Done). processor may read hash values. If an error occurs during hashing or no memory blocks are enabled for one-time hash, this bit will not be set even if the RTIC hardware interrupts are asserted. This bit is cleared by setting the CINT bit in the RTIC Command Register (see Section RTIC Command Register (RCMD)) or when the RTIC enters the run-time checking state.<br>0b - Boot authentication disabled<br>1b - Authenticate code/generate reference hash value. This bit cannot be modified during run-time checking mode. |
| 0 | RTIC Idle/Busy Status. When busy, the RTIC cannot be written to. |

| Field | Function |
|---|---|
| BSY | 0b - RTIC Idle.<br>1b - RTIC Busy. |

# 13.148 RTIC Command Register (RCMD)

## 13.148.1 Offset

| Register | Offset |
|---|---|
| RCMD | 6_000Ch |

## 13.148.2 Function

The Run Time Integrity Checking Command Register is used to issue commands to the RTIC. This register is used to instruct the RTIC to perform different functions. This register is only writeable when RTIC is in an idle state.

## 13.148.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | RTD | | | |
| W | | | | | | | | | | | | | | RTC | HO | CINT |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.148.4 Fields

| Field | Function |
|---|---|
| 31-4<br>— | Reserved |
| 3<br>RTD | Run Time Disable. Does not allow RTIC to be put into Run-Time mode. This bit will have higher priority in the case where both Run Time Check and Run Time Disable are set on the same write. Run Time Disable is ignored if RTIC is already in the Run Time Mode.<br>0b - Allow Run Time Mode<br>1b - Prevent Run Time Mode |
| 2<br>RTC | Run time check. Starts run-time integrity checking for any blocks having the corresponding RTME bit =1 (see RTIC Status Register (RSTA)). Some of the RTIC registers become read-only. This bit is self-clearing and always returns a logic-0 when read. Setting this bit will clear the ipi_done_int hardware interrupt as well as the HASH DONE bit in the RTIC Status Register. Note that it is possible to set both the HO bit and the RTC bit to 1 simultaneously. In this case the hash-once operations will complete on all blocks whose HOME bit =1, and then the done interrupt will be asserted for one clock cycle but immediately cleared as RTIC enters Run-Time Check mode.<br>If no memory blocks are enabled, setting the RUN TIME CHK bit will cause the RTIC to enter an idle state while waiting for a memory segment to be enabled. Some registers will be read only. No data is hashed and no interrupts or errors will be generated.<br>0b - Run-time checking disabled<br>1b - Verify run-time memory blocks continually |
| 1<br>HO | Hash once. Starts one-time hash/boot code authentication for any blocks having the corresponding HOME bit =1 (see RTIC Status Register (RSTA)). The resulting hash value is stored in the Hash Register File. This bit is self-clearing and always returns a logic-0 when read. If no memory blocks are enabled, a done interrupt will be immediately generated. Note that it is possible to set both the HO bit and the RTC bit to 1 simultaneously. In this case the hash-once operations will complete on all blocks whose HOME bit =1, and then the done interrupt will be asserted for one clock cycle but immediately cleared as RTIC enters Run-Time Check mode.<br>0b - Boot authentication disabled<br>1b - Authenticate code/generate reference hash value. This bit cannot be modified during run-time checking mode. |
| 0<br>CINT | Clear Interrupt. Clears RTIC hardware interrupt signal. This bit is self-clearing and always returns a logic 0 when read.<br>0b - Do not clear interrupt<br>1b - Clear interrupt. This bit cannot be modified during run-time checking mode |

# 13.149 RTIC Control Register (RCTL)

## 13.149.1 Offset

| Register | Offset |
|---|---|
| RCTL | 6_0014h |

## 13.149.2 Function

The RTIC is configured by writing to the Run Time Integrity Checking Control Register. No bits in this register are writable unless RTIC is idle or, if RTIC is in Run-Time Mode, unless the control bits for the memory block are disabled and unlocked.

## 13.149.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | DSV | Reserved | | | | | DECOSEL | | Reserved | | | RIDLE | RALG | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RTMU | | | | RTME | | | | HOME | | | | RREQS | | | IE |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.149.4 Fields

| Field | Function |
|---|---|
| 31<br>DSV | DECOSEL Valid. If DSV=1, the DECOSEL field indicates the number of the DECO in which RTIC descriptors are to be run. This is intended for use during debugging. If DSV=0, the DECOSEL field is ignored and RTIC descriptors are run in any available DECO. |
| 30-26<br>— | Reserved |
| 25-24<br>DECOSEL | DECO Select. If DSV=1, DECOSEL indicates the number of the DECO in which RTIC descriptors are to be run. This is intended for use during debugging. If DSV=0, the DECOSEL field is ignored and RTIC descriptors are run in any available DECO.<br>DECO Select is interpreted as shown below. Note that the use of any value other than those listed will generate an error.<br>00b - run RTIC descriptors in DECO 0<br>01b - run RTIC descriptors in DECO 1<br>10b - run RTIC descriptors in DECO 2 |
| 23-21<br>— | Reserved |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>20<br>RIDLE</td><td>RTIC setting for the IPG_IDLE signal. If RIDLE=1, the signal ipg_idle will be negated if RTIC is in Run-Time Mode and one or more Memory Blocks are enabled for Run-Time Mode (i.e. one or more of the RTME bits is 1). If RIDLE=0 and SEC is otherwise idle, the signal ipg_idle will still occasionally negate while RTIC is actually hashing a chunk of memory.</td></tr>
<tr><td>19-16<br>RALG</td><td>RTIC Algorithm Select. Selects which algorithms should be used per Memory Block. All of these bits are writable when RTIC is idle. When RTIC is in Run-Time Mode, only those bits corresponding to unlocked memory blocks are writable. (see RTMU field description)
<table>
<tr><td>xxx0b - SHA-256 selected for Memory Block A</td><td>xxx1b - SHA-512 selected for Memory Block A</td></tr>
<tr><td>xx0xb - SHA-256 selected for Memory Block B</td><td>xx1xb - SHA-512 selected for Memory Block B</td></tr>
<tr><td>x0xxb - SHA-256 selected for Memory Block C</td><td>x1xxb - SHA-512 selected for Memory Block C</td></tr>
<tr><td>0xxxb - SHA-256 selected for Memory Block D</td><td>1xxxb - SHA-512 selected for Memory Block D</td></tr>
</table></td></tr>
<tr><td>15-12<br>RTMU</td><td>Run Time Memory Unlock. Unlocks memory block(s) for run-time hashing. If a memory block is unlocked it can be enabled and disabled at any time even if RTIC Run-Time Mode has started. These bits are not writable once RTIC Run-Time Mode has started. These bits are intended to allow some of the RTIC memory blocks to be used during RTIC Run-Time Mode by trusted software to verify the integrity of dynamically loaded software. The remaining (locked) memory blocks would be used to verify the integrity of the operating system and the trusted software itself.<br>xxx1b - Unlock Memory Block A<br>xx1xb - Unlock Memory Block B<br>x1xxb - Unlock Memory Block C<br>1xxxb - Unlock Memory Block D</td></tr>
<tr><td>11-8<br>RTME</td><td>Run Time Memory Enable. Enables memory block(s) for run-time hashing. All of these bits are writable when RTIC is idle. When RTIC is in Run-Time Mode, only those bits corresponding to unlocked memory blocks are writable. (see RTMU field description)<br>xxx1 - Enable Memory Block A<br>xx1x - Enable Memory Block B<br>x1xx - Enable Memory Block C<br>1xxx - Enable Memory Block D</td></tr>
<tr><td>7-4<br>HOME</td><td>Hash Once Memory Enable. Enables memory block(s) for one-time hashing. All of these bits are writable when RTIC is idle. When RTIC is in Run-Time Mode, only those bits corresponding to unlocked memory blocks are writable. (see RTMU field description)<br>xxx1 - Enable Memory Block A<br>xx1x - Enable Memory Block B<br>x1xx - Enable Memory Block C<br>1xxx - Enable Memory Block D</td></tr>
<tr><td>3-1<br>RREQS</td><td>RTIC Request Size. These bits are used during run-time mode to specify how many blocks of data are hashed every time the throttle counter expires. A block size is determined by the Algorithm that is selected.<br>Block Size:<br>SHA-256 = 64 bytes<br>SHA-512 = 128 bytes<br>Values:</td></tr>
</table>

*Table continues on the next page...*

| Field | Function |
|---|---|
| | 000b - 1 Block<br>001b - 1 Block<br>010b - 2 Blocks<br>011b - 3 Blocks<br>100b - 4 Blocks<br>101b - 5 Blocks<br>110b - 6 Blocks<br>111b - 7 Blocks |
| 0<br>IE | Interrupt Enable. Enables the RTIC interrupt. This bit is writable only while RTIC is in an idle state. Hardware interrupts are disabled by default after reset.<br>0b - Interrupts disabled<br>1b - Interrupts enabled |

## 13.150 RTIC Throttle Register (RTHR)

### 13.150.1 Offset

| Register | Offset |
|---|---|
| RTHR | 6_001Ch |

### 13.150.2 Function

The Run Time Integrity Checking Throttle Register can be set to specify how many clock cycles to wait between RTIC hashing operations when RTIC is in run-time mode. The register becomes read-only when RTIC is in run-time mode.

## 13.150.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RTHR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RTHR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.150.4 Fields

| Field | Function |
|---|---|
| 31-0<br>RTHR | Run Time Mode DMA Throttle. Programmable Timer that can be set to specify how many cycles of the system clock to wait between RTIC hashing operations during run time mode. At boot time, this register would generally be set to a value that will allow all four memory blocks to be hashed in a reasonable time without high bus utilization. |

# 13.151 RTIC Watchdog Timer (RWDOG)

## 13.151.1 Offset

| Register | Offset | Description |
|---|---|---|
| RWDOG | 6_0028h | When the endianness is in the default configuration, this address is for the most-significant 32 bits; the least-significant 32 bits can be accessed at this address +4h. |

## 13.151.2 Function

The RTIC Watchdog Register holds the starting value for the RTIC Watchdog Timer, which is used during Run Time Mode to prevent a denial of service attack on RTIC. When RTIC is in Run Time Mode, the RTIC Watchdog Timer begins counting down when run time hashing starts on the first memory block and it resets to the starting value when the last memory block has been hashed. If the RTIC Watchdog Timer times out prior to the last memory block's completion then an RTIC Watchdog error will be generated. Note that the RTIC Watchdog Register is not writable after RTIC enters Run Time Mode, so prior to placing RTIC into Run Time Mode software must write a large enough value into the register to prevent the RTIC Watchdog Timer from expiring under normal conditions. Upon entering low-power mode the RTIC Watchdog Timer will stop counting until low-power mode is exited. Upon exiting low-power mode, the RTIC Watchdog Timer will resume from where it left off.

## 13.151.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RWDOG | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RWDOG | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RWDOG | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.151.4 Fields

| Field | Function |
|---|---|
| 63-48<br>— | Reserved |
| 47-0<br>RWDOG | Run Time Watchdog Time-Out value. This holds the starting value of the RTIC Run Time Watchdog Timer. |

# 13.152 RTIC Endian Register (REND)

## 13.152.1 Offset

| Register | Offset |
|---|---|
| REND | 6_0034h |

## 13.152.2 Function

The RTIC Endian Register is used to allow for data ordering corrections when data is not retrieved from external memory in the proper order. These data ordering corrections are most likely to be needed on a mixed endian platform. The bit assignments of this register appear in the figure below and the description and settings for the register are given in the following table.

## 13.152.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | RDWS | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RWS | | | | RHWS | | | | RBS | | | | REPO | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.152.4 Fields

| Field | Function |
|---|---|
| 31-20<br>— | Reserved |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>19-16<br>RDWS</td><td>RTIC Double Word Swap. This allows for a software controllable double word swap in the DMA to assist with mixed Endianess platforms. This may be necessary if data coming into RTIC is double-word swapped when a 128 bit bus is used.<br>The memory blocks are configured as follows:<br>1xxxb - Double Word Swap Memory Block D<br>x1xxb - Double Word Swap Memory Block C<br>xx1xb - Double Word Swap Memory Block B<br>xxx1b - Double Word Swap Memory Block A</td></tr>
<tr><td>15-12<br>RWS</td><td>RTIC Word Swap. This allows for a software controllable word swap in the DMA to assist with mixed Endianess platforms. This may be necessary if data coming into RTIC is word swapped when a 64 bit or 128 bit bus is used.<br>The memory blocks are configured as follows:<br>1xxxb - Word Swap Memory Block D<br>x1xxb - Word Swap Memory Block C<br>xx1xb - Word Swap Memory Block B<br>xxx1b - Word Swap Memory Block A</td></tr>
<tr><td>11-8<br>RHWS</td><td>RTIC Half-Word Swap. This allows for a software controllable half-word swap in the DMA to assist with mixed Endianess platforms. This may be necessary if message data is not swapped properly when accessing memories. The word 01234567h placed in memory will become 45670123h when written into the hashing engine.<br>The memory blocks are configured as follows:<br>1xxxb - Half-Word Swap Memory Block D<br>x1xxb - Half-Word Swap Memory Block C<br>xx1xb - Half-Word Swap Memory Block B<br>xxx1b - Half-Word Swap Memory Block A</td></tr>
<tr><td>7-4<br>RBS</td><td>RTIC Byte Swap. This allows for a software controllable byte swap to assist with mixed Endianess platforms. This byte swap works in conjunction with the platform endian configuration indicated by the PLEND bit in the SEC Status Register. The word 01234567h placed in memory becomes 67452301h when written into the hashing engine.<br>The memory blocks are configured as follows:
<table>
<tr><th>Byte Swap Bit</th><th>PLEND</th><th>WORD</th></tr>
<tr><td>0</td><td>0</td><td>67452301h</td></tr>
<tr><td>1</td><td>0</td><td>01234567h</td></tr>
<tr><td>0</td><td>1</td><td>01234567h</td></tr>
<tr><td>1</td><td>1</td><td>67452301h</td></tr>
</table>
1xxxb - Byte Swap Memory Block D<br>x1xxb - Byte Swap Memory Block C<br>xx1xb - Byte Swap Memory Block B<br>xxx1b - Byte Swap Memory Block A</td></tr>
<tr><td>3-0<br>REPO</td><td>RTIC Endian Platform Override. This allows for the current platform endian configuration bit (PLEND bit in the SEC Status Register) to be overridden by bits in the REPO field. PLEND is either Big Endian =1 or Little Endian =0. Setting a REPO bit to 1 will cause the data read from the corresponding memory block to be interpreted as Big Endian if PLEND specifies Little Endian, or Little Endian if PLEND specifies Big Endian.<br>The memory blocks are configured as follows:</td></tr>
</table>

| Field | Function |
|---|---|
| | 1xxxb - Byte Swap Memory Block D<br>x1xxb - Byte Swap Memory Block C<br>xx1xb - Byte Swap Memory Block B<br>xxx1b - Byte Swap Memory Block A |

# 13.153 RTIC Memory Block a Address b Register (RMAA0 - RMDA1)

## 13.153.1 Offset

For a = A to D (0 to 3); b = 0 to 1:

| Register | Offset | Description |
|---|---|---|
| RMaAb | 6_0100h + (a × 20h) + (b × 10h) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R). |

## 13.153.2 Function

For an explanation of the RTIC Memory Block registers, see RTIC Memory Block Address/Length Registers

The RTIC Memory Block a Address b Register (RMaAb) specifies the starting address of segment b (b = 0 or 1) of Memory Block a (a = A,B,C,D). The length of data referred to by this pointer (see Address pointers.) is found in the RTIC Memory Block a Length b Register (RMaLb). The RTIC Memory Block Address registers and the RTIC Memory Block Length registers are writeable when RTIC is in an IDLE state, or during Run-Time mode if both the RTMU bit is set and the RTME bit is cleared (see Section RTIC Control Register (RCTL)) for the corresponding memory block.

### 13.153.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | MEMBLKADDR | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MEMBLKADDR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MEMBLKADDR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.153.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved. |
| 39-0<br>MEMBLKADDR | Memory Block Address. The MEMBLKADDR field of RMaAb holds the starting address of segment b (b=0,1) of RTIC Memory Block a (a=A,B,C,D). |

## 13.154 RTIC Memory Block a Length b Register (RMAL0 - RMDL1)

### 13.154.1 Offset

For a = A to D (0 to 3); b = 0 to 1:

| Register | Offset |
|---|---|
| RMaLb | 6_010Ch + (a × 20h) + (b × 10h) |

## 13.154.2 Function

For an explanation of the RTIC Memory Block registers, see RTIC Memory Block Address/Length Registers

The RTIC Memory Block a Length b Register (RMaLb) specifies the number of bytes to hash in segment b (b = 0 or 1) of Memory Block a (a = A,B,C,D). The starting address of segment b of RTIC Memory Block a is specified in the RTIC Memory Block a Address b Register (RMaAb). The RTIC Memory Block Address registers and the RTIC Memory Block Length registers are writeable when RTIC is in an IDLE state, or during Run-Time mode if both the RTMU bit is set and the RTME bit is cleared (see Section RTIC Control Register (RCTL)) for the corresponding memory block.

Note that programming a memory segment (A, B, C, D) to have a zero length (length_1 and length_2) will cause RTIC to generate a bad descriptor.

In RTIC versions RMJV= 0 and RMNV <=1 this can be detected by means of a watchdog timer. In hash-once operation this will be detected only if the DECO watchdog timer is enabled. This will cause the descriptor that is programmed by RTIC to be detected by the watchdog and flagged as an Address Error in the status register. In run-time operation the bad descriptor will be flagged by either the RTIC watchdog timer or the DECO watchdog timer. If the RTIC watchdog timer detects this condition then it will be flagged as an RTIC Watchdog Error. If instead the DECO watchdog catches it, then it will be flagged as an Address Error.

In later versions of RTIC bad RTIC descriptors will be flagged immediately as Address Errors.

### 13.154.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MEMBLKLEN | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MEMBLKLEN | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.154.4 Fields

| Field | Function |
|---|---|
| 31-0<br>MEMBLKLEN | Memory Block Lengths. The MEMBLKLEN field of RMaLb holds the length, in bytes, of segment b (b=0,1) of RTIC Memory Block a (a=A,B,C,D). |

## 13.155 RTIC Memory Block a c Endian Hash Result Word d (RAMDB_0 - RDMDL_31)

### 13.155.1 Offset

| Register | Offset | Description |
|---|---|---|
| RAMDB_0 | 6_0200h | RTIC Mem Block A Hash Result Big Endian Format Word 0 |
| RAMDB_1 | 6_0204h | RTIC Mem Block A Hash Result Big Endian Format Word 1 |
| RAMDB_2 | 6_0208h | RTIC Mem Block A Hash Result Big Endian Format Word 2 |
| RAMDB_3 | 6_020Ch | RTIC Mem Block A Hash Result Big Endian Format Word 3 |
| RAMDB_4 | 6_0210h | RTIC Mem Block A Hash Result Big Endian Format Word 4 |
| RAMDB_5 | 6_0214h | RTIC Mem Block A Hash Result Big Endian Format Word 5 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RAMDB_6 | 6_0218h | RTIC Mem Block A Hash Result Big Endian Format Word 6 |
| RAMDB_7 | 6_021Ch | RTIC Mem Block A Hash Result Big Endian Format Word 7 |
| RAMDB_8 | 6_0220h | RTIC Mem Block A Hash Result Big Endian Format Word 8 |
| RAMDB_9 | 6_0224h | RTIC Mem Block A Hash Result Big Endian Format Word 9 |
| RAMDB_10 | 6_0228h | RTIC Mem Block A Hash Result Big Endian Format Word 10 |
| RAMDB_11 | 6_022Ch | RTIC Mem Block A Hash Result Big Endian Format Word 11 |
| RAMDB_12 | 6_0230h | RTIC Mem Block A Hash Result Big Endian Format Word 12 |
| RAMDB_13 | 6_0234h | RTIC Mem Block A Hash Result Big Endian Format Word 13 |
| RAMDB_14 | 6_0238h | RTIC Mem Block A Hash Result Big Endian Format Word 14 |
| RAMDB_15 | 6_023Ch | RTIC Mem Block A Hash Result Big Endian Format Word 15 |
| RAMDB_16 | 6_0240h | RTIC Mem Block A Hash Result Big Endian Format Word 16 |
| RAMDB_17 | 6_0244h | RTIC Mem Block A Hash Result Big Endian Format Word 17 |
| RAMDB_18 | 6_0248h | RTIC Mem Block A Hash Result Big Endian Format Word 18 |
| RAMDB_19 | 6_024Ch | RTIC Mem Block A Hash Result Big Endian Format Word 19 |
| RAMDB_20 | 6_0250h | RTIC Mem Block A Hash Result Big Endian Format Word 20 |
| RAMDB_21 | 6_0254h | RTIC Mem Block A Hash Result Big Endian Format Word 21 |
| RAMDB_22 | 6_0258h | RTIC Mem Block A Hash Result Big Endian Format Word 22 |
| RAMDB_23 | 6_025Ch | RTIC Mem Block A Hash Result Big Endian Format Word 23 |
| RAMDB_24 | 6_0260h | RTIC Mem Block A Hash Result Big Endian Format Word 24 |
| RAMDB_25 | 6_0264h | RTIC Mem Block A Hash Result Big Endian Format Word 25 |
| RAMDB_26 | 6_0268h | RTIC Mem Block A Hash Result Big Endian Format Word 26 |
| RAMDB_27 | 6_026Ch | RTIC Mem Block A Hash Result Big Endian Format Word 27 |
| RAMDB_28 | 6_0270h | RTIC Mem Block A Hash Result Big Endian Format Word 28 |
| RAMDB_29 | 6_0274h | RTIC Mem Block A Hash Result Big Endian Format Word 29 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RAMDB_30 | 6_0278h | RTIC Mem Block A Hash Result Big Endian Format Word 30 |
| RAMDB_31 | 6_027Ch | RTIC Mem Block A Hash Result Big Endian Format Word 31 |
| RAMDL_0 | 6_0280h | RTIC Mem Block A Hash Result Little Endian Format Word 0 |
| RAMDL_1 | 6_0284h | RTIC Mem Block A Hash Result Little Endian Format Word 1 |
| RAMDL_2 | 6_0288h | RTIC Mem Block A Hash Result Little Endian Format Word 2 |
| RAMDL_3 | 6_028Ch | RTIC Mem Block A Hash Result Little Endian Format Word 3 |
| RAMDL_4 | 6_0290h | RTIC Mem Block A Hash Result Little Endian Format Word 4 |
| RAMDL_5 | 6_0294h | RTIC Mem Block A Hash Result Little Endian Format Word 5 |
| RAMDL_6 | 6_0298h | RTIC Mem Block A Hash Result Little Endian Format Word 6 |
| RAMDL_7 | 6_029Ch | RTIC Mem Block A Hash Result Little Endian Format Word 7 |
| RAMDL_8 | 6_02A0h | RTIC Mem Block A Hash Result Little Endian Format Word 8 |
| RAMDL_9 | 6_02A4h | RTIC Mem Block A Hash Result Little Endian Format Word 9 |
| RAMDL_10 | 6_02A8h | RTIC Mem Block A Hash Result Little Endian Format Word 10 |
| RAMDL_11 | 6_02ACh | RTIC Mem Block A Hash Result Little Endian Format Word 11 |
| RAMDL_12 | 6_02B0h | RTIC Mem Block A Hash Result Little Endian Format Word 12 |
| RAMDL_13 | 6_02B4h | RTIC Mem Block A Hash Result Little Endian Format Word 13 |
| RAMDL_14 | 6_02B8h | RTIC Mem Block A Hash Result Little Endian Format Word 14 |
| RAMDL_15 | 6_02BCh | RTIC Mem Block A Hash Result Little Endian Format Word 15 |
| RAMDL_16 | 6_02C0h | RTIC Mem Block A Hash Result Little Endian Format Word 16 |
| RAMDL_17 | 6_02C4h | RTIC Mem Block A Hash Result Little Endian Format Word 17 |
| RAMDL_18 | 6_02C8h | RTIC Mem Block A Hash Result Little Endian Format Word 18 |
| RAMDL_19 | 6_02CCh | RTIC Mem Block A Hash Result Little Endian Format Word 19 |
| RAMDL_20 | 6_02D0h | RTIC Mem Block A Hash Result Little Endian Format Word 20 |
| RAMDL_21 | 6_02D4h | RTIC Mem Block A Hash Result Little Endian Format Word 21 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RAMDL_22 | 6_02D8h | RTIC Mem Block A Hash Result Little Endian Format Word 22 |
| RAMDL_23 | 6_02DCh | RTIC Mem Block A Hash Result Little Endian Format Word 23 |
| RAMDL_24 | 6_02E0h | RTIC Mem Block A Hash Result Little Endian Format Word 24 |
| RAMDL_25 | 6_02E4h | RTIC Mem Block A Hash Result Little Endian Format Word 25 |
| RAMDL_26 | 6_02E8h | RTIC Mem Block A Hash Result Little Endian Format Word 26 |
| RAMDL_27 | 6_02ECh | RTIC Mem Block A Hash Result Little Endian Format Word 27 |
| RAMDL_28 | 6_02F0h | RTIC Mem Block A Hash Result Little Endian Format Word 28 |
| RAMDL_29 | 6_02F4h | RTIC Mem Block A Hash Result Little Endian Format Word 29 |
| RAMDL_30 | 6_02F8h | RTIC Mem Block A Hash Result Little Endian Format Word 30 |
| RAMDL_31 | 6_02FCh | RTIC Mem Block A Hash Result Little Endian Format Word 31 |
| RBMDB_0 | 6_0300h | RTIC Mem Block B Hash Result Big Endian Format Word 0 |
| RBMDB_1 | 6_0304h | RTIC Mem Block B Hash Result Big Endian Format Word 1 |
| RBMDB_2 | 6_0308h | RTIC Mem Block B Hash Result Big Endian Format Word 2 |
| RBMDB_3 | 6_030Ch | RTIC Mem Block B Hash Result Big Endian Format Word 3 |
| RBMDB_4 | 6_0310h | RTIC Mem Block B Hash Result Big Endian Format Word 4 |
| RBMDB_5 | 6_0314h | RTIC Mem Block B Hash Result Big Endian Format Word 5 |
| RBMDB_6 | 6_0318h | RTIC Mem Block B Hash Result Big Endian Format Word 6 |
| RBMDB_7 | 6_031Ch | RTIC Mem Block B Hash Result Big Endian Format Word 7 |
| RBMDB_8 | 6_0320h | RTIC Mem Block B Hash Result Big Endian Format Word 8 |
| RBMDB_9 | 6_0324h | RTIC Mem Block B Hash Result Big Endian Format Word 9 |
| RBMDB_10 | 6_0328h | RTIC Mem Block B Hash Result Big Endian Format Word 10 |
| RBMDB_11 | 6_032Ch | RTIC Mem Block B Hash Result Big Endian Format Word 11 |
| RBMDB_12 | 6_0330h | RTIC Mem Block B Hash Result Big Endian Format Word 12 |
| RBMDB_13 | 6_0334h | RTIC Mem Block B Hash Result Big Endian Format Word 13 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RBMDB_14 | 6_0338h | RTIC Mem Block B Hash Result Big Endian Format Word 14 |
| RBMDB_15 | 6_033Ch | RTIC Mem Block B Hash Result Big Endian Format Word 15 |
| RBMDB_16 | 6_0340h | RTIC Mem Block B Hash Result Big Endian Format Word 16 |
| RBMDB_17 | 6_0344h | RTIC Mem Block B Hash Result Big Endian Format Word 17 |
| RBMDB_18 | 6_0348h | RTIC Mem Block B Hash Result Big Endian Format Word 18 |
| RBMDB_19 | 6_034Ch | RTIC Mem Block B Hash Result Big Endian Format Word 19 |
| RBMDB_20 | 6_0350h | RTIC Mem Block B Hash Result Big Endian Format Word 20 |
| RBMDB_21 | 6_0354h | RTIC Mem Block B Hash Result Big Endian Format Word 21 |
| RBMDB_22 | 6_0358h | RTIC Mem Block B Hash Result Big Endian Format Word 22 |
| RBMDB_23 | 6_035Ch | RTIC Mem Block B Hash Result Big Endian Format Word 23 |
| RBMDB_24 | 6_0360h | RTIC Mem Block B Hash Result Big Endian Format Word 24 |
| RBMDB_25 | 6_0364h | RTIC Mem Block B Hash Result Big Endian Format Word 25 |
| RBMDB_26 | 6_0368h | RTIC Mem Block B Hash Result Big Endian Format Word 26 |
| RBMDB_27 | 6_036Ch | RTIC Mem Block B Hash Result Big Endian Format Word 27 |
| RBMDB_28 | 6_0370h | RTIC Mem Block B Hash Result Big Endian Format Word 28 |
| RBMDB_29 | 6_0374h | RTIC Mem Block B Hash Result Big Endian Format Word 29 |
| RBMDB_30 | 6_0378h | RTIC Mem Block B Hash Result Big Endian Format Word 30 |
| RBMDB_31 | 6_037Ch | RTIC Mem Block B Hash Result Big Endian Format Word 31 |
| RBMDL_0 | 6_0380h | RTIC Mem Block B Hash Result Little Endian Format Word 0 |
| RBMDL_1 | 6_0384h | RTIC Mem Block B Hash Result Little Endian Format Word 1 |
| RBMDL_2 | 6_0388h | RTIC Mem Block B Hash Result Little Endian Format Word 2 |
| RBMDL_3 | 6_038Ch | RTIC Mem Block B Hash Result Little Endian Format Word 3 |
| RBMDL_4 | 6_0390h | RTIC Mem Block B Hash Result Little Endian Format Word 4 |
| RBMDL_5 | 6_0394h | RTIC Mem Block B Hash Result Little Endian Format Word 5 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RBMDL_6 | 6_0398h | RTIC Mem Block B Hash Result Little Endian Format Word 6 |
| RBMDL_7 | 6_039Ch | RTIC Mem Block B Hash Result Little Endian Format Word 7 |
| RBMDL_8 | 6_03A0h | RTIC Mem Block B Hash Result Little Endian Format Word 8 |
| RBMDL_9 | 6_03A4h | RTIC Mem Block B Hash Result Little Endian Format Word 9 |
| RBMDL_10 | 6_03A8h | RTIC Mem Block B Hash Result Little Endian Format Word 10 |
| RBMDL_11 | 6_03ACh | RTIC Mem Block B Hash Result Little Endian Format Word 11 |
| RBMDL_12 | 6_03B0h | RTIC Mem Block B Hash Result Little Endian Format Word 12 |
| RBMDL_13 | 6_03B4h | RTIC Mem Block B Hash Result Little Endian Format Word 13 |
| RBMDL_14 | 6_03B8h | RTIC Mem Block B Hash Result Little Endian Format Word 14 |
| RBMDL_15 | 6_03BCh | RTIC Mem Block B Hash Result Little Endian Format Word 15 |
| RBMDL_16 | 6_03C0h | RTIC Mem Block B Hash Result Little Endian Format Word 16 |
| RBMDL_17 | 6_03C4h | RTIC Mem Block B Hash Result Little Endian Format Word 17 |
| RBMDL_18 | 6_03C8h | RTIC Mem Block B Hash Result Little Endian Format Word 18 |
| RBMDL_19 | 6_03CCh | RTIC Mem Block B Hash Result Little Endian Format Word 19 |
| RBMDL_20 | 6_03D0h | RTIC Mem Block B Hash Result Little Endian Format Word 20 |
| RBMDL_21 | 6_03D4h | RTIC Mem Block B Hash Result Little Endian Format Word 21 |
| RBMDL_22 | 6_03D8h | RTIC Mem Block B Hash Result Little Endian Format Word 22 |
| RBMDL_23 | 6_03DCh | RTIC Mem Block B Hash Result Little Endian Format Word 23 |
| RBMDL_24 | 6_03E0h | RTIC Mem Block B Hash Result Little Endian Format Word 24 |
| RBMDL_25 | 6_03E4h | RTIC Mem Block B Hash Result Little Endian Format Word 25 |
| RBMDL_26 | 6_03E8h | RTIC Mem Block B Hash Result Little Endian Format Word 26 |
| RBMDL_27 | 6_03ECh | RTIC Mem Block B Hash Result Little Endian Format Word 27 |
| RBMDL_28 | 6_03F0h | RTIC Mem Block B Hash Result Little Endian Format Word 28 |
| RBMDL_29 | 6_03F4h | RTIC Mem Block B Hash Result Little Endian Format Word 29 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RBMDL_30 | 6_03F8h | RTIC Mem Block B Hash Result Little Endian Format Word 30 |
| RBMDL_31 | 6_03FCh | RTIC Mem Block B Hash Result Little Endian Format Word 31 |
| RCMDB_0 | 6_0400h | RTIC Mem Block C Hash Result Big Endian Format Word 0 |
| RCMDB_1 | 6_0404h | RTIC Mem Block C Hash Result Big Endian Format Word 1 |
| RCMDB_2 | 6_0408h | RTIC Mem Block C Hash Result Big Endian Format Word 2 |
| RCMDB_3 | 6_040Ch | RTIC Mem Block C Hash Result Big Endian Format Word 3 |
| RCMDB_4 | 6_0410h | RTIC Mem Block C Hash Result Big Endian Format Word 4 |
| RCMDB_5 | 6_0414h | RTIC Mem Block C Hash Result Big Endian Format Word 5 |
| RCMDB_6 | 6_0418h | RTIC Mem Block C Hash Result Big Endian Format Word 6 |
| RCMDB_7 | 6_041Ch | RTIC Mem Block C Hash Result Big Endian Format Word 7 |
| RCMDB_8 | 6_0420h | RTIC Mem Block C Hash Result Big Endian Format Word 8 |
| RCMDB_9 | 6_0424h | RTIC Mem Block C Hash Result Big Endian Format Word 9 |
| RCMDB_10 | 6_0428h | RTIC Mem Block C Hash Result Big Endian Format Word 10 |
| RCMDB_11 | 6_042Ch | RTIC Mem Block C Hash Result Big Endian Format Word 11 |
| RCMDB_12 | 6_0430h | RTIC Mem Block C Hash Result Big Endian Format Word 12 |
| RCMDB_13 | 6_0434h | RTIC Mem Block C Hash Result Big Endian Format Word 13 |
| RCMDB_14 | 6_0438h | RTIC Mem Block C Hash Result Big Endian Format Word 14 |
| RCMDB_15 | 6_043Ch | RTIC Mem Block C Hash Result Big Endian Format Word 15 |
| RCMDB_16 | 6_0440h | RTIC Mem Block C Hash Result Big Endian Format Word 16 |
| RCMDB_17 | 6_0444h | RTIC Mem Block C Hash Result Big Endian Format Word 17 |
| RCMDB_18 | 6_0448h | RTIC Mem Block C Hash Result Big Endian Format Word 18 |
| RCMDB_19 | 6_044Ch | RTIC Mem Block C Hash Result Big Endian Format Word 19 |
| RCMDB_20 | 6_0450h | RTIC Mem Block C Hash Result Big Endian Format Word 20 |
| RCMDB_21 | 6_0454h | RTIC Mem Block C Hash Result Big Endian Format Word 21 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RCMDB_22 | 6_0458h | RTIC Mem Block C Hash Result Big Endian Format Word 22 |
| RCMDB_23 | 6_045Ch | RTIC Mem Block C Hash Result Big Endian Format Word 23 |
| RCMDB_24 | 6_0460h | RTIC Mem Block C Hash Result Big Endian Format Word 24 |
| RCMDB_25 | 6_0464h | RTIC Mem Block C Hash Result Big Endian Format Word 25 |
| RCMDB_26 | 6_0468h | RTIC Mem Block C Hash Result Big Endian Format Word 26 |
| RCMDB_27 | 6_046Ch | RTIC Mem Block C Hash Result Big Endian Format Word 27 |
| RCMDB_28 | 6_0470h | RTIC Mem Block C Hash Result Big Endian Format Word 28 |
| RCMDB_29 | 6_0474h | RTIC Mem Block C Hash Result Big Endian Format Word 29 |
| RCMDB_30 | 6_0478h | RTIC Mem Block C Hash Result Big Endian Format Word 30 |
| RCMDB_31 | 6_047Ch | RTIC Mem Block C Hash Result Big Endian Format Word 31 |
| RCMDL_0 | 6_0480h | RTIC Mem Block C Hash Result Little Endian Format Word 0 |
| RCMDL_1 | 6_0484h | RTIC Mem Block C Hash Result Little Endian Format Word 1 |
| RCMDL_2 | 6_0488h | RTIC Mem Block C Hash Result Little Endian Format Word 2 |
| RCMDL_3 | 6_048Ch | RTIC Mem Block C Hash Result Little Endian Format Word 3 |
| RCMDL_4 | 6_0490h | RTIC Mem Block C Hash Result Little Endian Format Word 4 |
| RCMDL_5 | 6_0494h | RTIC Mem Block C Hash Result Little Endian Format Word 5 |
| RCMDL_6 | 6_0498h | RTIC Mem Block C Hash Result Little Endian Format Word 6 |
| RCMDL_7 | 6_049Ch | RTIC Mem Block C Hash Result Little Endian Format Word 7 |
| RCMDL_8 | 6_04A0h | RTIC Mem Block C Hash Result Little Endian Format Word 8 |
| RCMDL_9 | 6_04A4h | RTIC Mem Block C Hash Result Little Endian Format Word 9 |
| RCMDL_10 | 6_04A8h | RTIC Mem Block C Hash Result Little Endian Format Word 10 |
| RCMDL_11 | 6_04ACh | RTIC Mem Block C Hash Result Little Endian Format Word 11 |
| RCMDL_12 | 6_04B0h | RTIC Mem Block C Hash Result Little Endian Format Word 12 |
| RCMDL_13 | 6_04B4h | RTIC Mem Block C Hash Result Little Endian Format Word 13 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RCMDL_14 | 6_04B8h | RTIC Mem Block C Hash Result Little Endian Format Word 14 |
| RCMDL_15 | 6_04BCh | RTIC Mem Block C Hash Result Little Endian Format Word 15 |
| RCMDL_16 | 6_04C0h | RTIC Mem Block C Hash Result Little Endian Format Word 16 |
| RCMDL_17 | 6_04C4h | RTIC Mem Block C Hash Result Little Endian Format Word 17 |
| RCMDL_18 | 6_04C8h | RTIC Mem Block C Hash Result Little Endian Format Word 18 |
| RCMDL_19 | 6_04CCh | RTIC Mem Block C Hash Result Little Endian Format Word 19 |
| RCMDL_20 | 6_04D0h | RTIC Mem Block C Hash Result Little Endian Format Word 20 |
| RCMDL_21 | 6_04D4h | RTIC Mem Block C Hash Result Little Endian Format Word 21 |
| RCMDL_22 | 6_04D8h | RTIC Mem Block C Hash Result Little Endian Format Word 22 |
| RCMDL_23 | 6_04DCh | RTIC Mem Block C Hash Result Little Endian Format Word 23 |
| RCMDL_24 | 6_04E0h | RTIC Mem Block C Hash Result Little Endian Format Word 24 |
| RCMDL_25 | 6_04E4h | RTIC Mem Block C Hash Result Little Endian Format Word 25 |
| RCMDL_26 | 6_04E8h | RTIC Mem Block C Hash Result Little Endian Format Word 26 |
| RCMDL_27 | 6_04ECh | RTIC Mem Block C Hash Result Little Endian Format Word 27 |
| RCMDL_28 | 6_04F0h | RTIC Mem Block C Hash Result Little Endian Format Word 28 |
| RCMDL_29 | 6_04F4h | RTIC Mem Block C Hash Result Little Endian Format Word 29 |
| RCMDL_30 | 6_04F8h | RTIC Mem Block C Hash Result Little Endian Format Word 30 |
| RCMDL_31 | 6_04FCh | RTIC Mem Block C Hash Result Little Endian Format Word 31 |
| RDMDB_0 | 6_0500h | RTIC Mem Block D Hash Result Big Endian Format Word 0 |
| RDMDB_1 | 6_0504h | RTIC Mem Block D Hash Result Big Endian Format Word 1 |
| RDMDB_2 | 6_0508h | RTIC Mem Block D Hash Result Big Endian Format Word 2 |
| RDMDB_3 | 6_050Ch | RTIC Mem Block D Hash Result Big Endian Format Word 3 |
| RDMDB_4 | 6_0510h | RTIC Mem Block D Hash Result Big Endian Format Word 4 |
| RDMDB_5 | 6_0514h | RTIC Mem Block D Hash Result Big Endian Format Word 5 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RDMDB_6 | 6_0518h | RTIC Mem Block D Hash Result Big Endian Format Word 6 |
| RDMDB_7 | 6_051Ch | RTIC Mem Block D Hash Result Big Endian Format Word 7 |
| RDMDB_8 | 6_0520h | RTIC Mem Block D Hash Result Big Endian Format Word 8 |
| RDMDB_9 | 6_0524h | RTIC Mem Block D Hash Result Big Endian Format Word 9 |
| RDMDB_10 | 6_0528h | RTIC Mem Block D Hash Result Big Endian Format Word 10 |
| RDMDB_11 | 6_052Ch | RTIC Mem Block D Hash Result Big Endian Format Word 11 |
| RDMDB_12 | 6_0530h | RTIC Mem Block D Hash Result Big Endian Format Word 12 |
| RDMDB_13 | 6_0534h | RTIC Mem Block D Hash Result Big Endian Format Word 13 |
| RDMDB_14 | 6_0538h | RTIC Mem Block D Hash Result Big Endian Format Word 14 |
| RDMDB_15 | 6_053Ch | RTIC Mem Block D Hash Result Big Endian Format Word 15 |
| RDMDB_16 | 6_0540h | RTIC Mem Block D Hash Result Big Endian Format Word 16 |
| RDMDB_17 | 6_0544h | RTIC Mem Block D Hash Result Big Endian Format Word 17 |
| RDMDB_18 | 6_0548h | RTIC Mem Block D Hash Result Big Endian Format Word 18 |
| RDMDB_19 | 6_054Ch | RTIC Mem Block D Hash Result Big Endian Format Word 19 |
| RDMDB_20 | 6_0550h | RTIC Mem Block D Hash Result Big Endian Format Word 20 |
| RDMDB_21 | 6_0554h | RTIC Mem Block D Hash Result Big Endian Format Word 21 |
| RDMDB_22 | 6_0558h | RTIC Mem Block D Hash Result Big Endian Format Word 22 |
| RDMDB_23 | 6_055Ch | RTIC Mem Block D Hash Result Big Endian Format Word 23 |
| RDMDB_24 | 6_0560h | RTIC Mem Block D Hash Result Big Endian Format Word 24 |
| RDMDB_25 | 6_0564h | RTIC Mem Block D Hash Result Big Endian Format Word 25 |
| RDMDB_26 | 6_0568h | RTIC Mem Block D Hash Result Big Endian Format Word 26 |
| RDMDB_27 | 6_056Ch | RTIC Mem Block D Hash Result Big Endian Format Word 27 |
| RDMDB_28 | 6_0570h | RTIC Mem Block D Hash Result Big Endian Format Word 28 |
| RDMDB_29 | 6_0574h | RTIC Mem Block D Hash Result Big Endian Format Word 29 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RDMDB_30 | 6_0578h | RTIC Mem Block D Hash Result Big Endian Format Word 30 |
| RDMDB_31 | 6_057Ch | RTIC Mem Block D Hash Result Big Endian Format Word 31 |
| RDMDL_0 | 6_0580h | RTIC Mem Block D Hash Result Little Endian Format Word 0 |
| RDMDL_1 | 6_0584h | RTIC Mem Block D Hash Result Little Endian Format Word 1 |
| RDMDL_2 | 6_0588h | RTIC Mem Block D Hash Result Little Endian Format Word 2 |
| RDMDL_3 | 6_058Ch | RTIC Mem Block D Hash Result Little Endian Format Word 3 |
| RDMDL_4 | 6_0590h | RTIC Mem Block D Hash Result Little Endian Format Word 4 |
| RDMDL_5 | 6_0594h | RTIC Mem Block D Hash Result Little Endian Format Word 5 |
| RDMDL_6 | 6_0598h | RTIC Mem Block D Hash Result Little Endian Format Word 6 |
| RDMDL_7 | 6_059Ch | RTIC Mem Block D Hash Result Little Endian Format Word 7 |
| RDMDL_8 | 6_05A0h | RTIC Mem Block D Hash Result Little Endian Format Word 8 |
| RDMDL_9 | 6_05A4h | RTIC Mem Block D Hash Result Little Endian Format Word 9 |
| RDMDL_10 | 6_05A8h | RTIC Mem Block D Hash Result Little Endian Format Word 10 |
| RDMDL_11 | 6_05ACh | RTIC Mem Block D Hash Result Little Endian Format Word 11 |
| RDMDL_12 | 6_05B0h | RTIC Mem Block D Hash Result Little Endian Format Word 12 |
| RDMDL_13 | 6_05B4h | RTIC Mem Block D Hash Result Little Endian Format Word 13 |
| RDMDL_14 | 6_05B8h | RTIC Mem Block D Hash Result Little Endian Format Word 14 |
| RDMDL_15 | 6_05BCh | RTIC Mem Block D Hash Result Little Endian Format Word 15 |
| RDMDL_16 | 6_05C0h | RTIC Mem Block D Hash Result Little Endian Format Word 16 |
| RDMDL_17 | 6_05C4h | RTIC Mem Block D Hash Result Little Endian Format Word 17 |
| RDMDL_18 | 6_05C8h | RTIC Mem Block D Hash Result Little Endian Format Word 18 |
| RDMDL_19 | 6_05CCh | RTIC Mem Block D Hash Result Little Endian Format Word 19 |
| RDMDL_20 | 6_05D0h | RTIC Mem Block D Hash Result Little Endian Format Word 20 |
| RDMDL_21 | 6_05D4h | RTIC Mem Block D Hash Result Little Endian Format Word 21 |

*Table continues on the next page...*

| Register | Offset | Description |
|---|---|---|
| RDMDL_22 | 6_05D8h | RTIC Mem Block D Hash Result Little Endian Format Word 22 |
| RDMDL_23 | 6_05DCh | RTIC Mem Block D Hash Result Little Endian Format Word 23 |
| RDMDL_24 | 6_05E0h | RTIC Mem Block D Hash Result Little Endian Format Word 24 |
| RDMDL_25 | 6_05E4h | RTIC Mem Block D Hash Result Little Endian Format Word 25 |
| RDMDL_26 | 6_05E8h | RTIC Mem Block D Hash Result Little Endian Format Word 26 |
| RDMDL_27 | 6_05ECh | RTIC Mem Block D Hash Result Little Endian Format Word 27 |
| RDMDL_28 | 6_05F0h | RTIC Mem Block D Hash Result Little Endian Format Word 28 |
| RDMDL_29 | 6_05F4h | RTIC Mem Block D Hash Result Little Endian Format Word 29 |
| RDMDL_30 | 6_05F8h | RTIC Mem Block D Hash Result Little Endian Format Word 30 |
| RDMDL_31 | 6_05FCh | RTIC Mem Block D Hash Result Little Endian Format Word 31 |

## 13.155.2 Function

The results of the RTIC hashing operations are stored in the RTIC Hash Result Registers (256 bits for SHA-256, 512 bits for SHA-512). The hash result for Memory Block a (a= A,B,C,D) is accessed in contiguous word addresses beginning at the base address of RTIC Hash Result Register a. For each Memory Block, there are 2 addresses associated with RTIC Hash Result Register a. Reading successive words starting at the RaMDB address will return successive words, in big endian format, of the hash result for Memory Block a. Reading successive words starting at the RaMDL address will return successive words, in little endian format, of the hash result for Memory Block a.

### 13.155.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RTIC_Hash_Result | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | RTIC_Hash_Result | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.155.4 Fields

| Field | Function |
|---|---|
| 31-0<br>RTIC_Hash_Result | RTIC_Hash_Result |

## 13.156 Recoverable Error Indication Record 0 for RTIC (REIR0RTIC)

### 13.156.1 Offset

| Register | Offset |
|---|---|
| REIR0RTIC | 6_0E00h |

### 13.156.2 Function

If a recoverable error occurs related to execution of a job from RTIC, error information will be captured in RTIC's REIR registers. Data for a second recoverable error related to jobs from RTIC will not be captured until the REIR0RTIC is written. If another bus error from RTIC occurs before then, the double error status bit (MISS) in REIR0RTIC will be set. When REIR0RTIC is written, all of RTIC's REIRRTIC registers are cleared and error capture is re-enabled.

## 13.156.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MISS | Reserved | | | | | TYPE | | Reserved | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.156.4 Fields

| Field | Function |
|---|---|
| 31<br>MISS | If MISS=1, a second recoverable error associated with RTIC occurred before REIR0RTIC was written following a previous RTIC recoverable error. |
| 30-26<br>— | Reserved |
| 25-24<br>TYPE | This field indicates the type of the recoverable error.<br>If TYPE = 00b : reserved<br>If TYPE = 01b : memory access error<br>If TYPE = 10b : reserved<br>If TYPE = 11b : reserved |
| 23-0<br>— | Reserved |

# 13.157 Recoverable Error Indication Record 2 for RTIC (REIR2RTIC)

### 13.157.1 Offset

| Register | Offset | Description |
|---|---|---|
| REIR2RTIC | 6_0E08h | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |

### 13.157.2 Function

See the description for Recoverable Error Indication Record 0 for RTIC (REIR0RTIC).

### 13.157.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.157.4 Fields

| Field | Function |
|---|---|
| 63-0 | This register holds the address associated with the recoverable error. |

| Field | Function |
|---|---|
| ADDR | |

# 13.158 Recoverable Error Indication Record 4 for RTIC (REIR 4RTIC)

## 13.158.1 Offset

| Register | Offset |
|---|---|
| REIR4RTIC | 6_0E10h |

## 13.158.2 Function

See the description for Recoverable Error Indication Record 0 for RTIC (REIR0RTIC).

## 13.158.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MIX | | ERR | | Reserved | | | | RWB | AXPROT | | | AXCACHE | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | ICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.158.4 Fields

| Field | Function |
|---|---|
| 31-30<br>MIX | This field holds the memory interface index associated with the recoverable error. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 29-28<br>ERR | This field holds the AXI error response associated with the recoverable error. |
| 27-24<br>— | Reserved |
| 23<br>RWB | This field specifies whether the memory access was a read or write. |
| 22-20<br>AXPROT | This field holds the AXI protection transaction attribute used for the memory access. |
| 19-16<br>AXCACHE | This field holds the AXI cache control transaction attribute used for the memory access. |
| 15-12<br>— | Reserved |
| 11-0<br>ICID | This field holds the ICID associated with the recoverable error. |

## 13.159 Recoverable Error Indication Record 5 for RTIC (REIR5RTIC)

### 13.159.1 Offset

| Register | Offset |
|---|---|
| REIR5RTIC | 6_0E14h |

### 13.159.2 Function

See the description for Recoverable Error Indication Record 0 for RTIC (REIR0RTIC).

### 13.159.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | SAFE | Reserved | | | | BID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.159.4 Fields

| Field | Function |
|---|---|
| 31-25<br>— | Reserved |
| 24<br>SAFE | SAFE indicates whether the AXI transaction associated with the recoverable error was a "safe" transaction. |
| 23-20<br>— | Reserved |
| 19-16<br>BID | This field holds the block identifier (see Table 13-1) of the source of the AXI transaction associated with the recoverable error. |
| 15-0<br>— | Reserved |

## 13.160 Queue Interface Control Register, most-significant (QICTL_MS)

### 13.160.1 Offset

| Register | Offset |
|---|---|
| QICTL_MS | 7_0000h |

## 13.160.2 Function

Queue Interface operation is controlled with the Queue Interface Control Register. The fields of the Queue Interface Control Register are accessed from the IP bus as two 32-bit words.

## 13.160.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | DWSO | PEO | DMBS |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | CDWSO | CWSO | CHWSO | CBSO | MDWSO | MWSO | MHWSO | MBSO | CDWSI | CWSI | CHWSI | CBSI | MDWSI | MWSI | MHWSI | MBSI |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.160.4 Fields

| Field | Function |
|---|---|
| 31-19<br>— | Reserved |
| 18<br>DWSO | Double Word Swap Override. Setting DWSO=1 complements the swap control determined by MCFGR[DWT] and QICTL_MS[PEO]. |
| 17<br>PEO | Platform Endianness Override. Setting PEO=1 complements the default platform endianness, which is indicated by the PLEND bit in the SEC Status Register (see Section SEC Status Register (SSTA)). (Complements the value of the default platform endianness for SEC.) |
| 16<br>DMBS | Descriptor Message Data Byte Swap (this applies only to internal message data transfers to/from DECO Descriptor Buffers. |
| 15<br>CDWSO | Control data doubleword swap on output. Control data are PreHeaders and Compound Frame Scatter/Gather Tables read via the DMA. Swaps doublewords within 128-bit wide data bus during a DMA write to help with endianness conversion issues. The word 0102030405060708090a0b0c0d0e0fh placed in memory will yield the following result when written into an internal cryptographic engine: |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td>
<table>
<tr><th>CDWSO</th><th>Data Result</th></tr>
<tr><td>0</td><td>0102030405060708090a0b0c0d0e0fh</td></tr>
<tr><td>1</td><td>08090a0b0c0d0e0f01020304050607h</td></tr>
</table>
</td></tr>
<tr><td>14<br>CWSO</td><td>Control data word swap on output. Control data are PreHeaders and Compound Frame Scatter/Gather Tables read via the DMA. Swaps words within 64-bit or 128-bit wide data bus during a DMA write to help with endianness conversion issues. The word 0123456789abcdefh placed in memory will yield the following result when written into an internal cryptographic engine:
<table>
<tr><th>CWSO</th><th>Data Result</th></tr>
<tr><td>0</td><td>0123456789abcdefh</td></tr>
<tr><td>1</td><td>89abcdef01234567h</td></tr>
</table>
</td></tr>
<tr><td>13<br>CHWSO</td><td>Control data halfword swap on output. Control data are PreHeaders and Compound Frame Scatter/Gather Tables read via the DMA. Swaps halfwords within words during a DMA write to help with endianness conversion issues. The word 01234567h placed in memory will yield the following result when written into an internal cryptographic engine.
<table>
<tr><th>CHWSO</th><th>Data Result</th></tr>
<tr><td>0</td><td>01234567h</td></tr>
<tr><td>1</td><td>45670123h</td></tr>
</table>
</td></tr>
<tr><td>12<br>CBSO</td><td>Control data byte swap on output. Control data are PreHeaders and Compound Frame Scatter/Gather Tables read via the DMA. Swaps bytes within words during a DMA write to help with endianness conversion issues. The word 01234567h placed in memory will give the following result when written into an internal cryptographic engine:
<table>
<tr><th>CBSO</th><th>Data Result</th></tr>
<tr><td>0</td><td>01234567h</td></tr>
<tr><td>1</td><td>67452301h</td></tr>
</table>
</td></tr>
<tr><td>11<br>MDWSO</td><td>Message Double Word Swap For Output Data. Allows for software-controllable double word swapping to assist with mixed Endianness platforms. This should be set if the hardware configuration requires double word swapping to get correct output for a 64-bit interface. The data is only corrected during a DMA write. The word 0102030405060708090a0b0c0d0e0fh placed in memory will yield the following result when written into an internal cryptographic engine.
<table>
<tr><th>CDWSI</th><th>Data Result</th></tr>
<tr><td>0</td><td>0102030405060708090a0b0c0d0e0fh</td></tr>
<tr><td>1</td><td>08090a0b0c0d0e0f01020304050607h</td></tr>
</table>
</td></tr>
<tr><td>10<br>MWSO</td><td>Message Word Swap for Output Data. Allows for software-controllable word swapping to assist with mixed Endianness platforms. This should be set if the hardware configuration requires word swapping to get correct output for a 64-bit interface. The data is only corrected during a DMA write. The following word 0123456789abcdefh will be written to external memory as follows.</td></tr>
</table>

*Table continues on the next page...*

**Queue Interface Control Register, most-significant (QICTL_MS)**

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td><table><tr><th>MWSO</th><th>Data Result</th></tr><tr><td>0</td><td>0123456789abcdefh</td></tr><tr><td>1</td><td>89abcdef01234567h</td></tr></table></td></tr>
<tr><td>9<br>MHWSO</td><td>Message Half Word Swap for Output Data. Allows for software-controllable half-word swapping to assist with mixed Endianness platforms. This should be set if the hardware configuration is little endian and the output data needs to be written as big endian data into a 16-bit external memory. The data is only corrected during a DMA write. The following word 01234567h will be written to external memory as follows.<table><tr><th>MHWSO</th><th>Data Result</th></tr><tr><td>0</td><td>01234567h</td></tr><tr><td>1</td><td>45670123h</td></tr></table></td></tr>
<tr><td>8<br>MBSO</td><td>Message Byte Swap for Output Data. Allows for a software-controllable byte swapping to assist with mixed Endianness platforms. This should be set if the hardware configuration is little endian and the output data needs to be written as big endian data. The data is only corrected during a DMA write. The word 01234567h placed in memory will give the following result when written into an internal cryptographic engine:<table><tr><th>MBSO</th><th>Data Result</th></tr><tr><td>0</td><td>01234567h</td></tr><tr><td>1</td><td>67452301h</td></tr></table></td></tr>
<tr><td>7<br>CDWSI</td><td>Control data doubleword swap on input. Control data are PreHeaders and Compound Frame Scatter/Gather Tables read via the DMA. Swaps doublewords within 128-bit wide data bus during a DMA read to help with endianness conversion issues. The word 0102030405060708090a0b0c0d0e0fh placed in memory will yield the following result when written into an internal cryptographic engine.<table><tr><th>CDWSI</th><th>Data Result</th></tr><tr><td>0</td><td>0102030405060708090a0b0c0d0e0fh</td></tr><tr><td>1</td><td>08090a0b0c0d0e0f01020304050607h</td></tr></table></td></tr>
<tr><td>6<br>CWSI</td><td>Control data word swap on input. Control data are PreHeaders and Compound Frame Scatter/Gather Tables read via the DMA. Swaps words within 64-bit or 128-bit wide data bus during a DMA read to help with endianness conversion issues. The word 0123456789abcdefh placed in memory will yield the following result when written into an internal cryptographic engine.<table><tr><th>CWSI</th><th>Data Result</th></tr><tr><td>0</td><td>0123456789abcdefh</td></tr><tr><td>1</td><td>89abcdef01234567h</td></tr></table></td></tr>
</table>

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>5<br>CHWSI</td><td>Control data halfword swap on input. Control data are PreHeaders and Compound Frame Scatter/Gather Tables read via the DMA. Swaps halfwords within words during a DMA read to help with endianness conversion issues. The word 01234567h placed in memory will yield the following result when written into an internal cryptographic engine.
<table>
<tr><th>CHWSI</th><th>Data Result</th></tr>
<tr><td>0</td><td>01234567h</td></tr>
<tr><td>1</td><td>45670123h</td></tr>
</table></td></tr>
<tr><td>4<br>CBSI</td><td>Control data byte swap on input. Control data are PreHeaders and Compound Frame Scatter/Gather Tables read via the DMA. Swaps bytes within words during a DMA read to help with endianness conversion issues. The word 01234567h placed in memory will give the following result when written into an internal cryptographic engine:
<table>
<tr><th>CBSI</th><th>Data Result</th></tr>
<tr><td>0</td><td>01234567h</td></tr>
<tr><td>1</td><td>67452301h</td></tr>
</table></td></tr>
<tr><td>3<br>MDWSI</td><td>Allows for a software-controllable message data double word swap to assist with mixed Endianness platforms. The data is only corrected during a DMA read. The word 0102030405060708090a0b0c0d0e0fh placed in memory will yield the following result when written into an internal cryptographic engine.
<table>
<tr><th>CDWSI</th><th>Data Result</th></tr>
<tr><td>0</td><td>0102030405060708090a0b0c0d0e0fh</td></tr>
<tr><td>1</td><td>08090a0b0c0d0e0f01020304050607h</td></tr>
</table></td></tr>
<tr><td>2<br>MWSI</td><td>Allows for a software-controllable message data word swap to assist with mixed Endianness platforms. The data is only corrected during a DMA read. This may be necessary if message data does not arrive properly when using a 64-bit interface. The word 0123456789abcdefh placed in memory will yield the following result when written into an internal cryptographic engine.
<table>
<tr><th>MWSI</th><th>Data Result</th></tr>
<tr><td>0</td><td>0123456789abcdefh</td></tr>
<tr><td>1</td><td>89abcdef01234567h</td></tr>
</table></td></tr>
<tr><td>1<br>MHWSI</td><td>Allows for a software-controllable message data half-word swap to assist with mixed Endianness platforms. The data is only corrected during a DMA read. This may be necessary if message data is not swapped properly when accessing 16 bit memories. The word 01234567h placed in memory will yield the following result when written into an internal cryptographic engine.
<table>
<tr><th>MHWSI</th><th>Data Result</th></tr>
<tr><td>0</td><td>01234567h</td></tr>
<tr><td>1</td><td>45670123h</td></tr>
</table></td></tr>
</table>

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>0<br>MBSI</td><td>Allows for a software-controllable message data byte swap to assist with mixed Endianness platforms. This byte swap will work in conjunction with the platform endianness (see PEO field definition above). The data is corrected only during a DMA read. A 0h value for the platform endianness will cause a byte swap. The word 01234567h placed in memory will give the following result when written into an internal cryptographic engine:
<table>
<tr><th></th><th></th><th colspan="2">Platform Endianness<br>(PLEND XOR PEO)</th></tr>
<tr><th></th><th></th><th>0</th><th>1</th></tr>
<tr><td rowspan="2">Mess Byte Swap (MBSI)</td><td>0</td><td>01234567h</td><td>67452301h</td></tr>
<tr><td>1</td><td>67452301h</td><td>01234567h</td></tr>
</table>
</td></tr>
</table>

# 13.161 Queue Interface Control Register, least-significant (QICTL_LS)

## 13.161.1 Offset

| Register | Offset |
|---|---|
| QICTL_LS | 7_0004h |

## 13.161.2 Function

Queue Interface operation is controlled with the Queue Interface Control Register. The fields of the Queue Interface Control Register are accessed from the IP bus as two 32-bit words.

## 13.161.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | CROV |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | SOE | STOP | DQEN |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.161.4 Fields

| Field | Function |
|---|---|
| 31-17<br>— | Reserved |
| 16<br>CROV | Critical Resource Override. If set, critical resource workload is not considered when selecting the next job for transfer to the job queue controller for job scheduling. If clear, critical workload may affect job selection priority as follows. If the Preheader associated with a job specifies a critical resource ID and all instances of that resource are in use by jobs already transferred to the job queue controller, the job is assigned the lowest selection priority. |
| 15-3<br>— | Reserved |
| 2<br>SOE | Stop on error. If set, a QI-detected error will cause QI to stop frame dequeue and enqueue operations and transfer of jobs to the job queue controller. See the Queue Interface Status Register (see Section Queue Interface Status Register (QISTA)) for a description of the errors. If the REI registers are programmed to halt SEC after a recoverable error and that recoverable error occurs, this will cause the DEBUGCTL[STOP] bit to assert. SEC will acknowledge that the stop is complete by setting the DEBUGCTL[STOP_ACK] bit. The DEBUGCTL[STOP] bit must be cleared in order to restart SEC. |
| 1<br>STOP | Stop. Write 1 to STOP to direct QI to gracefully stop all operations. Once stopped, the STOPD in the QI Status Register will be 1. To reset the QI, write 1 to STOP (again) and 0 to DQEN. (DQEN may already be 0.) QI will assert a signal that causes SEC to terminate processing of all QI jobs with a DECO DNR error status. No new dequeue commands will be issued since DQEN is 0. The STOPD bit will be cleared while the QI jobs are flushed. After all QI jobs have been flushed (enqueued to QMan), QI will reset itself, restoring all registers to their default/reset state, including resetting the STOP bit to 0. |
| 0<br>DQEN | Dequeue enable. If set, the Queue Interface will dequeue frames, if available, from the Queue Manager. |

# 13.162 Queue Interface Status Register (QISTA)

## 13.162.1 Offset

| Register | Offset |
|---|---|
| QISTA | 7_000Ch |

## 13.162.2 Function

Software can determine the current status of the Queue Interface by reading the Queue Interface Status Register. Note that the error bits are "sticky", and will reflect all errors that have occurred since the error bits were cleared. The status for a particular error is cleared by writing a 1 to the appropriate error status bit. If an error occurs on the same clock cycle that the corresponding bit was cleared, the error bit will be left set so that the error is not missed.

## 13.162.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | STOPD | Reserved | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | TBTSERR | TBPDERR | OFTLERR | CFWRERR | BTSERR | BPDERR | OFWRERR | CFRDERR | PHRDERR |
| W | | | | | | | | W1C | W1C | W1C | W1C | W1C | W1C | W1C | W1C | W1C |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.162.4 Fields

| Field | Function |
|---|---|
| 31<br>STOPD | Stopped. Frame dequeue and enqueue operations and transfer of jobs to the Job Queue Controller has stopped. This condition will occur after the QICTL Register STP bit is set or after an error bit is set with the QICTL Register SOE bit set. There may be a delay of several cycles from the time the stopping event occurs until STOPD is asserted, while Queue Interface state machine finishes current operations. This bit is read-only. |
| 30-9<br>— | Reserved |
| 8<br>TBTSERR | Table buffer too small error. The buffer allocated for a Compound Frame or Output Frame Scatter/Gather Table is too small to hold the required number of entries. |
| 7<br>TBPDERR | Table buffer pool depletion error. The buffer pool specified by the TBPID field of the Preheader is depleted. |
| 6<br>OFTLERR | Output Frame too large error. Number of bits required to represent Output Frame length exceeds size of length field in Frame Descriptor. Length field size is different for short and long frame formats. This bit is cleared by writing a "1" to this bit position. |
| 5<br>CFWRERR | Compound Frame write error. Error reported by the DMA while writing the Output Frame entry in the Compound Frame Scatter/Gather Table. This bit is cleared by writing a "1" to this bit position. |
| 4<br>BTSERR | Buffer too small error. The buffer allocated for an Output Frame Scatter/Gather Table is too small to hold the required number of entries or the buffer offset specified by the PreHeader is greater than or equal to the buffer size. This bit is cleared by writing a "1" to this bit position. |
| 3<br>BPDERR | Buffer pool depletion error. The number of buffers required for an Output Frame could not be acquired because the buffer pool specified in the PreHeader is depleted. This bit is cleared by writing a "1" to this bit position. |
| 2<br>OFWRERR | Output Frame write error. Error reported by the DMA while writing the Output Frame Scatter/Gather Table. This bit is cleared by writing a "1" to this bit position. |
| 1<br>CFRDERR | Compound Frame read error. Error reported by the DMA during Compound Frame Scatter/Gather Table read or format of Compound Frame Scatter/Gather Table invalid (F-bit set in first table entry or not set in second table entry). This bit is cleared by writing a "1" to this bit position. If this error occurs, SEC will not attempt to update the Output Frame entry of the Compound Frame Scatter/Gather Table when the frame is enqueued. If the reported error was due to a DMA error, a write would likely result in another DMA error. |
| 0<br>PHRDERR | PreHeader read error. An error was reported by the DMA during a PreHeader read. This bit is cleared by writing a "1" to this bit position. |

## 13.163 Queue Interface Dequeue Configuration Register, most-significant half (QIDQC_MS)

### 13.163.1 Offset

| Register | Offset |
|---|---|
| QIDQC_MS | 7_0010h |

### 13.163.2 Function

Queue Interface dequeue command parameters are specified with this register. The fields of the Queue Interface Dequeue Configuration Register are accessed as two 32-bit registers.

### 13.163.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | SPFCNT | | | Reserved | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | BCNT | | | | | | | | | | | | | | | |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

### 13.163.4 Fields

| Field | Function |
|---|---|
| 31-27<br>— | Reserved. |
| 26-24<br>SPFCNT | Subportal frame count threshold. Each dequeue command issued to the Queue Manager specifies a subportal ID. Queue Interface uses multiple subportals to get frames from different Frame Queues and it will only issue a dequeue command for a subportal if the number of frames being processed for that subportal is less than this frame count threshold. Setting this field to 0 will stop dequeue commands since the number of frames being processed is never less than 0. |
| 23-16<br>— | Reserved. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 15-0<br>BCNT | Dequeue command byte count. This is the command byte count value used in dequeue commands to the Queue Manager. The default value of all 1's indicates that the byte count limit is not used. Smaller byte counts can be programmed to limit the number of large frames that are dequeued when the FCNT value is 1. Queue Manager will always dequeue at least one frame (unless there are no frames available). If the amount of data in the first frame is less than the requested byte count, and more frames are available, Queue Manager will provide a second frame in the dequeue response. If the amount of data in the first two frames is less than the requested byte count, and more frames are available, Queue Manager will provide a third frame in the dequeue response. |

## 13.164 Queue Interface Dequeue Configuration Register, least-significant half (QIDQC_LS)

### 13.164.1 Offset

| Register | Offset |
|---|---|
| QIDQC_LS | 7_0014h |

### 13.164.2 Function

Queue Interface dequeue command parameters are specified with this register. The fields of the Queue Interface Dequeue Configuration Register are accessed as two 32-bit registers.

### 13.164.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SRC | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SRC | | | | | | | | Reserved | | | FCNT | VERB | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

### 13.164.4 Fields

| Field | Function |
|---|---|
| 31-8<br>SRC | Dequeue command source. This is the command source value used in dequeue commands to the Queue Manager. For normal operation, only the default value should be programmed. Other values are allowed for test and debug. |
| 7-5<br>— | Reserved |
| 4<br>FCNT | Dequeue command frame count. This is the frame count value used in dequeue commands to the Queue Manager.<br>See the Multi-core Datapath Acceleration Architecture Infrastructure Usage document for more detail on Frame Descriptions.<br>0b - Dequeue one Frame Description<br>1b - Dequeue up to three Frame Descriptions. (QMan may supply less than three based on availability and FQ configuration) |
| 3-0<br>VERB | Dequeue command verb. This is the command verb value used in dequeue commands to the Queue Manager. The default value indicates that the Queue Manager should perform a scheduled dequeue from the channel dedicated to SEC with priority precedence. For normal operation, only the default value should be programmed. Other values are allowed for test and debug. Note that unscheduled dequeues are not supported. |

## 13.165 Queue Interface Enqueue Configuration Register, most-significant half (QIEQC_MS)

### 13.165.1 Offset

| Register | Offset |
|---|---|
| QIEQC_MS | 7_0018h |

### 13.165.2 Function

Queue Interface enqueue command parameters are specified with this register. The fields of the Queue Interface Enqueue Configuration Register are accessed as two 32-bit registers.

### 13.165.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | FC | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

### 13.165.4 Fields

| Field | Function |
|---|---|
| 31-2<br>— | Reserved |
| 1-0<br>FC | Enqueue command frame color. This is the command frame color value used in Enqueue Commands to the Queue Manager. See the Queue Manager for more detail. |

## 13.166 Queue Interface Enqueue Configuration Register, least-significant half (QIEQC_LS)

### 13.166.1 Offset

| Register | Offset |
|---|---|
| QIEQC_LS | 7_001Ch |

### 13.166.2 Function

Queue Interface enqueue command parameters are specified with this register. The fields of the Queue Interface Enqueue Configuration Register are accessed as two 32-bit registers.

### 13.166.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | TAG | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | TAG | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.166.4 Fields

| Field | Function |
|---|---|
| 31-0<br>TAG | Enqueue command tag. This is the command tag value used in enqueue commands to the Queue Manager. See the Queue Manager for more detail. |

## 13.167 Queue Interface ICID Configuration Register, most-significant half (QIIC_MS)

### 13.167.1 Offset

| Register | Offset |
|---|---|
| QIIC_MS | 7_0020h |

### 13.167.2 Function

The Queue Interface ICID Configuration Register specifies base and mask values used to derive SEQ and Non-SEQ ICID values for each job processed through the Queue Manager Interface. The SEQ ICID is asserted by SEC during DMA transactions associated with sequence command execution. The Non-SEQ ICID is used for DMA transactions associated with non-sequence command execution. These two ICID values for each job are used by both the Queue Manager Interface and DECO. The fields of the Queue Interface ICID Configuration Register are accessed as two 32-bit registers.

### 13.167.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | QNSIOM | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.167.4 Fields

| Field | Function |
|---|---|
| 31-26<br>— | Reserved |
| 25-16<br>QNSIOM | QI Non-SEQ ICID Mask When computing the Non-SEQ ICID for Jobs received via the Queue Manager Interface, this mask is ANDed with the ICID from the dequeued frame description for the job before being added to the QI Non-SEQ ICID Base.<br>Non-SEQ ICID = (ICID & QNSIOM) + QNSICIDB<br>The calculation of the Non-SEQ ICID may be overridden by a special code in the STATUS/CMD field of the dequeued Frame Descriptor. See Frame descriptors for more details.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |
| 15-0<br>— | Reserved |

## 13.168 Queue Interface ICID Configuration Register, least-significant half (QIIC_LS)

### 13.168.1 Offset

| Register | Offset |
|---|---|
| QIIC_LS | 7_0024h |

### 13.168.2 Function

The Queue Interface ICID Configuration Register specifies base and mask values used to derive SEQ and Non-SEQ ICID values for each job processed through the Queue Manager Interface. The SEQ ICID is asserted by SEC during DMA transactions associated with sequence command execution. The Non-SEQ ICID is used for DMA transactions associated with non-sequence command execution. These two ICID values for each job are used by both the Queue Manager Interface and DECO. The fields of the Queue Interface ICID Configuration Register are accessed as two 32-bit registers.

### 13.168.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | QNSICIDB | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | QSICIDB | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.168.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved |
| 27-16<br>QNSICIDB | QI Non-SEQ ICID Base. When computing the Non-SEQ ICID, the QNSIOM mask is ANDed with the ICID from the dequeued frame description for the job before being added to the QI Non-SEQ ICID Base. The Non-SEQ ICID is output by the DMA for transactions associated with execution of a command that is not a SEQ command (e.g. KEY, LOAD, STORE....). |

*Table continues on the next page...*

| Field | Function |
|---|---|
| | The calculation of the Non-SEQ ICID may be overridden by a special code in the STATUS/CMD field of the dequeued Frame Descriptor. See Frame descriptors for more details.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |
| 15-12<br>— | Reserved |
| 11-0<br>QSICIDB | QI SEQ ICID Base. When computing the SEQ ICID, this Base is added to the ICID from the dequeued frame description for the job. The SEQ ICID is output by the DMA for transactions associated with execution of a SEQ command (e.g. SEQ KEY, SEQ LOAD, SEQ STORE....).<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

# 13.169 Queue Interface Descriptor Word 0 Register (QIDESC0)

## 13.169.1 Offset

| Register | Offset |
|---|---|
| QIDESC0 | 7_0100h |

## 13.169.2 Function

Several internal registers are accessible for debug purposes. The contents of these registers change as jobs move through the SEC, so access to that data may only be useful if operations are hung or stalled.

## 13.169.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | DESCWD | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | DESCWD | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.169.4 Fields

| Field | Function |
|---|---|
| 31-0<br>DESCWD | Descriptor Word. This register contains the first word of the Job Descriptor, a Header Command, for the job awaiting transfer to the job queue controller. The contents of this register will be zero when no job transfer is pending. The length of the descriptor can be determined from a field in the Header command. |

## 13.170 Queue Interface Descriptor Word a Registers (QIDESC1 - QIDESC12)

### 13.170.1 Offset

For a = 1 to 12:

| Register | Offset | Description |
|---|---|---|
| QIDESCa | 7_0100h + (a × 4h) | QI Descriptor Word a |

### 13.170.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | DESCWD | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | DESCWD | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.170.3 Fields

| Field | Function |
|---|---|
| 31-0<br>DESCWD | Descriptor Word. These registers contain words of the Job Descriptor for the job awaiting transfer to the job queue controller. The contents of this register will be zero when no job transfer is pending. |

## 13.171 Queue Interface Compound Frame Scatter/Gather Table Registers (QICFOFH_MS - QICFIFL_LS)

### 13.171.1 Offset

| Register | Offset | Description |
|---|---|---|
| QICFOFH_MS | 7_0210h | Output Frame High_MS |
| QICFOFH_LS | 7_0214h | Output Frame High_LS |
| QICFOFL_MS | 7_0218h | Output Frame Low_MS |
| QICFOFL_LS | 7_021Ch | Output Frame Low_LS |
| QICFIFH_MS | 7_0220h | Input Frame High_MS |
| QICFIFH_LS | 7_0224h | Input Frame High_LS |
| QICFIFL_MS | 7_0228h | Input Frame Low_MS |
| QICFIFL_LS | 7_022Ch | Input Frame Low_LS |

### 13.171.2 Function

QICFOFH_MS QI Compound Frame Output Frame High

QICFOFL_MS QI Compound Frame Output Frame Low

QICFIFH_MS QI Compound Frame Input Frame High

QICFIFL_MS QI Compound Frame Input Frame Low

### 13.171.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | CFSGT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | CFSGT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.171.4 Fields

| Field | Function |
|---|---|
| 31-0<br>CFSGT | Compound Frame Scatter/Gather Table. These registers contain the compound frame scatter/gather table most recently read from memory. |

## 13.172 Queue Interface Job ID Valid Register (QIJIDVALID)

### 13.172.1 Offset

| Register | Offset |
|---|---|
| QIJIDVALID | 7_0300h |

### 13.172.2 Function

Information and status for each job handled by the Queue Interface is accessible through the Queue Interface Job Registers while job processing is in progress. Each 64-bit register is accessed as two 32-bit words. After determining that a specific Job ID is in use (by reading QIJIDVALID), data for the job assigned that Job ID can be accessed by programming the QI Job Select Register (QIJOBSELECT) with the Job ID. As long as the Job ID remains valid, information and status for the corresponding job can be read from QIJIDRDY, QIJIDXFRD, QIJIDEXEC, QIJIDDONE, QISPFC, QISPDQRD, QIJOBSELECT, QIJOBSTAT, QIJOBSDAL, QIJOBFOQID, QIJOBERR, QIJOBTEOL, QIJOBFD, QIJOBOFD, and QIJOBIFD. (These registers replace the 152 Queue Interface Registers that were used to retrieve status for up to 15 active QI jobs in previous versions of the SEC.)

Data for each active job handled by the Queue Interface is stored in a "Job Buffer". A Job ID from 1 through JOBIDMAX is associated with each Job Buffer. JOBIDMAX is the number of jobs that can be managed by the Queue Interface at one time. Its value can be found by reading the JOBIDMAX field of the QI Job Select Register. Some job data need to be stored by the Queue Interface only until the job has been transferred to the Job Queue Controller. For that data there are a number of Job Buffer Extensions (or Job Extensions). The number of Job Extensions is smaller than the number of Job Buffers and can be determined by reading the JBXIDMAX field of the QI Job Select Register. The ID of the Job Extension assigned to a job can be determined by writing the Job ID of the job to the JOBID field and 0 to the JBXID field of the QI Job Select Register, then reading the JBXID field.

Access to the Job Registers requires:

(1) valid value in Job Buffer Index field of Job Select Register

(2) valid value or 0 in Job Buffer Extension Index field of Job Select Register and

(3) job buffer selected by Job Buffer Index must be in use

If the job selected by the Job Buffer Index has been transferred to the Job Queue Controller, the Job Buffer Extension data is no longer available. In that case, those fields of the registers will read as 0. In addition, when accessing registers that contain a mix of job buffer and job buffer extension data, the extension data will be 0 if the job has been transferred to the JQ. (After job transfer, the Job Buffer Extension is released for use with another job.)

## 13.172.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JIDxx | | | | | | | | | | | | | | | Reserved |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.172.4 Fields

| Field | Function |
|---|---|
| 63-32<br>— | Reserved |
| 31-16<br>— | Reserved |
| 15-1<br>JIDxx | Job ID xx Valid. Job ID xx and the associated job buffer are currently in use.<br>000000000000000b - Job ID xx is not currently in use.<br>000000000000001b - Job ID xx is currently in use. |
| 0<br>— | Reserved |

# 13.173 Queue Interface Job ID Job Ready Register (QIJIDRDY)

## 13.173.1 Offset

| Register | Offset |
|---|---|
| QIJIDRDY | 7_0308h |

## 13.173.2 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JIDxx | | | | | | | | | | | | | | | Reserved |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.173.3 Fields

| Field | Function |
|---|---|
| 63-32<br>— | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 31-16<br>— | Reserved |
| 15-1<br>JIDxx | Job ID xx Ready. The job with Job ID xx is ready for processing or ready for enqueue. This bit is first set when the job is ready for transfer to the job queue controller for the start of processing. It is cleared when processing is done. The bit is set again when post-processing is done and the job is ready for enqueue. It remains set until the job is enqueued.<br>000000000000000b - The job with Job ID xx is not yet ready for enqueue or transfer.<br>000000000000001b - The job with Job ID xx is ready for enqueue or transfer. |
| 0<br>— | Reserved |

## 13.174 Recoverable Error Indication Record 0 for the Queue Interface (REIR0QI)

### 13.174.1 Offset

| Register | Offset |
|---|---|
| REIR0QI | 7_0700h |

### 13.174.2 Function

One type of recoverable error is defined for QI: system bus access (memory read/write) errors. If a recoverable error occurs related to execution of a job from QI, error information will be captured in the QI's REIR registers. Data for a second QI recoverable error will not be captured unless REIR0QI was written (with any value) prior to the occurrence of that error. If another recoverable error occurs before that write, the missed error status bit (MISS) in REIR0QI will be set. When REIR0QI is written, all of QI's REIRQI registers are cleared and error capture is re-enabled.

### 13.174.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MISS | Reserved | | | | | TYPE | | Reserved | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.174.4 Fields

| Field | Function |
|---|---|
| 31<br>MISS | If MISS=1, a second QI recoverable error occurred before REIR0QI was written to re-enable error data capture. |
| 30-26<br>— | Reserved |
| 25-24<br>TYPE | This field indicates the type of the recoverable error.<br>If TYPE = 1 : memory access error<br>All other values reserved. |
| 23-0<br>— | Reserved |

## 13.175 Recoverable Error Indication Record 1 for the Queue Interface (REIR1QI)

### 13.175.1 Offset

| Register | Offset |
|---|---|
| REIR1QI | 7_0704h |

## 13.175.2 Function

This register is used to report information related to isolation errors (REIR0QI.TYPE = 2). REIR1QI will return all zeros for memory access errors (TYPE = 1). See the description for Recoverable Error Indication Record 0 for the Queue Interface (REIR 0QI).

## 13.175.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | BDI | Reserved | | | NONSEQ_ICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.175.4 Fields

| Field | Function |
|---|---|
| 31-16<br>— | Reserved |
| 15<br>BDI | For TYPE 2 and 3 errors this field holds the BDI associated with the recoverable error. For TYPE 1 errors this field will return 0. |
| 14-12<br>— | Reserved |
| 11-0<br>NONSEQ_ICID | For TYPE 2 and 3 errors this field holds the Non-Sequence_ICID associated with the recoverable error. For TYPE 1 errors this field will return 00h.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

# 13.176 Recoverable Error Indication Record 2 for the Queue Interface (REIR2QI)

## 13.176.1 Offset

| Register | Offset |
|---|---|
| REIR2QI | 7_0708h |

## 13.176.2 Function

See the description for Recoverable Error Indication Record 0 for the Queue Interface (REIR0QI).

## 13.176.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ADDR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.176.4 Fields

| Field | Function |
|---|---|
| 63-0<br>ADDR | REIR2QI holds the address associated with the recoverable QI error. Note that this register may be double-word swapped. See MCFGR[DWT] (Master Configuration Register (MCFGR)). |

## 13.177 Recoverable Error Indication Record 4 for the Queue Interface (REIR4QI)

### 13.177.1 Offset

| Register | Offset |
|---|---|
| REIR4QI | 7_0710h |

### 13.177.2 Function

See the description for Recoverable Error Indication Record 0 for the Queue Interface (REIR0QI).

### 13.177.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | MIX | | ERR | | Reserved | | | | RWB | AXPROT | | | AXCACHE | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | ICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.177.4 Fields

| Field | Function |
|---|---|
| 31-30<br>MIX | This field holds the memory interface index associated with the recoverable error. |
| 29-28<br>ERR | This field holds the AXI error response associated with the recoverable error. |
| 27-24<br>— | Reserved |
| 23<br>RWB | This field specifies whether the memory access was a read or write. |
| 22-20<br>AXPROT | This field holds the AXI protection transaction attribute used for the memory access. |
| 19-16<br>AXCACHE | This field holds the AXI cache control transaction attribute used for the memory access. |
| 15-12<br>— | Reserved |
| 11-0<br>ICID | For Type 1 errors this field holds the ICID transaction attribute associated with the recoverable error.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

## 13.178 Recoverable Error Indication Record 5 for the Queue Interface (REIR5QI)

### 13.178.1 Offset

| Register | Offset |
|---|---|
| REIR5QI | 7_0714h |

### 13.178.2 Function

See the description for Recoverable Error Indication Record 0 for the Queue Interface (REIR0QI).

### 13.178.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | SAFE | Reserved | | | | BID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.178.4 Fields

| Field | Function |
|---|---|
| 31-25<br>— | Reserved |
| 24<br>SAFE | SAFE indicates whether the AXI transaction associated with the recoverable error was a safe transaction (either read-safe or write-safe). |
| 23-20<br>— | Reserved |
| 19-16<br>BID | BID holds the block identifier (see Table 13-1) of the source of the AXI transaction associated with the recoverable error. |
| 15-0<br>— | Reserved |

## 13.179 CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG)

### 13.179.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC1MR_RNG | 8_0004h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.179.2 Function

The Class 1 Mode Register is used to tell the Class 1 CHAs which operation is being requested. There is one copy of this register per DECO/CCB. The interpretation of this register will be unique for each CHA. The Class 1 Mode Register has several independent definitions, one for Public Key algorithms (see Section CCB a Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK - C2C1MR_PK)), one for RNG (see Section CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG)), and one for all others (see this section). The Class 1 Mode Register is automatically written by the OPERATION Command. Using a descriptor, the only way to write to the Class 1 Mode Register is via the OPERATION Command. This register is automatically cleared when a key is to be encrypted or decrypted using the KEY or FIFO STORE Commands. This register is also automatically cleared when the signature over a Trusted Descriptor is checked or a Trusted Descriptor is re-signed.

When the Class 1 Mode register is used to control the RNG, the following format is used.

## 13.179.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | ALG | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | C2K | SK | AI | PS | OBP | NZB | Reserved | | SH | | AS | | PR | TST |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.179.4 Fields

| Field | Function |
|---|---|
| 31-24<br>— | Reserved. Must be 0. |
| 23-16<br>ALG | Algorithm. This field specifies which algorithm is being selected.<br>01010000b - RNG |
| 15-14<br>— | Reserved. Must be 0. |
| 13<br>C2K | Class 2 Key. This bit is ignored for all algorithms other than AES.<br>0b - AES will use the Class 1 Key for CCM and GCM modes.<br>1b - AES will use the Class 2 Key for CCM and GCM modes. Setting this bit =1 will result in a mode error for other AES modes. |
| 12<br>SK | Secure Key. For RNG OPERATION commands this bit of the AAI field is interpreted as the Secure Key field. If SK=1 and AS=00 (Generate), the RNG will generate data to be loaded into the JDKEK, TDKEK and TDSK. If a second Generate command is issued with SK=1, a Secure Key error will result. If SK=0 and AS=00 (Generate), the RNG will generate data to be stored as directed by the FIFO STORE command. The SK field is ignored if AS!=00. |
| 11<br>AI | Additional Input Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Additional Input Included field. If AS=00 (Generate) and AI=1, the256 bits of additional data supplied via the Class 1 Context Register will be used as additional entropy during random number generation. If AS=10 (Reseed) and AI=1, the additional data supplied via the Class 1 Context register will be used as additional entropy input during the reseeding operation. The AI field is ignored if AS=01 (Instantiate) or AS=11 (Uninstantiate). |
| 10<br>PS | Personalization String Included. For RNG OPERATION commands this bit of the AAI field is interpreted as the Personalization String Included field. If AS=01 (Instantiate) and PS=1, a personalization string of 256 bits supplied via the Class 1 Context register is used as additional “entropy” input during instantiation. Note that the personalization string does not need to be random. A device-unique value can be used to further guarantee that no two RNGs are ever instantiated with the same seed value. (Note that the entropy generated by the TRNG already ensures this with high probability.) The PS field is ignored if AS≠01. |
| 9<br>OBP | Odd Byte Parity. For RNG Operation commands this bit of the AAI field is interpreted as the Odd Byte Parity field. If AS=00 (Generate) and OBP=1, every byte of data generated during random number generation will have odd parity. That is, the 128 possible bytes values that have odd parity will be generated at random. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The OBP field is ignored if AS≠00. |
| 8<br>NZB | NonZero bytes. For RNG OPERATION commands this bit of the AAI field is interpreted as the NonZero Bytes field. If AS=00 (Generate) and NZB=1, no byte of data generated during random number generation will be 00h, but (if OBP=0) the remaining 255 values will be generated at random. Note that setting NZB=1 has no effect if OBP=1, since zero bytes are already excluded when odd byte parity is selected. If AS=00 (Generate) and OBP=0 and NZB=0, all 256 possible byte values will be generated at random. The NZB field is ignored if AS≠00. |
| 7-6<br>— | Reserved. For RNG commands these bits of the AAI field are reserved. |
| 5-4<br>SH | State Handle. For RNG OPERATION commands these bits of the AAI field are interpreted as the State Handle field. The command is issued to the State Handle selected via this field. An error will be generated if the selected state handle is not implemented.<br>00b - State Handle 0<br>01b - State Handle 1 |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td>10b - Reserved<br>11b - Reserved</td></tr>
<tr><td>3-2<br>AS</td><td>Algorithm State. For RNG OPERATION commands these bits select RNG commands as shown below:
<table>
<tr><th>AS Value</th><th>State Handle is already instantiated</th><th>State Handle is NOT already instantiated</th></tr>
<tr><td>00 Generate</td><td>Generate random data per the mode in which the state handle was instantiated.</td><td>Error</td></tr>
<tr><td>01 Instantiate</td><td>Error</td><td>Instantiate the state handle in either test mode or non-deterministic mode as specified by TST, and either to support prediction resistance or not to support prediction resistance as specified by PR.</td></tr>
<tr><td>10 Reseed</td><td>Reseed the state handle.</td><td>Error</td></tr>
<tr><td>11 Uninstantiate</td><td>Uninstantiate the state handle.</td><td>Error</td></tr>
</table>
1. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in Test mode but a Generate operation requests non-deterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.</td></tr>
<tr><td>1<br>PR</td><td>Prediction Resistance. For RNG OPERATION commands this bit is interpreted as:
<table>
<tr><th>AS Value</th><th>PR = 0</th><th>PR = 1</th></tr>
<tr><td>00 Generate</td><td>Do NOT reseed prior to generating new random data</td><td>If the state handle was instantiated to support prediction resistance, reseed prior to generating new random data. If the state handle was NOT instantiated to support prediction resistance, generate an error.</td></tr>
<tr><td>01 Instantiate</td><td>Instantiate the state handle to NOT support prediction resistance</td><td>Instantiate the state handle to support prediction resistance</td></tr>
<tr><td>10 Reseed</td><td>Reseed the state handle. PR bit is ignored.</td><td>Reseed the state handle. PR bit is ignored.</td></tr>
<tr><td>11 Uninstantiate</td><td>Uninstantiate the state handle. PR bit is ignored.</td><td>Uninstantiate the state handle. PR bit is ignored.</td></tr>
</table>
1. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in Test mode but a Generate operation requests non-deterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.</td></tr>
<tr><td>0<br>TST</td><td>Test Mode Request. For RNG OPERATION commands this bit is interpreted as:</td></tr>
</table>

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td>
<table>
<tr><th>AS Value</th><th>TST = 0</th><th>TST = 1</th></tr>
<tr><td>00 Generate</td><td>If the selected state handle is in non-deterministic mode, generate new random data.<br>If the selected state handle is in deterministic mode, generate a Test error.-[1]</td><td>If the selected state handle is in deterministic mode, generate new random data.<br>If the selected state handle is in non-deterministic mode, generate a Test error..</td></tr>
<tr><td>01 Instantiate</td><td>Instantiate the state handle in normal (non-deterministic) mode.</td><td>Instantiate the state handle in test (deterministic) mode.</td></tr>
<tr><td>10 Reseed</td><td>If the selected state handle is in non-deterministic mode, reseed the state handle.<br>If the selected state handle is in deterministic mode, generate a Test error.</td><td>If the selected state handle is in non-deterministic mode, reseed the state handle.<br>If the selected state handle is in deterministic mode, generate a Test error.</td></tr>
<tr><td>11 Uninstantiate</td><td>Uninstantiate the state handle. TST bit is ignored.</td><td>Uninstantiate the state handle. TST bit is ignored.</td></tr>
</table>
<br>1. There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in Test mode but a Generate operation requests non-deterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.
</td></tr>
</table>

[1] There is one exception to this rule. A Test Error will not be generated if State Handle 0 is in Test mode but a Generate operation requests non-deterministic data from State Handle 0. This permits deterministic testing of the built-in protocols prior to setting the RNGSH0 bit in the Security Configuration Register. Setting RNGSH0 would normally be performed during the boot process after testing is complete.

## 13.180 CCB a Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK - C2C1MR_PK)

### 13.180.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC1MR_PK | 8_0004h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.180.2 Function

The Class 1 Mode Register is used to tell the Class 1 CHAs which operation is being requested. There is one copy of this register per DECO/CCB. The interpretation of this register will be unique for each CHA. The Class 1 Mode Register has several independent definitions, one for Public Key algorithms (see Section CCB a Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK - C2C1MR_PK)), one for RNG (see Section CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG)), and one for all others (see this section). The Class 1 Mode Register is automatically written by the OPERATION Command. Using a descriptor, the only way to write to the Class 1 Mode Register is via the OPERATION Command. This register is automatically cleared when a key is to be encrypted or decrypted using the KEY or FIFO STORE Commands. This register is also automatically cleared when the signature over a Trusted Descriptor is checked or a Trusted Descriptor is re-signed.

The following figure shows the Class 1 Mode Register format that is used with public key algorithms, which are algorithms that use PKHA. The Class 1 Mode register is automatically cleared following a PKHA Command. The bit assignments for the PKHA_MODE field shown in CCB a Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK - C2C1MR_PK) will be different depending on which of the three types of PKHA functions is being called. The three function types are: 1) Clear Memory, 2) Modular Arithmetic, and 3) Copy Memory. Detailed descriptions of their mode formats can be found in Table PKHA OPERATION: clear memory function, Table PKHA OPERATION: Arithmetic Functions and Table PKHA OPERATION: copy memory functions.

## 13.180.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | PKHA_MODE_MS | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | PKHA_MODE_LS | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.180.4 Fields

| Field | Function |
|---|---|
| 31-20<br>— | Reserved |
| 19-16<br>PKHA_MODE_MS | PKHA_MODE most-significant 4 bits. The format of the PKHA_MODE field differs depending on which of the three types of PKHA functions is being executed. The three function types are: 1) Clear Memory, 2) Modular Arithmetic, and 3) Copy Memory. Detailed descriptions of their mode formats can be found in Table PKHA OPERATION: clear memory function, Table PKHA OPERATION: Arithmetic Functions and Table PKHA OPERATION: copy memory functions. |
| 15-12<br>— | Reserved |
| 11-0<br>PKHA_MODE_LS | PKHA_MODE least significant 12 bits. The format of the PKHA_MODE field differs depending on which of the three types of PKHA functions is being executed. The three function types are: 1) Clear Memory, 2) Modular Arithmetic, and 3) Copy Memory. Detailed descriptions of their mode formats can be found in Table PKHA OPERATION: clear memory function, Table PKHA OPERATION: Arithmetic Functions and Table PKHA OPERATION: copy memory functions. |

## 13.181 CCB a Class 1 Mode Register Format for Non-Public Key Algorithms (C0C1MR_NPK - C2C1MR_NPK)

### 13.181.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC1MR_NPK | 8_0004h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.181.2 Function

The Class 1 Mode Register is used to tell the Class 1 CHAs which operation is being requested. There is one copy of this register per DECO/CCB. The interpretation of this register will be unique for each CHA. The Class 1 Mode Register has several independent definitions, one for Public Key algorithms (see Section CCB a Class 1 Mode Register Format for Public Key Algorithms (C0C1MR_PK - C2C1MR_PK)), one for RNG (see Section CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG)), and one for all others (see this section). The Class 1 Mode Register is automatically written by the OPERATION Command. Using a descriptor, the only way to write to the Class 1 Mode Register is via the OPERATION Command. This register is automatically cleared when a key is to be encrypted or decrypted using the KEY or FIFO STORE Commands. This register is also automatically cleared when the signature over a Trusted Descriptor is checked or a Trusted Descriptor is re-signed.

This section defines the format of the Class 1 Mode Register when used with non-public-key algorithms. The Non-Public-Key algorithms are those that do not use the PKHA.

Some examples of how to build the Class 1 Mode Register for non-Public Key algorithms:

**Table 13-3. Class 1 Mode Register examples for non-Public Key algorithms**

| Crypto service performed | ALG Mnemonic | AAI Mnemonic | AS Mnemonic | ICV | Encrypt/<br>Decrypt/<br>Protect/<br>Authenticate | ALGORITHM OPERATION Command | 32-bit Value Loaded into C1 Mode Reg |
|---|---|---|---|---|---|---|---|
| AES GCM | AES | GCM | Init / Finalize | yes | Decrypt | 8201090Eh | 0001090Eh |
| AES Counter with mod=$2^{128}$ | AES | CTR<br>Modulus $2^{128}$ | -- | no | Encrypt | 82010001h | 00010001h |
| Kasumi f9 | Kasumi | f9 | Init / Finalize | yes | Authenticate | 8207020Eh | 0007020Eh |
| Triple DES OFB mode with key parity | DES | OFB | -- | no | Decrypt | 82021400h | 00021400h |

## 13.181.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | ALG | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | C2K | AAI | | | | | | | | | AS | | ICV_TEST | ENC |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.181.4 Fields

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>31-24<br>—</td><td>Reserved. Must be 0.</td></tr>
<tr><td>23-16<br>ALG</td><td>Algorithm. This field specifies which algorithm is being selected.<br>00010000b - AES<br>00100000b - DES<br>00100001b - 3DES<br>01010000b - RNG<br>01100000b - SNOW f8<br>01110000b - Kasumi Encryption<br>10110000b - ZUC Encryption</td></tr>
<tr><td>15-14<br>—</td><td>Reserved. Must be 0.</td></tr>
<tr><td>13<br>C2K</td><td>Class 2 Key. This bit is ignored for all algorithms other than AES.<br>0b - AES will use the Class 1 Key for CCM and GCM modes.<br>1b - AES will use the Class 2 Key for CCM and GCM modes. Setting this bit =1 will result in a mode error for other AES modes.</td></tr>
<tr><td>12-4<br>AAI</td><td>Additional Algorithm information. This field contains additional mode information that is associated with the algorithm that is being executed. See also the section describing the appropriate CHA. For RNG OPERATION commands the AAI field is interpreted as shown in CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG).<br>**NOTE:** Some algorithms do not require additional algorithm information and in those cases this field should be all 0s.<br>AAI Interpretation for AES Modes
<table>
<tr><th colspan="5">[For AES the MSB of AAI is the DK (Decrypt Key) bit.]</th></tr>
<tr><th>Code[1]</th><th>Interpretation</th><th></th><th>Code[1]</th><th>Interpretation</th></tr>
<tr><td>00h</td><td>CTR (mod $2^{128}$)</td><td></td><td>80h</td><td>CCM[2]</td></tr>
<tr><td>10h</td><td>CBC</td><td></td><td>90h</td><td>GCM[2]</td></tr>
<tr><td>20h</td><td>ECB</td><td></td><td>A0h</td><td>CBC_XCBC_MAC</td></tr>
<tr><td>30h</td><td>CFB</td><td></td><td>B0h</td><td>CTR_XCBC_MAC</td></tr>
<tr><td>40h</td><td>OFB</td><td></td><td>C0h</td><td>CBC_CMAC</td></tr>
<tr><td>50h</td><td>XTS</td><td></td><td>D0h</td><td>CTR_CMAC_LTE</td></tr>
<tr><td>60h</td><td>CMAC</td><td></td><td>E0h</td><td>CTR_CMAC</td></tr>
<tr><td>70h</td><td>XCBC-MAC</td><td colspan="3"></td></tr>
<tr><td colspan="5">Setting the DK bit (i.e. ORing 100h with any AES code above) causes Key Register to be loaded with the AES Decrypt key, rather than the AES Encrypt key. See the discussion in Differences between the AES encrypt and decrypt keys</td></tr>
</table>
1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).</td></tr>
</table>

*Table continues on the next page...*

| Field | Function |
|---|---|
| | 2. If C2K= 0, CCM and GCM use the key in the Class 1 Key Register. If C2K = 1, CCM and GCM use the key in the Class 2 Key Register. |

### Table 13-4. AAI Interpretation for Kasumi

| Code[1] | Interpretation | | Code[1] | Interpretation |
|---|---|---|---|---|
| 10h | GSM | | 20h | EDGE |
| C08 | f8 | | C8h | f9 |

1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).

### Table 13-5. AAI Interpretation for ZUC

| Code[1] | Interpretation | For Authentication mode,see CCB a Class 2 Mode Register (C0C2MR - C2C2MR) |
|---|---|---|
| C0h | encryption mode | |

1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).

Additional Algorithm information. This field contains additional mode information that is associated with the algorithm that is being executed. See also the section describing the appropriate CHA.

**NOTE:** Some algorithms do not require additional algorithm information and in those cases this field should be all 0s.

**NOTE:** For RNG OPERATION commands the AAI field is interpreted as shown in the SK, AI, PS, OBP, NZ and SH fields shown in CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG).

### Table 13-6. AAI Interpretation for DES

| Code[1] | Interpretation | | Code[1] | Interpretation |
|---|---|---|---|---|
| 10h | CBC | | 30h | CFB |
| 20h | ECB | | 40h | OFB |
| 80h ORed with any DES code above: Check odd parity | | | | |

1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).

### Table 13-7. AAI Interpretation for RNG

| Code[1] | Interpretation |
|---|---|
| 00h | Random Numbers |
| 10h | Random Numbers with No Zero Bytes |
| 20h | Random Numbers with odd byte parity |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td>Table 13-8. AAI Interpretation for SNOW 3G<br><table><tr><th>Code[1]</th><th>Interpretation</th><th></th><th>For f9 mode, see CCB a Class 2 Mode Register (C0C2MR - C2C2 MR)</th></tr><tr><td>C0h</td><td>f8</td><td></td><td></td></tr></table><br>1. The codes are mutually exclusive (i.e. they cannot be ORed with each other).</td></tr>
<tr><td>3-2<br>AS</td><td>Algorithm State. This field defines the state of the algorithm that is being executed. This may not be used by every algorithm. For RNG commands, see CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG).<br>00b - Update<br>01b - Initialize<br>10b - Finalize<br>11b - Initialize/Finalize</td></tr>
<tr><td>1<br>ICV_TEST</td><td>ICV Checking / Test AES fault detection.<br>(This is the definition of this bit for CHAs other than RNG. For the definition of this bit in RNG commands, see CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG))<br>For algorithms other than AES ECB mode: ICV Checking<br>This bit selects whether the current algorithm should compare the known ICV versus the calculated ICV. This bit will be ignored by algorithms that do not support ICV checking.<br>0 - Don't compare<br>1 - Compare<br>For AES ECB mode: Test AES fault detection<br>In AES ECB mode, this bit activates fault detection testing by injecting bit level errors into AES core logic as defined in the first 128 bits of the context.<br>0 - Don't inject bit errors<br>1 - Inject bit errors</td></tr>
<tr><td>0<br>ENC</td><td>Encrypt/Decrypt.<br>(This is the definition of this bit for CHAs other than RNG.. For the definition of this bit in RNG commands, see CCB a Class 1 Mode Register Format for RNG4 (C0C1MR_RNG - C2C1MR_RNG).)<br>This bit selects encryption or decryption. This bit is ignored by all algorithms that do not have distinct encryption and decryption modes. However, for performance counting to be done correctly, this bit must be set appropriately even if the CHA or Algorithm does not use it to select cryptographic modes.<br>0b - Decrypt.<br>1b - Encrypt.</td></tr>
</table>

## 13.182 CCB a Class 1 Key Size Register (C0C1KSR - C2C1KSR)

## 13.182.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC1KSR | 8_000Ch + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.182.2 Function

The Class 1 Key Size Register is used to tell the Class 1 CHA the size of the key that was loaded into the Class 1 Key Register. The Class 1 Key Size Register must be written after the key is written into the Class 1 Key Register. Writing to the Class 1 Key Size Register will prevent the user from modifying the Class 1 Key Register. The Class 1 Key Size Register is automatically written by the KEY Command except in the following cases. When the PKHA E-RAM is loaded the PKHA E Size Register is automatically loaded with the correct size, rather than loading the Class 1 Key Size Register.

## 13.182.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | C1KS | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.182.4 Fields

| Field | Function |
|---|---|
| 31-7<br>— | Reserved |
| 6-0 | Class 1 Key Size. This is the size of a Class 1 Key measured in bytes |

| Field | Function |
|---|---|
| C1KS | Although the Class 1 Key Register holds only 32 bytes, it is possible to load a key as large as 9688 bytes. The first 32 bytes will be loaded into the Class 1 Key Register and the remaining bytes will be loaded into the Class 1 Context Register. The key bytes loaded into the Context Register will be treated as an "Extended Key Register" (i.e. as part of the Key Register) during cryptographic operations and when storing and loading Black Keys. |

## 13.183 CCB a Class 1 Data Size Register (C0C1DSR - C2C1 DSR)

### 13.183.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC1DSR | 8_0010h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.183.2 Function

The Class 1 Data Size Register is used to tell the Class 1 CHA the amount of data that will be loaded into the Input Data FIFO. For bit-oriented operations, the value in the NUMBITS field is appended to the C1CY and C1DS fields to form a data size that is measured in bits. Note that writing to the C1DS field in this register causes the written value to be added to the previous value in that field. That is, if the C1DS field currently has the value 14, writing 2 to the least-significant half of the Class 1 Data Size register (i.e. the C1DS field) will result in a value of 16 in the C1DS field. Although there is a C1CY field to hold the carry from this addition, care must be taken to avoid overflowing the 33-bit value held in the concatenation of the C1CY and C1DS fields. Any such overflow will be lost. Note that some CHAs decrement this register, so reading the register may return a value less than sum of the values that were written into it. FIFO LOAD commands can automatically load this register when automatic iNformation FIFO entries are enabled. This register is cleared whenever a key is decrypted or encrypted. Since the Class 1 Data Size Registers hold more than 32 bits, they are accessed from the IP bus as two 32-bit registers.

## 13.183.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | NUMBITS | | | Reserved | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | C1CY |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | C1DS | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | C1DS | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.183.4 Fields

| Field | Function |
|---|---|
| 63-61<br>NUMBITS | Class 1 Data Size Number of bits. For bit-oriented operations, this value is appended to the C1CY and C1DS fields to form a data size that is measured in bits. That is, the number of bits of data is given by the value ( C1CY ǁ C1DS ǁ NUMBITS). Note that if NUMBITS is nonzero, C1DS +1 bytes will be written to the Input Data FIFO, but only NUMBITS bits of the last byte will be consumed by the bit-oriented operation. Note that the NUMBITS field is not additive, so any write to the field will overwrite the previous value. |
| 60-33<br>— | Reserved |
| 32<br>C1CY | Class 1 Data Size Carry. Although this field is not writable, it will be set if a write to C1DS causes a carry out of the msb of C1DS. |
| 31-0<br>C1DS | Class 1 Data Size. This is the number of whole bytes of data that will be consumed by the Class 1 CHA. Note that one additional byte will be written into the Input Data FIFO if the NUMBITS field is nonzero. |

# 13.184 CCB a Class 1 ICV Size Register (C0C1ICVSR - C2C1 ICVSR)

## 13.184.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC1ICVSR | 8_001Ch + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.184.2 Function

The Class 1 ICV Size Register indicates how much of the last block of ICV is valid when performing AES integrity check modes (e.g. AES-CMAC, AES-XCBC-MAC, AES-GCM). Like the Class 1 Data Size register, the Class 1 ICV Size register is additive. That is, any value written to the C1ICVS field will be added to the previous value in the field. This register must be written prior to the corresponding word of data being consumed by AES. In practical terms, this means the register must be written either prior to the corresponding data being written to the Input Data FIFO or prior to the iNformation FIFO entry for this data. FIFO LOAD commands can automatically load it when ICV is loaded.

## 13.184.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | C1ICVS | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.184.4 Fields

| Field | Function |
|---|---|
| 31-5<br>— | Reserved |
| 4-0<br>C1ICVS | Class 1 ICV Size, in Bytes. |

## 13.185 CCB a CHA Control Register (C0CCTRL - C2CCTRL)

### 13.185.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaCCTRL | 8_0034h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.185.2 Function

The CHA Control Register is used to send control signals to the CHAs. This register is automatically written between Descriptors. Within a Descriptor, use the LOAD Command to reset blocks or unload memories.

### 13.185.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | Reserved | Reserved | | | Reserved | | | | | | | | | |
| W | | | | | UB | UA | | UN | UB3 | UB2 | UB1 | UB0 | UA3 | UA2 | UA1 | UA0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | Reserved | | | | | | | | | | | Reserved | | | |
| W | | | AES_C2 | ZUCA | ZUCE | SNF9 | RNG | CRC | MD | PK | SNF8 | KAS | | DES | AES | ALL |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.185.4 Fields

| Field | Function |
|---|---|
| 31-30<br>— | Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits. |
| 29<br>— | Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits. |
| 28<br>— | Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits. |
| 27<br>UB | Unload the PKHA B Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B memory into the Output Data FIFO. |
| 26<br>UA | Unload the PKHA A Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A memory into the Output Data FIFO. |
| 25<br>— | Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits. |
| 24<br>UN | Unload the PKHA N Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the N memory into the Output Data FIFO. |
| 23<br>UB3 | Unload the PKHA B3 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B3 memory into the Output Data FIFO. |
| 22<br>UB2 | Unload the PKHA B2 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B2 memory into the Output Data FIFO. |
| 21<br>UB1 | Unload the PKHA B1 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B1 memory into the Output Data FIFO. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 20<br>UB0 | Unload the PKHA B0 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the B0 memory into the Output Data FIFO. |
| 19<br>UA3 | Unload the PKHA A3 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A3 memory into the Output Data FIFO. |
| 18<br>UA2 | Unload the PKHA A2 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A2 memory into the Output Data FIFO. |
| 17<br>UA1 | Unload the PKHA A1 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A1 memory into the Output Data FIFO. |
| 16<br>UA0 | Unload the PKHA A0 Memory. Writing a 1 to this bit causes the PKHA to unload the contents of the A0 memory into the Output Data FIFO. |
| 15<br>— | Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits. |
| 14<br>— | Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits. |
| 13<br>AES_C2 | Reset AES Class 2 CHA. Writing a 1 to this bit resets the AES Authentication (Class 2) accelerator.<br>0b - Do Not Reset<br>1b - Reset AES Authentication accelerator |
| 12<br>ZUCA | Reset ZUCA. Writing a 1 to this bit resets the ZUC Authentication accelerator.<br>0b - Do Not Reset<br>1b - Reset ZUC Authentication accelerator |
| 11<br>ZUCE | Reset ZUCE. Writing a 1 to this bit resets the ZUC Encryption accelerator.<br>0b - Do Not Reset<br>1b - Reset ZUC Encryption accelerator |
| 10<br>SNF9 | Reset SNOW f9. Writing a 1 to this bit resets the SNOW f9 Accelerator.<br>0b - Do Not Reset<br>1b - Reset SNOW f9 Accelerator |
| 9<br>RNG | Reset Random Number Generator. Writing a 1 to this bit resets the Random Number Generator.<br>0b - Do Not Reset<br>1b - Reset Random Number Generator Block. |
| 8<br>CRC | Reset CRCA. Writing a 1 to this bit resets the CRC Accelerator.<br>0b - Do Not Reset<br>1b - Reset CRC Accelerator |
| 7<br>MD | Reset MDHA. Writing a 1 to this bit resets the Message Digest Hardware Accelerator.<br>0b - Do Not Reset<br>1b - Reset Message Digest Hardware Accelerator |
| 6<br>PK | Reset PKHA. Writing a 1 to this bit resets the Public Key Hardware Accelerator.<br>0b - Do Not Reset<br>1b - Reset Public Key Hardware Accelerator |
| 5<br>SNF8 | Reset SNOW f8. Writing a 1 to this bit resets the SNOW f8 Hardware Accelerator.<br>0b - Do Not Reset<br>1b - Reset SNOW f8 Accelerator |
| 4 | Reset KFHA. Writing a 1 to this bit resets the Kasumi f8/f9 Hardware Accelerator. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| KAS | 0b - Do Not Reset<br>1b - Reset Kasumi f8/f9 Accelerator |
| 3<br>— | Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits. |
| 2<br>DES | Reset DESA. Writing a 1 to this bit resets the DES Accelerator.<br>0b - Do Not Reset<br>1b - Reset DES Accelerator |
| 1<br>AES | Reset AESA. Writing a 1 to this bit resets the AES Accelerator.<br>0b - Do Not Reset<br>1b - Reset AES Accelerator |
| 0<br>ALL | Reset All Internal CHAs. Writing to this bit resets all CHAs in use by this CCB.<br>0b - Do Not Reset<br>1b - Reset all CHAs in use by this CCB. |

# 13.186 CCB a Interrupt Control Register (C0ICTL - C2ICTL)

## 13.186.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaICTL | 8_003Ch + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.186.2 Function

The IRQ Control Register shows the status of all CCB "done" interrupts and "error" interrupts and provides controls for clearing these interrupts.

## 13.186.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | ZAEI | ZEE_ | S9E_ | RNE_ | CE_ | MEI | PE_ | S8E_ | KEI | Reserved | DE_ | AEI | Reserved |
| W | | | | W1C | W1C | W1C | W1C | W1C | W1C | W1C | W1C | W1C | | W1C | W1C | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | ZADI | ZEDI | S9DI | RNDI | CDI | MDI | PDI | S8DI | KDI | Reserved | DDI | ADI | Reserved |
| W | | | | W1C | W1C | W1C | W1C | W1C | W1C | W1C | W1C | W1C | | W1C | W1C | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.186.4 Fields

| Field | Function |
|---|---|
| 31-29<br>— | Reserved. To preserve software compatibility with other versions of SEC, 0 should be written to all reserved bits (shaded or marked “RSV”) |
| 28<br>ZAEI | ZUCA error Interrupt asserted.<br><table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>ZUCA Error Interrupt asserted</td><td>Clear the ZUCA Error Interrupt</td></tr></table> |
| 27<br>ZEEI | ZUCE error Interrupt asserted.<br><table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>ZUCE Error Interrupt asserted</td><td>Clear the ZUCE Error Interrupt</td></tr></table> |
| 26<br>S9EI | SNOW-f9 error Interrupt asserted.<br><table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>SNOW-f9 Error Interrupt asserted</td><td>Clear the SNOW-f9 Error Interrupt</td></tr></table> |
| 25 | RNG error Interrupt asserted. |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>RNEI</td><td><table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>RNG Error Interrupt asserted</td><td>Clear the RNG Error Interrupt</td></tr></table></td></tr>
<tr><td>24<br>CEI</td><td>CRCA error Interrupt asserted.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>CRC Error Interrupt asserted</td><td>Clear the CRC Error Interrupt</td></tr></table></td></tr>
<tr><td>23<br>MEI</td><td>MDHA (hashing) error Interrupt asserted.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>MDHA Error Interrupt asserted</td><td>Clear the MDHA Error Interrupt</td></tr></table></td></tr>
<tr><td>22<br>PEI</td><td>Public Key error Interrupt asserted.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>PKHA Error Interrupt asserted</td><td>Clear the PKHA Error Interrupt</td></tr></table></td></tr>
<tr><td>21<br>S8EI</td><td>SNOW-f8 error asserted.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>SNOW-f8 Error Interrupt asserted</td><td>Clear the SNOW-f8 Error Interrupt</td></tr></table></td></tr>
<tr><td>20<br>KEI</td><td>KFHA (Kasumi) error Interrupt asserted.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>KFHA Error Interrupt asserted</td><td>Clear the KFHA Error Interrupt</td></tr></table></td></tr>
<tr><td>19<br>—</td><td>Reserved</td></tr>
<tr><td>18<br>DEI</td><td>DESA error Interrupt asserted.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr></table></td></tr>
</table>

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td><table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>1</td><td>DESA Error Interrupt asserted</td><td>Clear the DESA Error Interrupt</td></tr></table></td></tr>
<tr><td>17<br>AEI</td><td>AESA error Interrupt asserted.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error Write</td><td>No change</td></tr><tr><td>1</td><td>AESA Error Interrupt asserted</td><td>Clear the AESA Error Interrupt</td></tr></table></td></tr>
<tr><td>16-13<br>—</td><td>Reserved</td></tr>
<tr><td>12<br>ZADI</td><td>ZUCA Done interrupt.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error</td><td>No change</td></tr><tr><td>1</td><td>ZUCA Done Interrupt asserted</td><td>Clear the ZUCA Done Interrupt</td></tr></table></td></tr>
<tr><td>11<br>ZEDI</td><td>ZUCE Done interrupt.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Error</td><td>No change</td></tr><tr><td>1</td><td>ZUCE Error Interrupt asserted</td><td>Clear the ZUCE Done Interrupt</td></tr></table></td></tr>
<tr><td>10<br>S9DI</td><td>SNOW-f9 done interrupt.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr><tr><td>1</td><td>SNOW-f9 Done Interrupt asserted</td><td>Clear the SNOW-f9 Done Interrupt</td></tr></table></td></tr>
<tr><td>9<br>RNDI</td><td>RNG done interrupt.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr><tr><td>1</td><td>RNG Done Interrupt asserted</td><td>Clear the RNG Done Interrupt</td></tr></table></td></tr>
<tr><td>8<br>CDI</td><td>CRCA done interrupt.<table><tr><th>Value</th><th>Read</th><th>Write</th></tr><tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr><tr><td>1</td><td>CRCA Done Interrupt asserted</td><td>Clear the CRCA Done Interrupt</td></tr></table></td></tr>
</table>

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>7<br>MDI</td><td>MDHA (hashing) done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>MDHA Done Interrupt asserted</td><td>Clear the MDHA Done Interrupt</td></tr>
</table></td></tr>
<tr><td>6<br>PDI</td><td>PKHA (Public Key) done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>PKHA Done Interrupt asserted</td><td>Clear the PKHA Done Interrupt</td></tr>
</table></td></tr>
<tr><td>5<br>S8DI</td><td>SNOW-f8 done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>SNOW-f8 Done Interrupt asserted</td><td>Clear the SNOW-f8 Done Interrupt</td></tr>
</table>
Read: 0 No Done Interrupt Write: 0 No change<br>1 SNOW-f8 Done Interrupt asserted 1 Clear the SNOW-f8 Done Interrupt</td></tr>
<tr><td>4<br>KDI</td><td>KFHA (Kasumi) done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>KFHA Done Interrupt asserted</td><td>Clear the KFHA Done Interrupt</td></tr>
</table></td></tr>
<tr><td>3<br>—</td><td>Reserved</td></tr>
<tr><td>2<br>DDI</td><td>DESA done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>DESA Done Interrupt asserted</td><td>Clear the DESA Done Interrupt</td></tr>
</table></td></tr>
<tr><td>1<br>ADI</td><td>AESA done interrupt.
<table>
<tr><th>Value</th><th>Read</th><th>Write</th></tr>
<tr><td>0</td><td>No Done Interrupt</td><td>No change</td></tr>
<tr><td>1</td><td>AESA Done Interrupt asserted</td><td>Clear the AESA Done Interrupt</td></tr>
</table></td></tr>
<tr><td>0</td><td>Reserved</td></tr>
</table>

| Field | Function |
|---|---|
| — | |

## 13.187 CCB a Clear Written Register (C0CWR - C2CWR)

### 13.187.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaCWR | 8_0044h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.187.2 Function

The Clear Written Register is used to clear many of the internal registers. This register is automatically written, if necessary, by DECO between Shared Descriptors. All fields of this register are self-clearing.

### 13.187.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | | | | | | Reserved | | | | Reserved | | | Reserved | |
| W | CIF | COF | C1RST | C2RST | C1D | C2D | CDS | | | C2K | C2C | | | C2DS | | C2M |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | | | Reserved | | | | | | | Reserved | | | Reserved | |
| W | CPKE | CPKN | CPKB | CPKA | | | | | | C1K | C1C | | C1ICV | C1DS | | C1M |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.187.4 Fields

| Field | Function |
|---|---|
| 31<br>CIF | Clear Input FIFO (and NFIFO). Writing a 1 to this bit causes the Input Data FIFO and iNformation FIFO to be cleared. |
| 30<br>COF | Clear Output FIFO. Writing a 1 to this bit causes the Output FIFO to be cleared. |
| 29<br>C1RST | Reset Class 1 CHA. Writing a 1 to this bit causes a reset to any Class 1 CHA that is currently selected by this DECO. |
| 28<br>C2RST | Reset Class 2 CHA. Writing a 1 to this bit causes a reset to any Class 2 CHA that is currently selected by this DECO. |
| 27<br>C1D | Clear Class 1 Done Interrupt. Writing a 1 to this bit clears the Class 1 done interrupt. |
| 26<br>C2D | Clear Class 2 Done Interrupt. Writing a 1 to this bit clears the Class 2 done interrupt. |
| 25<br>CDS | Clear Descriptor Sharing signal. Writing a 1 to this bit clears the shared_descriptor signal in DECO. This signal tells DECO, and the protocols, whether this descriptor was shared from a previous run. If CDS is set via LOAD IMM to the Clear Written register the fact that this descriptor was shared will be forgotten and the descriptor will behave thereafter as if it was not shared. This is important in protocols where the protocol expects a "decrypt" key but an "encrypt" key is provided. This may occur when using RJD to re-key a flow. Note that writing 1 to this bit when the DECO/CCB is under direct software control will not clear sharing, but that is unimportant because sharing is not possible when the DECO is under direct software control. |
| 24-23<br>— | Reserved |
| 22<br>C2K | Clear the Class 2 Key Register. Writing a one to this bit causes the Class 2 Key and Key Size Registers to be cleared. |
| 21<br>C2C | Clear the Class 2 Context Register. Writing a one to this bit causes the Class 2 Context Register to be cleared. |
| 20-19<br>— | Reserved |
| 18<br>C2DS | Clear the Class 2 Data Size Registers. Writing a one to this bit causes the Class 2 Data Size and ICV Size Registers to be cleared. |
| 17<br>— | Reserved |
| 16<br>C2M | Clear the Class 2 Mode Register. Writing a one to this bit causes the Class 2 Mode Register to be cleared. |
| 15<br>CPKE | Clear the PKHA E Size Register. Writing a one to this bit causes the PKHA E Size Register to be cleared. |
| 14<br>CPKN | Clear the PKHA N Size Register. Writing a one to this bit causes the PKHA N Size Register to be cleared. |
| 13 | Clear the PKHA B Size Register. Writing a one to this bit causes the PKHA B Size Register to be cleared. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| CPKB | |
| 12<br>CPKA | Clear the PKHA A Size Register. Writing a one to this bit causes the PKHA A Size Register to be cleared. |
| 11-7<br>— | Reserved |
| 6<br>C1K | Clear the Class 1 Key Register. Writing a one to this bit causes the Class 1 Key and Key Size Registers to be cleared. |
| 5<br>C1C | Clear the Class 1 Context Register. Writing a one to this bit causes the Class 1 Context Register to be cleared. |
| 4<br>— | Reserved |
| 3<br>C1ICV | Clear the Class 1 ICV Size Register. Writing a one to this bit causes the Class 1 ICV Size Register to be cleared. |
| 2<br>C1DS | Clear the Class 1 Data Size Register. Writing a one to this bit causes the Class 1 Data Size Register to be cleared. This clears AAD Size as well. |
| 1<br>— | Reserved |
| 0<br>C1M | Clear the Class 1 Mode Register. Writing a one to this bit causes the Class 1 Mode Register to be cleared. |

## 13.188 CCB a Status and Error Register, most-significant half (C0CSTA_MS - C2CSTA_MS)

### 13.188.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaCSTA_MS | 8_0048h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.188.2 Function

The CCB Status and Error Register shows the status of the CCB and its internal registers. The fields of the CaCSTA are accessed as two 32-bit words.

## 13.188.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | CL2 | | | | Reserved | | | | | | | | ERRID2 | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | CL1 | | | | Reserved | | | | | | | | ERRID1 | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.188.4 Fields

| Field | Function |
|---|---|
| 31-28<br>CL2 | Class 2 Algorithms. The Class 2 Algorithms bits indicate which algorithm is asserting an error.<br>Others reserved.<br>0100b - MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 and SHA-512/224, SHA-512/256<br>1001b - CRC<br>1010b - SNOW f9<br>1100b - ZUC Authentication |
| 27-20<br>— | Reserved |
| 19-16<br>ERRID2 | Error ID 2. These bits indicate the type of error that was found while processing the Descriptor. The Algorithm that is associated with the error can be found in the CL2 field<br>Others Reserved<br>0001b - Mode Error<br>0010b - Data Size Error<br>0011b - Key Size Error<br>0110b - Data Arrived out of Sequence Error<br>1010b - ICV Check Failed<br>1011b - Internal Hardware Failure<br>1110b - Invalid CHA combination was selected.<br>1111b - Invalid CHA Selected |
| 15-12<br>CL1 | Class 1 algorithms. The Class 1 algorithms field indicates which algorithm is asserting an error.<br>Others reserved<br>0001b - AES<br>0010b - DES<br>0101b - RNG<br>0110b - SNOW<br>0111b - Kasumi<br>1000b - Public Key<br>1011b - ZUC Encryption |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 11-4<br>— | Reserved |
| 3-0<br>ERRID1 | Error ID 1. These bits indicate the type of error that was found while processing the Descriptor. The Algorithm that is associated with the error can be found in the CL1 field.<br>Others reserved.<br>0001b - Mode Error<br>0010b - Data Size Error, including PKHA N Memory Size Error<br>0011b - Key Size Error, including PKHA E Memory Size Error<br>0100b - PKHA A Memory Size Error<br>0101b - PKHA B Memory Size Error<br>0110b - Data Arrived out of Sequence Error<br>0111b - PKHA Divide by Zero Error<br>1000b - PKHA Modulus Even Error<br>1001b - DES Key Parity Error<br>1010b - ICV Check Failed<br>1011b - Internal Hardware Failure<br>1100b - CCM AAD Size Error (either 1. AAD flag in B0 =1 and no AAD type provided, 2. AAD flag in B0 = 0 and AAD provided, or 3. AAD flag in B0 =1 and not enough AAD provided - expecting more based on AAD size.)<br>1101b - Class 1 CHA is not reset<br>1110b - Invalid CHA combination was selected.<br>1111b - Invalid CHA Selected |

# 13.189 CCB a Status and Error Register, least-significant half (C0CSTA_LS - C2CSTA_LS)

## 13.189.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaCSTA_LS | 8_004Ch + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.189.2 Function

The CCB Status and Error Register shows the status of the CCB and its internal registers. The fields of the CaCSTA are accessed as two 32-bit words.

### 13.189.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | PIZ | GCD | PRM | Reserved | | | | | | SEI | PEI | Reserved | | SDI | PDI |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | ZAB | ZEB | S9B | RNB | CB | MB | PB | S8B | KB | Reserved | DB | AB | Reserved |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.189.4 Fields

| Field | Function |
|---|---|
| 31<br>— | Reserved |
| 30<br>PIZ | Public Key Operation is Zero. For Finite Field operations the result of a Public Key operation is zero. For ECC operations, the result is Point at infinity. |
| 29<br>GCD | GCD is One. The greatest common divisor of two numbers is one (that is, the two numbers are relatively prime). |
| 28<br>PRM | Public Key is Prime. The given number is probably prime (that is, it passes the Miller-Rabin primality test). |
| 27-22<br>— | Reserved |
| 21<br>SEI | Class 2 Error Interrupt. The Class 2 Error Interrupt has been asserted.<br>0b - No Error.<br>1b - Error Interrupt. |
| 20<br>PEI | Class 1 Error Interrupt. The Class 1 Error Interrupt has been asserted.<br>0b - Not Error.<br>1b - Error Interrupt. |
| 19-18<br>— | Reserved |
| 17<br>SDI | Class 2 Done Interrupt. The Class 2 Done Interrupt has been asserted.<br>0b - Not Done.<br>1b - Done Interrupt. |
| 16 | Class 1 Done Interrupt. The Class 1 Done Interrupt has been asserted. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| PDI | 0b - Not Done.<br>1b - Done Interrupt. |
| 15-13<br>— | Reserved |
| 12<br>ZAB | ZUCA Block Busy. This bit indicates that the ZUCA is busy. The CHA can either be busy processing data or resetting.<br>0b - ZUCA idle<br>1b - ZUCA busy. |
| 11<br>ZEB | ZUCE Block Busy. This bit indicates that the ZUCE is busy. The CHA can either be busy processing data or resetting.<br>0b - ZUCE idle<br>1b - ZUCE busy. |
| 10<br>S9B | SNOW f9 Busy. This bit indicates that the SNOW f9 Accelerator is busy. The CHA can either be busy processing data or resetting.<br>0b - SNOW f9 Idle<br>1b - SNOW f9 Busy. |
| 9<br>RNB | RNG Block Busy. This bit indicates that the RNG block is busy. The CHA can either be busy processing data or resetting.<br>0b - RNG Idle<br>1b - RNG Busy. |
| 8<br>CB | CRC Block Busy. This bit indicates that the CRCA is busy. The CHA can either be busy processing data or resetting.<br>0b - CRCA Idle<br>1b - CRCA Busy |
| 7<br>MB | MDHA Busy. This bit indicates that the MDHA is busy. The CHA can either be busy processing data or resetting.<br>0b - MDHA Idle<br>1b - MDHA Busy |
| 6<br>PB | PKHA Busy. This bit indicates that the Public Key Hardware Accelerator is busy. The CHA can either be busy processing data or resetting.<br>0b - PKHA Idle<br>1b - PKHA Busy. |
| 5<br>S8B | SNOW f8. This bit indicates that the SNOW f8 Accelerator is busy. The CHA can either be busy processing data or resetting.<br>0b - SNOW f8 Idle<br>1b - SNOW f8 Busy. |
| 4<br>KB | KFHA Busy. This bit indicates that the Kasumi f8/f9 Hardware Accelerator is busy. The CHA can either be busy processing data or resetting.<br>0b - KFHA Idle<br>1b - KFHA Busy |
| 3<br>— | Reserved |
| 2<br>DB | DESA Busy. This bit indicates that the DES Accelerator is busy. The CHA can either be busy processing data or resetting.<br>0b - DESA Idle<br>1b - DESA Busy. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 1<br>AB | AESA Busy. This bit indicates that the AES Accelerator is busy. The CHA can either be busy processing data or resetting.<br>0b - AESA Idle<br>1b - AESA Busy. |
| 0<br>— | Reserved |

## 13.190 CCB a AAD Size Register (C0AADSZR - C2AADSZR)

### 13.190.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaAADSZR | 8_005Ch + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.190.2 Function

The AAD Size Register is used by AESA to determine how much of the last block of AAD is valid. Like the Class 1 Data Size Register, writing to this register causes the written value to be added to the previous value in the register. The register is automatically written by FIFO LOAD commands.

### 13.190.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | AASZ | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.190.4 Fields

| Field | Function |
|---|---|
| 31-4<br>— | Reserved |
| 3-0<br>AASZ | AAD size in Bytes, mod 16. |

## 13.191 Class 1 IV Size Register (C0C1IVSZR - C2C1IVSZR)

### 13.191.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC1IVSZR | 8_0064h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.191.2 Function

The Class 1 IV Size Register tells the AES module how much of the last block of IV is valid. Like the Class 1 Data Size Register, writing to this register causes the written value to be added to the previous value in the register. The register is automatically written by FIFO LOAD commands.

### 13.191.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | IVSZ | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.191.4 Fields

| Field | Function |
|---|---|
| 31-4<br>— | Reserved |
| 3-0<br>IVSZ | IV size in bytes, mod 16. |

## 13.192 PKHA A Size Register (C0PKASZR - C2PKASZR)

### 13.192.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaPKASZR | 8_0084h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.192.2 Function

The PKHA A Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA A Memory. The PKHA A Size Register must be written before the data is written into or read from the PKHA A Memory. This will reserve the PKHA for the current job. The PKHA A Size Register can be automatically written by the MOVE, FIFO LOAD and FIFO STORE commands.

### 13.192.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | PKASZ | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.192.4 Fields

| Field | Function |
|---|---|
| 31-10<br>— | Reserved |
| 9-0<br>PKASZ | PKHA A Memory key size in bytes. |

## 13.193 PKHA B Size Register (C0PKBSZR - C2PKBSZR)

### 13.193.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaPKBSZR | 8_008Ch + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.193.2 Function

The PKHA B Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA B Memory. The PKHA B Size Register must be written before the data is written into or read from the PKHA B Memory. This will reserve the PKHA for the current job. The PKHA B Size Register can be automatically written by the FIFO LOAD and FIFO STORE commands.

### 13.193.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | PKBSZ | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.193.4 Fields

| Field | Function |
|---|---|
| 31-10<br>— | Reserved |
| 9-0<br>PKBSZ | PKHA B Memory key size in bytes. |

## 13.194 PKHA N Size Register (C0PKNSZR - C2PKNSZR)

### 13.194.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaPKNSZR | 8_0094h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.194.2 Function

The PKHA N Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA N Memory. The PKHA N Size Register must be written before the data is written into or read from the PKHA N Memory. This will reserve the PKHA for the current job. The PKHA N Size Register can be automatically written by the FIFO LOAD and FIFO STORE commands.

### 13.194.3 Diagram

<table>
<tr><td>Bits</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td>R<br>W</td><td colspan="16">Reserved</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

<table>
<tr><td>Bits</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>R<br>W</td><td colspan="6">Reserved</td><td colspan="10">PKNSZ</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

### 13.194.4 Fields

| Field | Function |
|---|---|
| 31-10<br>— | Reserved |
| 9-0<br>PKNSZ | PKHA N Memory key size in bytes. |

# 13.195 PKHA E Size Register (C0PKESZR - C2PKESZR)

## 13.195.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaPKESZR | 8_009Ch + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.195.2 Function

The PKHA E Size Register is used to indicate the size of the data that will be loaded into or unloaded from the PKHA E Memory. The PKHA E Size Register must be written before the data is written into or read from the PKHA E Memory. This will reserve the PKHA for the current job. The PKHA E Size Register is automatically written by the KEY Command.

## 13.195.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | PKESZ | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.195.4 Fields

| Field | Function |
|---|---|
| 31-10 | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 9-0<br>PKESZ | PKHA E Memory key size in bytes. |

# 13.196 CCB a Class 1 Context Register Word b (C0C1CTXR0 - C2C1CTXR15)

## 13.196.1 Offset

For a = 0 to 2; b = 0 to 15:

| Register | Offset | Description |
|---|---|---|
| CaC1CTXRb | 8_0100h + (a × 1_0000h) + (b × 4h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.196.2 Function

The Class 1 Context Register holds the context for the Class 1 CHAs. This register is 512 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 1 Context Register is accessible only as full-word reads or writes to sixteen 32-bit registers. The MSB is located at offset 0100h with respect to the register page. This register is cleared automatically when a Black Key is being encrypted or decrypted using AES-CCM.

Note that part of the Class 1 Context Register is also used as an "Extended Key Register" when a key larger than 32 bytes is loaded into the Class 1 Key Register. As many 8-byte chunks of the Class 1 Context Register as required are allocated to the Extended Key Register, beginning with the most-significant byte of the register. The Extended Key Register bytes cannot be overwritten and will return 0 when read. The remaining bytes are still available for context data. Note that clearing the Class 1 Context Register will also clear the Extended Key Register.

Note that some commands must block until a previous load to the Class 1 Context Register has completed. Loading the Class 1 Context Register, whether via the KEY Command, LOAD Command or MOVE Command or as a consequence of its usage as the Extended Key Register, sets an internal blocking flag until the Class 1 Context Register load has completed.

The bit assignments of this register are dependent on the algorithm, and in some cases the mode of that algorithm. See the appropriate section for the Context Register format used for that algorithm:

- AES ECB: Section AES ECB mode use of the Context Register
- AES CBC, OFB and CFB128: Section AES CBC, OFB, and CFB128 modes use of the Context Register
- AES CTR: Section AES CTR mode use of the Context Register
- AES XTS: Section AES XTS mode use of the Context Register
- AES XCBC-MAC, CMAC: Section AES XCBC-MAC and CMAC Modes use of the Context Register
- AES CCM: Section AES CCM mode use of the Context Register
- AES GCM: Section AES GCM mode use of the Mode Register
- AES Optimization modes: AES optimization modes use of the Context Register
- DES: Section DESA Context Register
- Kasumi f8/f9: Section KFHA use of the Context Register
- Random Numbers: Section RNG use of the Context Register
- SNOW 3G f8: Section SNOW 3G f8 use of the Context Register
- SNOW 3G f9: Section SNOW 3G f9 use of the Context Register
- Triple DES: Section DESA Context Register
- ZUC Authentication: Section ZUCA use of the Context Register
- ZUC Encryption: Section ZUCE use of the Context Register

## 13.196.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | C1CTX | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | C1CTX | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.196.4 Fields

| Field | Function |
|---|---|
| 31-0<br>C1CTX | Class 1 Context. |

## 13.197 CCB a Class 1 Key Registers Word b (C0C1KR0 - C2C1KR7)

### 13.197.1 Offset

For a = 0 to 2; b = 0 to 7:

| Register | Offset | Description |
|---|---|---|
| CaC1KRb | 8_0200h + (a × 1_0000h) + (b × 4h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.197.2 Function

The Class 1 Key Register normally holds the left-aligned key for the Class 1 CHAs. The MSB is in offset 200h. The Class 1 Key Register is 256 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 1 Key Register is accessible only as full-word reads or writes to eight 32-bit registers. Although the Class 1 Key Register is only 32 bytes long, via the KEY command it is possible to load a key larger than 32 bytes. In this case part of the Class 1 Context Register is used as an "Extended Key Register". The first 32 bytes are loaded into the Class 1 Key Register, then, starting with the most-significant end, as many 8-byte chunks of the Class 1 Context Register as required are allocated to the Extended Key Register and are used to hold the remaining key bytes. The Extended Key Register bytes cannot be overwritten and will return 0 when read. The remaining bytes are still available for context data. Clearing the Class 1 Key Register will also clear the Extended Key Register bytes in the Class 1 Context Register. The other bytes in the Class 1 Context Register will not be cleared. Note that clearing the Class 1 Context Register will not clear the Extended Key Register bytes.

The Class 1 Key Register can be written via a MOVE Command, a MATH Command, a LOAD Command or a KEY Command. Before the value in the Class 1 Key Register can be used in a cryptographic operation, the size of the key must be written into the Class 1 Key Size Register. Once the Class 1 Key Size Register has been written, the Class 1 Key Register cannot be written again until the Class 1 Key Size Register has been cleared. Writing the Class 1 Key Register via a KEY Command automatically writes the Class 1 Key Size Register, but if the Class 1 Key Register is written using a MOVE, MATH or LOAD Command the Class 1 Key Size Register must be written via a separate command after the Class 1 Key Register has been written. But until the Class 1 Key Size Register has been written the Class 1 Key Register remains writable via STORE/SEQ STORE, MATH or MOVE commands and readable via LOAD/SEQ LOAD, MATH or MOVE commands. If the Class 1 Key Size Register and the Class 1 Key Register have been cleared via the Clear Written Register, the Class 1 Key Register becomes writable and readable again. This allows the Class 1 Key Register to be used for temporary storage if it is not currently needed to hold a cryptographic key. Even when the Class 1 Key Register holds a key (i.e. the Class 1 Key Size Register has been written) it may still be possible to store the key in memory in encrypted form. The FIFO STORE Command can be used to store an encrypted copy of this key (i.e. a Black Key), unless storing the key has been prohibited via the NWB bit in the KEY Command. The encrypted key can later be loaded into the Class 1 Key Register via the KEY Command by setting the ENC bit to indicate that this is a Black (i.e. encrypted) Key. The Black Key will automatically be decrypted before it is loaded into the Class 1 Key Register. A Black Key can be loaded as long as the Key Encryption Key (KEK) has not been changed (as a consequence of a security violation or a POR). Note that the Class 1 Key register is cleared when any key (including Class 2 Keys) is encrypted or decrypted, so if a Black Key is to be loaded into or stored from the Class 2 Key Register, that must be done prior to loading a key into the Class 1 Key Register. Similarly, if a key is to be stored from the Class 1 Key Register as a Black Key and also used in a cryptographic operation, the cryptographic operation should be performed first, or the key will have to be loaded a second time.

## 13.197.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | C1KEY | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | C1KEY | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.197.4 Fields

| Field | Function |
|---|---|
| 31-0<br>C1KEY | Class 1 Key. |

## 13.198 CCB a Class 2 Mode Register (C0C2MR - C2C2MR)

### 13.198.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC2MR | 8_0404h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.198.2 Function

The Class 2 Mode Register is used to tell the Class 2 CHA which operation is being requested. The interpretation of this register is unique for each CHA. The Class 2 Mode Register is automatically written by the OPERATION Command. This register is automatically cleared when the signature over a Trusted Descriptor is checked or a Trusted Descriptor is re-signed.

## 13.198.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | ALG | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | AAI | | | | | | | | | AS | | ICV | AP |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.198.4 Fields

| Field | Function |
|---|---|
| 31-24<br>— | Reserved. Must be 0. |
| 23-16<br>ALG | Algorithm. This field specifies which algorithm has been requested for an OPERATION command.<br>00010000b - AES, when used as a Class 2 CHA<br>01000000b - MD5<br>01000001b - SHA-1<br>01000010b - SHA-224<br>01000011b - SHA-256<br>01000100b - SHA-384<br>01000101b - SHA-512<br>01000110b - SHA-512/224<br>01000111b - SHA-512/256<br>10010000b - CRC<br>10100000b - SNOW 3G f9<br>11000000b - ZUC Authentication |
| 15-13<br>— | Reserved. Must be 0. |
| 12-4<br>AAI | Additional Algorithm information. This field contains additional mode information that is associated with the algorithm that is being executed. A detailed list of additional modes can be found below. |

| Value | Description | Valid with ALG |
|---|---|---|
| 001h | IEEE 802 | CRC |
| 002h | IETF 3385 | CRC |
| 004h | CUST_POLY | CRC |
| 010h | DIS | CRC |
| 020h | DOS | CRC |
| 040h | DOC | CRC |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td>
<table>
<tr><th>Value</th><th>Description</th><th>Valid with ALG</th></tr>
<tr><td>001h</td><td>HMAC</td><td>MD5, SHA-*</td></tr>
<tr><td>002h</td><td>SMAC</td><td>MD5, SHA-1</td></tr>
<tr><td>004h</td><td>IPAD OPAD Generation</td><td>MD5, SHA-*</td></tr>
<tr><td>0C8h</td><td>f9</td><td>SNOW 3G</td></tr>
<tr><td>0C8h</td><td>Authentication</td><td>ZUC</td></tr>
<tr><td>060h</td><td>CMAC</td><td>AES</td></tr>
<tr><td>070h</td><td>XCBC-MAC</td><td>AES</td></tr>
<tr><td>Others</td><td>Reserved</td><td></td></tr>
</table>
</td></tr>
<tr><td>3-2<br>AS</td><td>Algorithm State. This field defines the state of the algorithm that is being executed. Not every algorithm uses this field. Check the individual algorithm sections to see if this field is used.<br>00b - Update.<br>01b - Initialize.<br>10b - Finalize.<br>11b - Initialize/Finalize.</td></tr>
<tr><td>1<br>ICV</td><td>ICV Checking. This bit selects whether the current algorithm should compare the known ICV versus the calculated ICV. This bit will be ignored by algorithms that do not support ICV checking.</td></tr>
<tr><td>0<br>AP</td><td>Authenticate / Protect.<br>0b - Authenticate<br>1b - Protect.</td></tr>
</table>

## 13.199 CCB a Class 2 Key Size Register (C0C2KSR - C2C2 KSR)

### 13.199.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC2KSR | 8_040Ch + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.199.2 Function

The Class 2 Key Size Register is used to tell the Class 2 CHA the size of the key that was loaded into the Class 2 Key Register. The Class 2 Key Size Register must be written after the key is written into the Class 2 Key Register. Writing to the Class 2 Key Size Register will prevent the user from modifying the Class 2 Key Register. The Class 2 Key Size Register is automatically written by the Key Command. This register is cleared when Trusted Descriptors are checked or re-signed.

### 13.199.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | C2KS | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.199.4 Fields

| Field | Function |
|---|---|
| 31-8<br>— | Reserved |
| 7-0<br>C2KS | Class 2 key size in bytes. |

## 13.200 CCB a Class 2 Data Size Register (C0C2DSR - C2C2DSR)

### 13.200.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC2DSR | 8_0410h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.200.2 Function

The Class 2 Data Size Register is used to tell the Class 2 CHA the amount of data that will be loaded into the Input Data FIFO. For bit-oriented operations, the value in the NUMBITS field is appended to the C2CY and C2DS fields to form a data size that is measured in bits. Note that writing to the C2DS field in this register causes the written value to be added to the previous value in that field. That is, if the C2DS field currently has the value 14, writing 2 to the least-significant half of the Class 2 Data Size register (i.e. the C2DS field) will result in a value of 16 in the C2DS field. Although there is a C2CY field to hold the carry from this addition, care must be taken to avoid overflowing the 33-bit value held in the concatenation of the C2CY and C2DS fields. Any such overflow will be lost. Note that some CHAs decrement this register, so reading the register may return a value less than sum of the values that were written into it. FIFO LOAD commands can automatically load this register when automatic iNformation FIFO entries are enabled. This register is reset when checking the signature over, or re-signing, Trusted Descriptors. Since the Class 2 Data Size Register holds more than 32 bits, it is accessed from the IP bus as two 32-bit registers.

## 13.200.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | NUMBITS | | | Reserved | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | C2CY |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | C2DS | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | C2DS | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.200.4 Fields

| Field | Function |
|---|---|
| 63-61<br>NUMBITS | Class 2 Data Size Number of bits. For bit-oriented operations, this value is appended to the C2CY and C2DS fields to form a data size that is measured in bits. That is, the number of bits of data is given by the value ( C2CY ǁ C2DS ǁ NUMBITS ). Note that if NUMBITS is nonzero, C2DS +1 bytes will be written to the Input Data FIFO, but only NUMBITS bits of the last byte will be consumed by the bit-oriented operation. Note that the NUMBITS field is not additive, so any write to the field will overwrite the previous value. |
| 60-33<br>— | Reserved |
| 32<br>C2CY | Class 2 Data Size Carry. Although this field is not writable, it will be set if a write to C2DS causes a carry out of the msb of C2DS. |
| 31-0<br>C2DS | Class 2 Data Size in Bytes. This is the number of whole bytes of data that will be consumed by the Class 2 CHA. Note that one additional byte will be written into the Input Data FIFO if the NUMBITS field is nonzero. |

# 13.201 CCB a Class 2 ICV Size Register (C0C2ICVSZR - C2C2 ICVSZR)

## 13.201.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaC2ICVSZR | 8_041Ch + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.201.2 Function

The Class 2 ICV Size Register indicates how much of the last block of ICV is valid when performing MDHA integrity check operations (e.g. SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 and MD5). For AES Class 2 operations the Class 2 ICV Size Register indicates the size of the ICV. Writing to this register causes the written value to be added to the previous value in the register. This register is automatically written by FIFO LOAD commands. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors.

## 13.201.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | ICVSZ | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.201.4 Fields

| Field | Function |
|---|---|
| 31-4<br>— | Reserved |
| 3-0<br>ICVSZ | Class 2 ICV size (mod 8) in bytes. For MDHA, writing 0 to this field will be interpreted as an ICV size of 8 bytes. For AESA, writing 0, 1, 2 or 3 to this field will be interpreted as an ICV size of 16 bytes. |

## 13.202 CCB a Class 2 Context Register Word b (C0C2CTXR0 - C2C2CTXR17)

### 13.202.1 Offset

For a = 0 to 2; b = 0 to 17:

| Register | Offset | Description |
|---|---|---|
| CaC2CTXRb | 8_0500h + (a × 1_0000h) + (b × 4h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.202.2 Function

The Class 2 Context Register holds the context for the Class 2 CHAs. This register is 576 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 2 Context Register is accessible only as full-word reads or writes to eighteen 32-bit registers. The MSB is located at offset 500h with respect to the register page. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors.

The bit assignments for this register are dependent on the algorithm. See the appropriate section for the Context Register format used by that algorithm.

- CRC: Section CRCA Context Register
- Kasumi f9: The Kasumi f9 authentication algorithm is implemented in the same CHA as the Kasumi f8 encryption algorithm. This is considered a Class 1 CHA. See Section KFHA use of the Context Register
- MD5: Section MDHA use of the Context Register

- SHA-*: Section MDHA use of the Context Register
- SNOW 3G f9: Section SNOW 3G f9 use of the Context Register
- ZUC Authentication: Section ZUCA use of the Context Register

### 13.202.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | C2CTXR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | C2CTXR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.202.4 Fields

| Field | Function |
|---|---|
| 31-0<br>C2CTXR | Class 2 Context. |

## 13.203 CCB a Class 2 Key Register Word b (C0C2KEYR0 - C2C2KEYR31)

### 13.203.1 Offset

For a = 0 to 2; b = 0 to 31:

| Register | Offset | Description |
|---|---|---|
| CaC2KEYRb | 8_0600h + (a × 1_0000h) + (b × 4h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.203.2 Function

The Class 2 Key Register holds the key for the Class 2 CHAs. For non-split keys, the key is left-aligned in the C2KEYR. Split keys are the IPAD and OPAD for certain MDHA operations. Note that the size of a split key is the sum of the sizes of the IPAD and the OPAD. The IPAD is left-aligned within the C2KEYR, and the OPAD is left-aligned starting at the mid-point of the C2KEYR. This register is 1024 bits in length. Individual byte writes are supported when this register is accessed via descriptor commands, but via the IP bus the Class 2 Key Register is accessible only as full-word reads or writes to thirty-two 32-bit registers. The MSB is located at offset 600h with respect to the start of the register page. This register is automatically written by KEY commands. The recommended practice is to write the Class 2 Key Register prior to writing any of the other Class 2 registers. This register is cleared when checking the signature over, or re-signing, Trusted Descriptors.

### 13.203.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | C2KEY | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | C2KEY | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.203.4 Fields

| Field | Function |
|---|---|
| 31-0<br>C2KEY | Class 2 Key. |

## 13.204 CCB a FIFO Status (C0FIFOSTA - C2FIFOSTA)

### 13.204.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaFIFOSTA | 8_07C0h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.204.2 Function

The CCB FIFO Status Register is used during debug to facilitate reading the CCB FIFOs. Software must keep track of the data written to the Input Data FIFO (CCB a Input Data FIFO (C0IFIFO - C2IFIFO)), but the data within the Output Data FIFO (CCB a Output Data FIFO (C0OFIFO - C2OFIFO)) can be read out. Both the Class 1 Alignment Block and the Class 2 Alignment Block (see Alignment blocks) draw data from the Input Data FIFO, and both the DMA and the DECO Alignment Block draw data from the Output Data FIFO. Reading the CaFIFOSTA register returns the current heads of the Alignment Block and DMA queues within these two FIFOs. Note that the values in this register will change as descriptors are executed, so the register should be read when the DECO is not actively executing a descriptor.

### 13.204.3 Diagram

<table>
<tr><td>Bits</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td>R</td><td colspan="8">C1IQHEAD</td><td colspan="8">C2IQHEAD</td></tr>
<tr><td>W</td><td colspan="8"></td><td colspan="8"></td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

<table>
<tr><td>Bits</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>R</td><td colspan="8">DMAOQHEAD</td><td colspan="8">DECOOQHEAD</td></tr>
<tr><td>W</td><td colspan="8"></td><td colspan="8"></td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

### 13.204.4 Fields

| Field | Function |
|---|---|
| 31-24<br>C1IQHEAD | This is the current head of the Class 1 Alignment Block queue located within the Input Data FIFO. The value in this field points to the next data that will be pulled from the Input Data FIFO by the Class 1 Alignment Block. |
| 23-16<br>C2IQHEAD | This is the current head of the Class 2 Alignment Block queue located within the Input Data FIFO. The value in this field points to the next data that will be pulled from the Input Data FIFO by the Class 2 Alignment Block. |
| 15-8<br>DMAOQHEAD | This is the current head of the DMA queue located within the Output Data FIFO. The value in this field points to the next data that will be pulled from the Output Data FIFO by the DMA controller. |
| 7-0<br>DECOOQHEAD | This is the current head of the DECO Alignment Block queue located within the Output Data FIFO. The value in this field points to the next data that will be pulled from the Output Data FIFO by the DECO Alignment Block. This is used during "out snooping" operations, i.e. when data is passed first through a Class 1 CHA and the results pushed into the OFIFO, and from there the results are sent through a Class 2 CHA. |

## 13.205 CCB a iNformation FIFO When STYPE Is Not 10 (C0NFIFO - C2NFIFO)

### 13.205.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaNFIFO | 8_07D0h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.205.2 Function

The iNformation FIFO (Input Information FIFO) is used to control the movement of data from any of four sources to any of the three alignment blocks (see Alignment blocks). The four sources are the Input Data FIFO, the Output Data FIFO, the CCB Padding Block and the Auxiliary Data FIFO. Note that the only way to get data out of any of these sources other than via the Output Data FIFO is to use an iNformation FIFO entry.

The depth of the iNformation FIFO is four entries. During normal operation, SEC will not cause the iNformation FIFO to overflow. Care must be taken to avoid overflowing the iNformation FIFO when writing to it directly as this can cause SEC to hang. This register can be automatically written by the FIFO LOAD and MOVE commands. (If data is written to the Input Data FIFO with the LOAD Command, or some other way that does not automatically generate an info FIFO entry, the user is responsible for writing to the iNformation FIFO. See LOAD Command destination codes 78h and 7Ah in LOAD commands.)

A single address is used to write to the iNformation FIFO for any particular DECO. The format of non-padding iNformation FIFO entries (STYPE ≠ 10) is shown below. The format of padding iNformation FIFO entries (STYPE = 10) is shown in CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C2NFIFO_2).

## 13.205.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | | | | | | | | | | | | | | |
| W | DEST | | LC2 | LC1 | FC2 | FC1 | STYPE | | DTYPE | | | | BND | PTYPE | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | Reserved | | | | | | | | | | | | | |
| W | OC | AST | | | DL | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.205.4 Fields

| Field | Function |
|---|---|
| 31-30<br>DEST | Destination. This specifies if the current entry defines data for the Class 1 CHA and/or Class 2 CHA. It can also be used to remove data from the FIFOs that are not needed.<br>00b - DECO Alignment Block. If DTYPE is Eh, data sent to the DECO Alignment Block is dropped. This is used to skip over input data. An error is generated if a DTYPE other than Eh (drop) or Fh (message) is used with the DECO Alignment Block destination.<br>01b - Class 1.<br>10b - Class 2.<br>11b - Both Class 1 and Class 2. |
| 29<br>LC2 | Last Class 2. This bit should be set when the data defined in the current iNformation FIFO entry is the last data going to the CHA or the last data prior to receiving ICV data going to the Class 2 CHA, as well as following the ICV data. When LC2 =1 the alignment block will be emptied as well. |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>28<br>LC1</td><td>Last Class 1.This bit should be set when the data defined in the current iNformation FIFO entry is the last data for the Class 1 CHA. When LC1 =1 a flush will be done and the alignment block will be emptied as well.</td></tr>
<tr><td>27<br>FC2</td><td>Flush Class 2. Same as LC2 except that data size ready for Class 2 is not asserted.<br>This bit can be set only via a LOAD Command and is only to be used when a MOVE from the Class 2 Alignment Block is to be done and the MOVE Command was executed when automatic information FIFO entries were disabled. In such cases, setting the LC2 bit could result in unpredictable behavior and the FC2 bit should be used.</td></tr>
<tr><td>26<br>FC1</td><td>Flush Class 1. Flush the remainder of the data out of the Class 1 alignment block.</td></tr>
<tr><td>25-24<br>STYPE</td><td>Source Type. This field defines the source of the data for the Alignment Block(s). (This is the register format description when STYPE ≠ 10. The register uses a different format when STYPE = 10. See CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C2NFIFO_2).) For STYPE ≠ 10, there are two interpretations of the STYPE field, depending on the setting of the AST bit:
<table>
<tr><th>AST=0</th><th>AST=1</th></tr>
<tr><td>STYPE = 00 : Input Data FIFO</td><td>STYPE = 00 : Auxiliary Data FIFO [-1]</td></tr>
<tr><td>STYPE = 01 : Output Data FIFO</td><td></td></tr>
<tr><td colspan="2">STYPE = 10 : Padding Block. The register format is different for this STYPE. See CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C2NFIFO_2).</td></tr>
<tr><td>STYPE = 11 : Out snooping [-1]</td><td>STYPE = 11 : Outsnooping from Auxiliary Data FIFO [-1]</td></tr>
</table>
<ol>
<li>The Auxiliary Data FIFO can be used to supply auxiliary data to the Alignment Blocks, should this be required for particular algorithms or protocols. Data is written into the Auxiliary Data FIFO using either a LOAD IMM command with DST=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO should be created in the NFIFO prior to executing a LOAD IMM or a MOVE (with WC=1) that writes to the Auxiliary Data FIFO, else DECO may hang.</li>
<li>When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.</li>
<li>This case is similar to the case of STYPE=11 and AST=0. The difference is that the Class 1 CHA gets its data from the Auxiliary Data FIFO instead of from the Input Data FIFO. The Class 2 Alignment Block still receives its data from the output FIFO.</li>
</ol></td></tr>
<tr><td>23-20<br>DTYPE</td><td>Data Type. This field defines the type of data that is going through the Input Data FIFO. This is used by the CHA to determine what type of processing needs to be done on the data. As shown below, the DTYPE is interpreted differently depending on the CHA that is consuming the data.
<table>
<tr><td>DTYPE</td><td>Data type for PKHA</td><td>Data type for other CHAs</td></tr>
<tr><td>0h</td><td>PKHA A0</td><td>Reserved</td></tr>
<tr><td>1h</td><td>PKHA A1</td><td>AAD (AESA AES GCM)</td></tr>
<tr><td>2h</td><td>PKHA A2</td><td>IV for (AESA AES GCM)</td></tr>
<tr><td>3h</td><td>PKHA A3</td><td>SAD (AESA)</td></tr>
<tr><td>4h</td><td>PKHA B0</td><td>Reserved</td></tr>
<tr><td>5h</td><td>PKHA B1</td><td>Reserved</td></tr>
<tr><td>6h</td><td>PKHA B2</td><td>Reserved</td></tr>
<tr><td>7h</td><td>PKHA B3</td><td>Reserved</td></tr>
<tr><td>8h</td><td>PKHA N</td><td>Reserved</td></tr>
</table></td></tr>
</table>

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td>
<table>
<tr><td>9h</td><td>PKHA E</td><td>Reserved</td></tr>
<tr><td>Ah</td><td>Reserved</td><td>ICV</td></tr>
<tr><td>Ch</td><td>PKHA A</td><td>Reserved</td></tr>
<tr><td>Dh</td><td>PKHA B</td><td>Reserved</td></tr>
<tr><td>Eh</td><td>Reserved</td><td>DECO Ignore (i.e. Skip)</td></tr>
<tr><td>Fh</td><td>Reserved</td><td>Message Data</td></tr>
<tr><td colspan="3">Other values are reserved.</td></tr>
</table>
1. The Auxiliary Data FIFO can be used to supply auxiliary data to the Alignment Blocks, should this be required for particular algorithms or protocols. Data is written into the Auxiliary Data FIFO using either a LOAD IMM command with DST=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO should be created in the NFIFO prior to executing a LOAD IMM or a MOVE (with WC=1) that writes to the Auxiliary Data FIFO, else DECO may hang.<br>
2. When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.<br>
3. This case is similar to the case of STYPE=11 and AST=0. The difference is that the Class 1 CHA gets its data from the Auxiliary Data FIFO instead of from the Input Data FIFO. The Class 2 Alignment Block still receives its data from the output FIFO.
</td></tr>
<tr><td>19<br>BND</td><td>Boundary padding. Boundary padding is selected if this bit is set. The boundary is always 16 bytes when STYPE ≠ 10 (Padding Block).</td></tr>
<tr><td>18-16<br>PTYPE</td><td>Pad Type. This field is ignored if STYPE ≠ 10 (Padding Block).<br>000b - All Zero.<br>001b - Random with nonzero bytes.<br>010b - Incremented (Starting with 01h).<br>011b - Random.<br>100b - All Zero with last byte containing the number of 0 bytes.<br>101b - Random with nonzero bytes with last byte 0.<br>110b - N bytes of padding all containing the number N -1.<br>111b - Random with nonzero bytes with last byte N.</td></tr>
<tr><td>15<br>OC</td><td>OFIFO Continuation - This bit causes the final word to not be popped from the Output Data FIFO.</td></tr>
<tr><td>14<br>AST</td><td>Additional Source Types. This bit selects between two meanings of the STYPE field. See the description of the STYPE field.</td></tr>
<tr><td>13-12<br>—</td><td>Reserved</td></tr>
<tr><td>11-0<br>DL</td><td>Data Length. The number of bytes that will be passed to a CHA. A maximum of 12 bits is supported. This means for larger chunks of data multiple entries in the iNformation FIFO will be required.</td></tr>
</table>

1. The Auxiliary Data FIFO can be used to supply auxiliary data to the Alignment Blocks, should this be required for particular algorithms or protocols. Data is written into the Auxiliary Data FIFO using either a LOAD IMM command with DST=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO should be created in the NFIFO prior to executing a LOAD IMM or a MOVE (with WC=1) that writes to the Auxiliary Data FIFO, else DECO may hang.
2. When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.
3. This case is similar to the case of STYPE=11 and AST=0. The difference is that the Class 1 CHA gets its data from the Auxiliary Data FIFO instead of from the Input Data FIFO. The Class 2 Alignment Block still receives its data from the output FIFO.

# 13.206 CCB a iNformation FIFO When STYPE Is 10 (C0NFIFO_2 - C2NFIFO_2)

## 13.206.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaNFIFO_2 | 8_07D0h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.206.2 Function

The format of padding iNformation FIFO entries (STYPE = 10) is shown below. The format of non-padding iNformation FIFO entries (STYPE <> 10) is shown in CCB a iNformation FIFO When STYPE Is Not 10 (C0NFIFO - C2NFIFO).

## 13.206.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | | | | | | | | | | | | | | |
| W | DEST | | LC2 | LC1 | FC2 | FC1 | STYPE | | DTYPE | | | | BND | PTYPE | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | Reserved | | | | | Reserved | | | | | | | | | |
| W | PR | | | | BM | PS | | | | PL | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.206.4 Fields

| Field | Function |
|---|---|
| 31-30<br>DEST | Destination. This specifies if the current entry defines data for the Class 1 CHA and/or Class 2 CHA. It can also be used to remove data from the FIFOs that are not needed. |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td>00b - DECO Alignment Block. If DTYPE is Eh, data sent to the DECO Alignment Block is dropped. This is used to skip over input data. An error is generated if a DTYPE other than Eh (drop) or Fh (message) is used with the DECO Alignment Block destination.<br>01b - Class 1.<br>10b - Class 2.<br>11b - Both Class 1 and Class 2.</td></tr>
<tr><td>29<br>LC2</td><td>Last Class 2. This bit should be set when the data defined in the current iNformation FIFO entry is the last data going to the CHA or the last data prior to receiving ICV data going to the Class 2 CHA, as well as following the ICV data. When LC2 =1 the alignment block will be emptied as well.</td></tr>
<tr><td>28<br>LC1</td><td>Last Class 1. This bit should be set when the data defined in the current iNformation FIFO entry is the last data for the Class 1 CHA. When LC1 =1 a flush will be done and the alignment block will be emptied as well.</td></tr>
<tr><td>27<br>FC2</td><td>Flush Class 2. Same as LC2 except that data size ready for Class 2 is not asserted.<br>This bit can only be set via a LOAD Command and is only to be used when a MOVE from the Class 2 Alignment Block is to be done and the MOVE Command was executed when automatic information FIFO entries were disabled. In such cases, setting the LC2 bit could result in unpredictable behavior and the FC2 bit should be used.</td></tr>
<tr><td>26<br>FC1</td><td>Flush Class 1. Flush the remainder of the data out of the Class 1 alignment block.</td></tr>
<tr><td>25-24<br>STYPE</td><td>Source Type. This field defines the source of the data for the Alignment Block(s). This is the register format description when STYPE = 10 (Padding Block). For STYPE ≠ 10, see CCB a iNformation FIFO When STYPE Is Not 10 (C0NFIFO - C2NFIFO).<br>* When Out snooping is selected, the Class 1 Alignment Block receives data from the Input Data FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.<br>** The Auxiliary Data FIFO can be used to supply auxiliary data to the Alignment Blocks, should this be required for particular algorithms or protocols. Data is written into the Auxiliary Data FIFO using either a LOAD IMM command with DST=78h or a MOVE command with DST=Fh. Note that the entry to consume the data from the Auxiliary Data FIFO must be created in the NFIFO prior to executing a LOAD IMM or a MOVE (with WC=1) that writes to the Auxiliary Data FIFO, else DECO will hang.</td></tr>
<tr><td>23-20<br>DTYPE</td><td>Data Type. This field defines the type of data that is going through the Input Data FIFO. This is used by the CHA to determine what type of processing needs to be done on the data. As shown below, the DTYPE is interpreted differently depending on the CHA that is consuming the data.
<table>
<tr><td>DTYPE</td><td>Data type for PKHA</td><td>Data type for other CHAs</td></tr>
<tr><td>0h</td><td>PKHA A0</td><td>Reserved</td></tr>
<tr><td>1h</td><td>PKHA A1</td><td>AAD (AESA AES GCM)</td></tr>
<tr><td>2h</td><td>PKHA A2</td><td>IV for (AESA AES GCM)</td></tr>
<tr><td>3h</td><td>PKHA A3</td><td>SAD (AESA)</td></tr>
<tr><td>4h</td><td>PKHA B0</td><td>Reserved</td></tr>
<tr><td>5h</td><td>PKHA B1</td><td>Reserved</td></tr>
<tr><td>6h</td><td>PKHA B2</td><td>Reserved</td></tr>
<tr><td>7h</td><td>PKHA B3</td><td>Reserved</td></tr>
<tr><td>8h</td><td>PKHA N</td><td>Reserved</td></tr>
<tr><td>9h</td><td>PKHA E</td><td>Reserved</td></tr>
<tr><td>Ah</td><td>Reserved</td><td>ICV</td></tr>
</table></td></tr>
</table>

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td></td><td><table>
<tr><td>Ch</td><td>PKHA A</td><td>Reserved</td></tr>
<tr><td>Dh</td><td>PKHA B</td><td>Reserved</td></tr>
<tr><td>Eh</td><td>Reserved</td><td>DECO Ignore (i.e. Skip)</td></tr>
<tr><td>Fh</td><td>Reserved</td><td>Message Data</td></tr>
<tr><td colspan="3">Other values are reserved.</td></tr>
</table></td></tr>
<tr><td>19<br>BND</td><td>Boundary padding. Boundary padding is selected if this bit is set. The boundary is always 16 bytes when STYPE ≠ Padding Block.</td></tr>
<tr><td>18-16<br>PTYPE</td><td>Pad Type. This field defines the type of padding that should be performed when the STYPE = Padding Block. This field is ignored if BND = 0 or STYPE ≠ Padding Block.<br>000b - All Zero.<br>001b - Random with nonzero bytes.<br>010b - Incremented (Starting with 01h).<br>011b - Random.<br>100b - All Zero with last byte containing the number of 0 bytes.<br>101b - Random with nonzero bytes with last byte 0.<br>110b - N bytes of padding all containing the number N -1.<br>111b - Random with nonzero bytes with last byte N.</td></tr>
<tr><td>15<br>PR</td><td>Prediction Resistance - If PTYPE specifies random data, setting PR=1 causes the RNG to supply random data with prediction resistance (i.e. reseeds the PRNG from the TRNG).</td></tr>
<tr><td>14-12<br>—</td><td>Reserved. Must be 0.</td></tr>
<tr><td>11<br>BM</td><td>Boundary Minus 1. When this bit is set with boundary padding, then boundary padding to a 4, 8 or 16-byte boundary minus 1 byte will be executed. For example, if a 16-byte boundary is selected with BM=1, padding will be done such that only 15 of the 16 bytes are used, leaving the 16th byte available for the user to fill.</td></tr>
<tr><td>10<br>PS</td><td>Pad Snoop. When this bit is set then the Class 2 CHA will snoop the padding data from the Output Data FIFO rather than getting it from the padding block. When snooping, the Class 1 Alignment Block receives data from the Input FIFO and the Class 2 Alignment Block receives data from the Output Data FIFO.</td></tr>
<tr><td>9-7<br>—</td><td>Reserved. Must be 0.</td></tr>
<tr><td>6-0<br>PL</td><td>Pad Length. The number of bytes needed to pad the current data. If boundary padding is selected then this should be set to 4, 8 or 16 bytes. PL includes the length byte or zero byte for all zero last N, random last N and random last 0 padding types.</td></tr>
</table>

# 13.207 CCB a Input Data FIFO (C0IFIFO - C2IFIFO)

## 13.207.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaIFIFO | 8_07E0h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.207.2 Function

There is one copy of this FIFO per DECO. Data to be processed by the various CHAs is first pushed into the Input Data FIFO for the appropriate DECO. Note that although the Input Data FIFO is 64-bits wide, a single 32-bit wide location is used to write data to the IFIFO. All data written to this location via the IP bus should be in big endian format. Like the other DECO/CCB registers, the Input Data FIFO supports byte enables, allowing one to four bytes to be written to the IFIFO from the IP bus, or one to eight bytes via a descriptor. Although data is normally pushed in multiples of 8 bits, there is a special mechanism that allows a 4-bit value to be pushed into the Input Data FIFO (see "Input Data FIFO Nibble Shift Register", value 76h, in LOAD commands). The IFIFO is sixteen entries deep, and each entry is eight bytes. During normal operation SEC will never overflow the Input Data FIFO. Care must be used to not overflow the Input Data FIFO when writing to it directly as results will be unpredictable. FIFO LOAD, FIFO STORE, LOAD, KEY, and MOVE commands can all automatically write to this register.

## 13.207.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | | | | | | | | | | | | | | |
| W | IFIFO | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | | | | | | | | | | | | | | | | |
| W | IFIFO | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.207.4 Fields

| Field | Function |
|---|---|
| 31-0<br>IFIFO | Input Data FIFO. |

## 13.208 CCB a Output Data FIFO (C0OFIFO - C2OFIFO)

### 13.208.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| CaOFIFO | 8_07F0h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.208.2 Function

There is one copy of this FIFO per DECO. Data that is output from the various CHAs is pushed into the Output Data FIFO for the appropriate DECO. The OFIFO is sixteen entries deep, and each entry is eight bytes. During normal operation, SEC will never overflow the Output Data FIFO. KEY, MOVE, MATH, and FIFO STORE commands will all read from the Output Data FIFO. Normally data is pushed in multiples of 8 bits, but there is a special mechanism that allows a 4-bit value to be pushed into the Output Data FIFO (see Output Data FIFO Nibble Shift Register, value 77h, in LOAD commands).

There are several commands that can result in data being pushed into the Output Data FIFO:

- The OPERATION Command can cause a Class 1 CHA to put data into the Output Data FIFO.
- The KEY Command uses the Output Data FIFO when it decrypts keys. Since the Output Data FIFO must be empty and all transactions must have completed before the KEY Command will start, there will be no collision between a CHA push and an ODFNSR push to the Output Data FIFO.
- The (SEQ) FIFO STORE Command, when encrypting keys, also pushes data into the Output Data FIFO.
- A LOAD IMMEDIATE can push data directly into the Output Data FIFO. DECO will automatically stall a LOAD IMMEDIATE if necessary to prevent a collision with a push from the ODFNSR.

- A LOAD IMMEDIATE to the CHA Control Register can be used to "unload" or various PKHA registers into the Output Data FIFO.
- The (SEQ) FIFO STORE Command, when generating random data, also pushes data into the Output Data FIFO.

Automatic DECO stalling is accomplished as follows. Once the ODFNSR is written, the stalls described above will continue until the ODFNSR is cleared. That means that the Class 1 CHA has to assert its done signal and the ODFNSR has to have pushed its final value into the Output Data FIFO. WARNING: If the DECO is stalling while waiting for the ODFNSR to empty, and there is no already executed command (such as a FIFO STORE or MOVE) that will drain the Output Data FIFO sufficiently to allow the ODFNSR to empty, the DECO will hang.

Internally the Output Data FIFO is 64-bits wide, but since the IP bus is 32-bits wide, the Output Data FIFO is read via the IP bus using 32-bit word reads. The most-significant half of the 64-bit word is read from address BASE+x7F0, and the least significant half is read from address BASE+x7F4. All data read from the OFIFO is big endian.

## 13.208.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | OFIFO | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | OFIFO | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | OFIFO | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | OFIFO | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.208.4 Fields

| Field | Function |
|---|---|
| 63-0<br>OFIFO | Output FIFO |

## 13.209 DECOa Job Queue Control Register, most-significant half (D0JQCR_MS - D2JQCR_MS)

### 13.209.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaJQCR_MS | 8_0800h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.209.2 Function

This register tells the Descriptor Controller about the current Descriptor. There is one copy of this register per DECO. During normal operation, this register is written by the job queue controller. When a DECO is under the direct control of software (see Register-based service interface), this register can be read or written from the IP Register bus. Writing to the most-significant half of this register causes the Descriptor Controller to start processing. Note that at least the first burst of the Descriptor (including the Job Descriptor Header and the JOB HEADER extension word, if any) must be written to the Descriptor buffer before the Job Queue Control Register is written.

## 13.209.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | STEP | SING | WHL | FOUR | ILE | SHR_FROM | | | Reserved | | | | DWS | Reserved | | SOB |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AMTD | JDIS | Reserved | | | SRC | | | Reserved | | | | ID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.209.4 Fields

| Field | Function |
|---|---|
| 31<br>STEP | Step. When in Single Step Mode, DECO should execute the next command in the descriptor. Note that protocols are a single step. Only used by the processor that has control of DECO. |
| 30<br>SING | Single Step Mode. This tells DECO to execute this descriptor, including jumps to non-local destinations, in single step mode. Only used by the processor that has control of DECO. |
| 29<br>WHL | Whole Descriptor. This bit indicates that the whole Descriptor was given to DECO by the job queue controller. This bit is set for certain Job Descriptors that are internally generated by SEC. |
| 28<br>FOUR | Four Words. job queue controller is passing at least 4 words of the Descriptor to DECO. |
| 27<br>ILE | Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors.<br>0b - No byte-swapping is performed for data transferred to or from the Descriptor Buffer.<br>1b - Byte-swapping is performed when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: Input Data FIFO, Output Data FIFO, and Class 1 Context, Class 2 Context, Class1 Key and Class 2 Key registers. |
| 26-24<br>SHR_FROM | Share From. This is the DECO block from which this DECO block will get the Shared Descriptor. This field is only used if the job queue controller wants this DECO to use a Shared Descriptor that is already in a DECO. This field is ignored when running descriptors via the IP bus (i.e. under the direct control of software). |
| 23-20<br>— | Reserved |
| 19<br>DWS | Double word swap. Causes/allows dword swapping of addresses, and MOVE and MATH immediate values. |
| 18-17 | Reserved |

*Table continues on the next page...*

| Field | Function |
|---|---|
| — | |
| 16<br>SOB | Shared Descriptor or input frame burst. If set, the whole shared descriptor and/or first bursts of QI input frame data were passed to DECO with the Job Descriptor. When descriptors are executed under direct software control, this bit simply indicates that the Shared Descriptor has been loaded. |
| 15<br>AMTD | Allow Make Trusted Descriptor. This field is read-only. If this bit is a 1, then a Job Descriptor with the MTD (Make Trusted Descriptor) bit set is allowed to execute. The bit will be 1 only if the Job Descriptor was run from a job ring with the AMTD bit set to 1 in the job ring's JRaICID Register. |
| 14<br>JDIS | Job Descriptor ICID Select. Determines whether the SEQ ICID or the Non-SEQ ICID is asserted when reading the Job Descriptor from memory.<br>0b - Non-SEQ ICID<br>1b - SEQ ICID |
| 13-11<br>— | Reserved |
| 10-8<br>SRC | Job Source. Source of the job. Determines which set of DMA configuration attributes (e.g. JRCFGR_JRa_MS) and endian configuration bits ) the DMA should use for bus transactions. It is illegal for the SRC field to have a value other than that of a job ring when running descriptors via the IP bus (i.e. under the direct control of software).<br>000b - Job Ring 0<br>001b - Job Ring 1<br>010b - Job Ring 2<br>011b - Job Ring 3<br>100b - RTIC<br>101b - Queue Manager Interface<br>110b - Reserved<br>111b - Reserved |
| 7-4<br>— | Reserved |
| 3-0<br>ID | Job ID. Unique tag given to each job by its source. Used to tell the source that the job has completed. |

## 13.210 DECOa Job Queue Control Register, least-significant half (D0JQCR_LS - D2JQCR_LS)

### 13.210.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaJQCR_LS | 8_0804h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.210.2 Function

This register tells the Descriptor Controller about the current Descriptor. There is one copy of this register per DECO. During normal operation, this register is written by the job queue controller. When a DECO is under the direct control of software (see Register-based service interface), this register can be read or written from the IP Register bus. Writing to the most-significant half of this register causes the Descriptor Controller to start processing. Note that at least the first burst of the Descriptor (including the Job Descriptor Header and the JOB HEADER extension word, if any) must be written to the Descriptor buffer before the Job Queue Control Register is written.

### 13.210.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | CMD | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | CMD | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.210.4 Fields

| Field | Function |
|---|---|
| 31-0<br>CMD | Command. In single-step mode, reading CMD returns the first word of the command that this DECO will execute next. This value is also readable via the STORE Command, but the value read is unpredictable. |

## 13.211 DECOa Descriptor Address Register (D0DAR - D2DAR)

### 13.211.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaDAR | 8_0808h + (a × 1_0000h) | For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.211.2 Function

This DECO register holds the address of the currently executing Job Descriptor. When using DECO in single-step mode (see Register-based service interface), this register must be written prior to the Job Queue Control Register.

### 13.211.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | DPTR | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | DPTR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | DPTR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.211.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved |
| 39-0<br>DPTR | Descriptor Pointer. Memory address of the Descriptor. Needed for write-back purposes. |

## 13.212 DECOa Operation Status Register, most-significant half (D0OPSTA_MS - D2OPSTA_MS)

### 13.212.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaOPSTA_MS | 8_0810h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.212.2 Function

The DECO Operation Status Register is used to show DECO status following descriptor processing. This includes error conditions (if any), index of the next command within the descriptor, and condition flags set by Public Key and Math Operations. Since the register is greater than 32 bits, the OPSTA register is accessed from the IP bus as two 32-bit registers.

## 13.212.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | STATUS_TYPE | | | | NLJ | Reserved | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | COMMAND_INDEX | | | | | | | STATUS | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.212.4 Fields

| Field | Function |
|---|---|
| 31-28<br>STATUS_TYPE | Status Type. The type of status that is being reported for the just-completed command, defined as follows:<br>0000b - no error<br>0001b - DMA error<br>0010b - CCB error<br>0011b - Jump Halt User Status<br>0100b - DECO error<br>0101b - Reserved<br>0110b - Reserved<br>0111b - Jump Halt Condition Code |
| 27<br>NLJ | Non-Local Jump. A jump was non-local. This includes non-local JUMP Commands and SEQ IN PTR RJD jumps and SEQ IN PTR INL jumps. |
| 26-15<br>— | Reserved. Always 0. |
| 14-8<br>COMMAND_INDEX | Command index: A pointer to a 32-bit word within the descriptor. If single stepping, this is the index of the next command to be executed. If not single stepping, this is the index of the command that is now executing. In the case of an error that is not a command problem, it is approximately the index of the command where the error occurred. If the error was due to a command problem, it is the index of the current command. A command problem is an error that is detectable by DECO as it executes the command (e.g. an illegal command type). Something that isn't a command problem is an error that occurs after the command has completed executing (e.g. illegal CHA modes, DMA errors, ICV check failures). |

*Table continues on the next page...*

<table>
<tr><th>Field</th><th>Function</th></tr>
<tr><td>7-0<br>STATUS</td><td>If ERRTYP indicates no error, this field contains PKHA/Math Status, as defined below. If there was an error, this field contains Error Status, defined as in the Job Ring Output Status Register DESC ERROR field (Job Ring Output Status Register for Job Ring a (JRSTAR_JR0 - JRSTAR_JR3)).
<table>
<tr><td>7<br>PIZ</td><td>Public Key Operation is Zero. For Finite Field operations the result of a Public Key operation is zero. For ECC operations, the result is Point at infinity.</td></tr>
<tr><td>6<br>GCD</td><td>GCD is One. The greatest common divisor of two numbers (i.e., the two numbers are relatively prime)</td></tr>
<tr><td>5<br>PRM</td><td>Public Key is Prime. The given number is probably prime (i.e., it passes the Miller-Rabin primality test)</td></tr>
<tr><td>4</td><td>Reserved</td></tr>
<tr><td>3<br>MN</td><td>Math N. The result is negative. Can only be set by add and subtract functions, 0 otherwise</td></tr>
<tr><td>2<br>MZ</td><td>Math Z. The result of a math operation is zero.</td></tr>
<tr><td>1<br>MC</td><td>Math C. The math operation resulted in a carry or borrow.</td></tr>
<tr><td>0<br>MNV</td><td>Math NV. Used for signed compares. This is a combination of the sign and overflow bits (i.e., Math N XOR Math C)</td></tr>
</table>
</td></tr>
</table>

# 13.213 DECOa Operation Status Register, least-significant half (D0OPSTA_LS - D2OPSTA_LS)

## 13.213.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaOPSTA_LS | 8_0814h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.213.2 Function

The DECO Operation Status Register is used to show DECO status following descriptor processing. This includes error conditions (if any), index of the next command within the descriptor, and condition flags set by Public Key and Math Operations. Since the register is greater than 32 bits, the OPSTA register is accessed from the IP bus as two 32-bit registers.

### 13.213.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | OUT_CT | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | OUT_CT | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.213.4 Fields

| Field | Function |
|---|---|
| 31-29<br>— | Reserved |
| 28-0<br>OUT_CT | Output Count. Number of bytes written to sequential out pointer. |

## 13.214 DECOa Checksum Register (D0CKSUMR - D2CKSUMR)

### 13.214.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaCKSUMR | 8_0818h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.214.2 Function

Checksum. This register shows the checksum value computed by DECO. This value is intended to assist in implementing the IPSEC protocol.

## 13.214.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | CKSUM | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.214.4 Fields

| Field | Function |
|---|---|
| 31-16<br>— | Reserved |
| 15-0<br>CKSUM | Checksum. All byte data written to the output frame via SEQ STORE or SEQ FIFO STORE is included in the checksum, with the following exceptions. If a type 31h SEQ FIFO STORE is executed, the previous checksum value (if any) due to non-type 31h SEQ FIFO STORE commands is discarded and the running checksum starts with the fresh type 31h or type 3Eh data. With one exception as noted below, all byte data stored with SEQ STORE or SEQ FIFO STORE 3Eh (meta data) is included in the running checksum until a SEQ FIFO STORE is executed that is not type 31h or 3Eh. After that no additional data is included in the checksum until another type 31h SEQ FIFO STORE is executed. At that point the running checksum is not cleared, and the running checksum includes all subsequent byte data written via a type 31h or type 3Eh SEQ FIFO STORE command. Commands that store the CKSUM register are an exception to the general rules above. When the CKSUM register is stored via a STORE or SEQ STORE command, the register value will not be included in the running checksum. If DECO was computing a checksum of type 31h or type 3Eh data when the CKSUM register was stored, following the store of the CKSUM register no further data will be included in the running checksum until another SEQ FIFO STORE of type 0x31 is executed. But if at the time that the CKSUM register was stored DECO was computing a checksum of data other than type 31h or type 3Eh data, all byte data stored via SEQ STOREs and SEQ |

| Field | Function |
|---|---|
| | FIFO STOREs will be included in the running checksum until a type 0x31 SEQ FIFO STORE is executed, at which point the running checksum will be cleared and a checksum of type 31h or type 3Eh data will begin. |

# 13.215 DECOa SDID / Trusted ICID Status Register (D0SD IDSR - D2SDIDSR)

## 13.215.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaSDIDSR | 8_0820h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.215.2 Function

This register shows the ICIDs available to the DECO for processing the current Descriptor. A descriptor can select among the available ICIDs via a LOAD IMM to the DECO Control Register. (see Value 06h in LOAD commands) This register is written by the job queue controller.

## 13.215.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | TZ | Reserved | | | SDID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | DTICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.215.4 Fields

| Field | Function |
|---|---|
| 31<br>TZ | TrustZone. This is the TrustZone value (where SecureWorld = 1) used by the DECO when processing the current Descriptor. |
| 30-28<br>— | Reserved |
| 27-16<br>SDID | Security Domain Identifier. This is the SDID value used by the DECO when processing the current Descriptor. |
| 15-12<br>— | Reserved |
| 11-0<br>DTICID | DECO Trusted ICID. Any type of descriptor can write to this field, but only Trusted Descriptors can choose to assert this ICID value during DMA transactions. Although this field can be read and written via the IP bus interface, this is of no consequence since Trusted Descriptors cannot be executed via the IP bus interface.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

## 13.216 DECOa ICID Status Register (D0ISR - D2ISR)

### 13.216.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaISR | 8_0820h + (a × 1_0000h) | Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.216.2 Function

This register shows the ICIDs available to the DECO for processing the current Descriptor. A descriptor can select among the available ICIDs via a LOAD IMM to the DECO Control Register. (see Value 06h in LOAD commands). This register is written by the job queue controller.

### 13.216.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | DNSICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | DSICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.216.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved |
| 27-16<br>DNSICID | DECO Non-SEQ ICID. This is the ICID that will be asserted during a DMA transaction that is related to descriptor operations other than Input or Output SEQ sequences. This field can be read but cannot be written via the IP bus interface.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |
| 15-12<br>— | Reserved |
| 11-0<br>DSICID | DECO SEQ ICID. This is the ICID that will be asserted during a DMA transaction that is related to an Input or Output SEQ sequence. This field can be read but cannot be written via the IP bus interface.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

## 13.217 DECOa Math Register b_MS (D0MTH0_MS - D2MTH7_MS)

### 13.217.1 Offset

For a = 0 to 2; b = 0 to 7:

| Register | Offset |
|---|---|
| DaMTHb_MS | 8_0840h + (a × 1_0000h) + (b × 8h) |

### 13.217.2 Function

The Math Registers are used by the DECO to perform Math operations that were requested via the MATH Command. The Math Registers consist of 8 64-bit registers per DECO. Data is moved into these registers via LOAD, MATH and MOVE commands.

### 13.217.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MATH_MS | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MATH_MS | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.217.4 Fields

| Field | Function |
|---|---|
| 31-0<br>MATH_MS | MATH register, most-significant 32 bits. |

## 13.218 DECOa Math Register b_LS (D0MTH0_LS - D2MTH7_LS)

### 13.218.1 Offset

For a = 0 to 2; b = 0 to 7:

| Register | Offset |
|---|---|
| DaMTHb_LS | 8_0844h + (a × 1_0000h) + (b × 8h) |

### 13.218.2 Function

The Math Registers are used by the DECO to perform Math operations that were requested via the MATH Command. The Math Registers consist of 8 64-bit registers per DECO. Data is moved into these registers via LOAD, MATH and MOVE commands.

### 13.218.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MATH_LS | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | MATH_LS | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.218.4 Fields

| Field | Function |
|---|---|
| 31-0<br>MATH_LS | MATH register, least-significant 32 bits. |

## 13.219 DECOa Gather Table Register b Word 0 (D0GTR0_0 - D2GTR3_0)

### 13.219.1 Offset

For a = 0 to 2; b = 0 to 3:

| Register | Offset |
|---|---|
| DaGTRb_0 | 8_0880h + (a × 1_0000h) + (b × 10h) |

### 13.219.2 Function

The Gather Table Registers and Scatter Table Registers hold the current Gather Table and Scatter Table entries that are being used by the DECO. (See Scatter/gather tables (SGTs).) SEC will fetch a burst's worth of entries at a time, so these entries are held in four Gather Table registers and four Scatter Table registers per DECO. A burst is 32 or 64 bytes, as indicated by the BURST field in the Master Configuration Register. Each register is 128 bits in length, so the data in each register is accessible over the 32-bit register bus as four 32-bit words.

### 13.219.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | ADDRESS_POINTER | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.219.4 Fields

| Field | Function |
|---|---|
| 31-8<br>— | Reserved. |
| 7-0<br>ADDRESS_POINTER | most-significant bits of memory address pointed to by table entry<br>This field holds the most-significant 8 bits of the memory address to which this table entry points. This will either be a memory buffer (if E=0) or a Scatter/Gather Table entry (if E=1). |

## 13.220 DECOa Gather Table Register b Word 1 (D0GTR0_1 - D2GTR3_1)

### 13.220.1 Offset

For a = 0 to 2; b = 0 to 3:

| Register | Offset |
|---|---|
| DaGTRb_1 | 8_0884h + (a × 1_0000h) + (b × 10h) |

### 13.220.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ADDRESS_POINTER | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ADDRESS_POINTER | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.220.3 Fields

| Field | Function |
|---|---|
| 31-0<br>ADDRESS_POINTER | This field holds the least-significant 32 bits of the memory address to which this table entry points. This will either be a memory buffer (if E=0) or a Scatter/Gather Table entry (if E=1). |

## 13.221 DECOa Gather Table Register b Word 2 (D0GTR0_2 - D2GTR3_2)

### 13.221.1 Offset

For a = 0 to 2; b = 0 to 3:

| Register | Offset |
|---|---|
| DaGTRb_2 | 8_0888h + (a × 1_0000h) + (b × 10h) |

### 13.221.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | E | F | Length | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Length | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.221.3 Fields

| Field | Function |
|---|---|
| 31<br>E | Extension bit. If set to a 1, then Address Pointer refers to a Scatter/Gather Table entry instead of a memory buffer. It is an error to set the E bit if the SGT entry is unused (i.e. Length, BPID and Address Pointer all 0s). |
| 30<br>F | Final Bit. If set, this is the last entry of this Scatter/Gather Table. |
| 29-0<br>Length | This field specifies how many bytes of data (for Gather Tables) or available space (for Scatter Tables) are located at the address pointed to by the Address Pointer. |

## 13.222 DECOa Gather Table Register b Word 3 (D0GTR0_3 - D2GTR3_3)

### 13.222.1 Offset

For a = 0 to 2; b = 0 to 3:

| Register | Offset |
|---|---|
| DaGTRb_3 | 8_088Ch + (a × 1_0000h) + (b × 10h) |

## 13.222.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | BPID | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | Offset | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.222.3 Fields

| Field | Function |
|---|---|
| 31-24<br>— | Reserved |
| 23-16<br>BPID | Buffer Pool ID. Indicates Buffer Pool owning the buffer referred to by the Address Pointer. |
| 15-13<br>— | Reserved |
| 12-0<br>Offset | Offset (measured in bytes) into memory where significant data is to be found. The use of an offset permits reuse of a memory buffer without recalculating the address. |

## 13.223 DECOa Scatter Table Register b Word 0 (D0STR0_0 - D2STR3_0)

### 13.223.1 Offset

For a = 0 to 2; b = 0 to 3:

| Register | Offset |
|---|---|
| DaSTRb_0 | 8_0900h + (a × 1_0000h) + (b × 10h) |

## 13.223.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | ADDRESS_POINTER | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.223.3 Fields

| Field | Function |
|---|---|
| 31-8<br>— | Reserved. |
| 7-0<br>ADDRESS_POINTER | most-significant bits of memory address pointed to by table entry<br>This field holds the most-significant 8 bits of the memory address to which this table entry points. This will either be a memory buffer (if E=0) or a Scatter/Gather Table entry (if E=1). |

# 13.224 DECOa Scatter Table Register b Word 1 (D0STR0_1 - D2STR3_1)

## 13.224.1 Offset

For a = 0 to 2; b = 0 to 3:

| Register | Offset |
|---|---|
| DaSTRb_1 | 8_0904h + (a × 1_0000h) + (b × 10h) |

### 13.224.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ADDRESS_POINTER | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | ADDRESS_POINTER | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.224.3 Fields

| Field | Function |
|---|---|
| 31-0<br>ADDRESS_POINTER | This field holds the least-significant 32 bits of the memory address to which this table entry points. This will either be a memory buffer (if E=0) or a Scatter/Gather Table entry (if E=1). |

## 13.225 DECOa Scatter Table Register b Word 2 (D0STR0_2 - D2STR3_2)

### 13.225.1 Offset

For a = 0 to 2; b = 0 to 3:

| Register | Offset |
|---|---|
| DaSTRb_2 | 8_0908h + (a × 1_0000h) + (b × 10h) |

### 13.225.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | E | F | Length | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Length | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.225.3 Fields

| Field | Function |
|---|---|
| 31<br>E | Extension bit. If set to a 1, then Address Pointer refers to a Scatter/Gather Table entry instead of a memory buffer. It is an error to set the E bit if the SGT entry is unused (i.e. Length, BPID and Address Pointer all 0s). |
| 30<br>F | Final Bit. If set, this is the last entry of this Scatter/Gather Table. |
| 29-0<br>Length | This field specifies how many bytes of data (for Gather Tables) or available space (for Scatter Tables) are located at the address pointed to by the Address Pointer. |

## 13.226 DECOa Scatter Table Register b Word 3 (D0STR0_3 - D2STR3_3)

### 13.226.1 Offset

For a = 0 to 2; b = 0 to 3:

| Register | Offset |
|---|---|
| DaSTRb_3 | 8_090Ch + (a × 1_0000h) + (b × 10h) |

### 13.226.2 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | BPID | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | Offset | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.226.3 Fields

| Field | Function |
|---|---|
| 31-24<br>— | Reserved |
| 23-16<br>BPID | Buffer Pool ID. Indicates Buffer Pool owning the buffer referred to by the Address Pointer. |
| 15-13<br>— | Reserved |
| 12-0<br>Offset | Offset (measured in bytes) into memory where significant data is to be found. The use of an offset permits reuse of a memory buffer without recalculating the address. |

## 13.227 DECOa Descriptor Buffer Word b (D0DESB0 - D2DESB63)

### 13.227.1 Offset

For a = 0 to 2; b = 0 to 63:

| Register | Offset |
|---|---|
| DaDESBb | 8_0A00h + (a × 1_0000h) + (b × 4h) |

## 13.227.2 Function

The Descriptor Buffer is used by the DECO to buffer a Descriptor that has been fetched from memory. The Descriptor Buffer consists of 64 32-bit registers in consecutive addresses, beginning at the addresses shown above. For performance reasons, DECO doesn't execute the commands directly from the Descriptor Buffer. Instead, DECO executes commands from a four-word pipeline. Since commands vary in length from one to four words, up to three words in addition to the current command may also be resident in the pipeline. (They won't be executed if the job terminates or the pipeline is flushed as described below.) As a result, operations that modify the Descriptor Buffer may not have an immediate effect on the next few commands that execute. To avoid anomalous behavior when overwriting the portion of the Descriptor Buffer containing the start of the currently executing command or the following two or three words, any commands in the pipeline that the programmer intends to execute should be completely contained within the pipeline.

There are several ways to flush the pipeline to ensure that recently loaded commands are executed rather than the pipeline-resident commands:

- Execute a JUMP command with a negative offset
- Use the JOB HEADER or SHARED HEADER commands to do an absolute jump.
- JUMP forward more than 3 words.

Note that the Descriptor Buffer is cleared between unrelated descriptors; that is, if two successive descriptors to execute in the same DECO do not share the same shared descriptor.

## 13.227.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Descriptor_Buffer | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Descriptor_Buffer | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.227.4 Fields

| Field | Function |
|---|---|
| 31-0<br>Descriptor_Buffer | Descriptor_Buffer |

## 13.228 DECOa Debug Job (D0DJR - D2DJR)

### 13.228.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaDJR | 8_0E00h + (a × 1_0000h) | For DECOa. |

### 13.228.2 Function

The DECOa Debug Job, DECOa Debug DECO, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see Register-based service interface). Note that the DECOa Debug Job has the same format as the most-significant half of the DECO Job Queue Control Register. Note that this register is read-only.

## 13.228.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | STEP | SING | WHL | FOUR | ILE | SHR_FROM | | | Reserved | | | | DWS | Reserved | | GSD |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | AMTD | JDIS | Reserved | | | SRC | | | Reserved | | | | ID | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.228.4 Fields

| Field | Function |
|---|---|
| 31<br>STEP | Step. When in Single Step Mode, DECO should execute the next command in the descriptor. Note that protocols are a single step. Only used by the processor that has control of DECO. |
| 30<br>SING | Single Step Mode. This tells DECO to execute this descriptor, including jumps to non-local destinations, in single step mode. Only used by the processor that has control of DECO. |
| 29<br>WHL | Whole Descriptor. This bit indicates that the whole Descriptor was given to DECO by the job queue controller (or by the processor that has control of DECO). This bit is set for certain Job Descriptors that are internally generated by SEC. |
| 28<br>FOUR | Four Words. The job queue controller (or the processor that has control of DECO) has passed at least 4 words of the Descriptor to DECO. |
| 27<br>ILE | Immediate Little Endian. This bit controls the byte-swapping of Immediate data embedded within descriptors.<br>0b - No byte-swapping is performed for data transferred to or from the Descriptor Buffer.<br>1b - Byte-swapping is performed when data is transferred between the Descriptor Buffer and any of the following byte-stream sources and destinations: Input Data FIFO, Output Data FIFO, and Class 1 Context, Class 2 Context, Class1 Key and Class 2 Key registers. |
| 26-24<br>SHR_FROM | Share From. This is the DECO block from which this DECO block will get the Shared Descriptor. This field is only used if the job queue controller wants this DECO to use a Shared Descriptor that is already in a DECO. This field is ignored when running descriptors via the IP bus (i.e. under the direct control of software). |
| 23-20<br>— | Reserved |
| 19<br>DWS | Double Word Swap. Double word swapping was set. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 18-17<br>— | Reserved |
| 16<br>GSD | Got Shared Descriptor. A Shared Descriptor was already available in a DECO so the DECO didn't need to fetch the Shared Descriptor from memory. |
| 15<br>AMTD | Allow Make Trusted Descriptor. If this bit is a 1, then a Job Descriptor whose HEADER command has TDES=11b (candidate trusted descriptor) is allowed to execute. The AMTD bit will be 1 only if the Job Descriptor was run from a job ring with the AMTD bit set to 1 in the job ring's JRaICID Register. |
| 14<br>JDIS | Job Descriptor ICID Select. Determines whether the SEQ ICID or the Non-SEQ ICID is asserted when reading the Job Descriptor from memory.<br>0b - Non-SEQ ICID<br>1b - SEQ ICID |
| 13-11<br>— | Reserved |
| 10-8<br>SRC | Job Source. Source of the job. Determines which set of DMA configuration attributes (e.g. JRCFGR_JRa_MS) and endian configuration bits ) the DMA should use for bus transactions. When running descriptors via the IP bus (i.e. under the direct control of software), the job queue controller automatically sets this field to indicate a job ring source.<br>000b - Job Ring 0<br>001b - Job Ring 1<br>010b - Job Ring 2<br>011b - Job Ring 3<br>100b - RTIC<br>101b - Queue Manager Interface<br>110b - Reserved<br>111b - Reserved |
| 7-4<br>— | Reserved |
| 3-0<br>ID | Job ID. Unique tag given to each job by its source (see SRC field). Used to tell the source that the job has completed. |

## 13.229 DECOa Debug DECO (D0DDR - D2DDR)

### 13.229.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaDDR | 8_0E04h + (a × 1_0000h) | For DECOa. |

## 13.229.2 Function

The DECOa Debug Job, DECOa Debug DECO, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see Register-based service interface). Note that this register is read-only.

## 13.229.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | VALID | SD | TRCT | | SEQLSEL | | NSEQLSEL | | DECO_STATE | | | | PDB_WB_ST | | PDB_STALL | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | PTCL_RUN | NLJ | CMD_INDEX | | | | | | CMD_STAGE | | | CSA | NC | BWB | BRB | CT |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.229.4 Fields

| Field | Function |
|---|---|
| 31<br>VALID | Valid. If VALID=1, there is currently a job descriptor running in this DECO. The descriptor has been loaded and has started executing and is still executing. |
| 30<br>SD | Shared Descriptor. The job descriptor that is running in this DECO has received a shared descriptor from another job descriptor. That is, some other job descriptor used this shared descriptor (in the same DECO or a different DECO), and this job descriptor is using the shared descriptor without having to load it from memory. In the case of SERIAL or WAIT sharing, then the keys were shared as well. If the SC bit was on, then the context was also shared. |

*Table continues on the next page...*

| Field | Function |
|---|---|
| 29-28<br>TRCT | DMA Transaction Count. This indicates how many outstanding external DMA transactions are pending. This is the total of reads and writes. DECO is limited to, at most, two such transactions. |
| 27-26<br>SEQLSEL | SEQ ICID Select. This indicates which type of ICID is being used for SEQ commands:<br>01b - SEQ ICID<br>10b - Non-SEQ ICID<br>11b - Trusted ICID |
| 25-24<br>NSEQLSEL | Non-SEQ ICID Select. This indicates which type of ICID is being used for Non-SEQ commands:<br>01b - SEQ ICID<br>10b - Non-SEQ ICID<br>11b - Trusted ICID |
| 23-20<br>DECO_STATE | DECO State. The current state of DECO's main state machine. |
| 19-18<br>PDB_WB_ST | PDB Writeback State. Lower two bits of the state machine that tracks the state of PDB writebacks. |
| 17-16<br>PDB_STALL | PDB Stall State. The state of the machine that tracks the stalling of PDB writebacks. Used in conjunction with PDB_WB_ST. Used only if there is more than one DECO. |
| 15<br>PTCL_RUN | Protocol running. PTCL_RUN=1 indicates that a protocol is running in this DECO. |
| 14<br>NLJ | Took Non-local JUMP. If NLJ=1 the original job descriptor running in this DECO has caused another job descriptor to be executed. This is true for JUMP NON-LOCAL, SEQ IN PTR INLINE, and SEQ IN PTR RJD. |
| 13-8<br>CMD_INDEX | Command Index. If this DECO is currently executing a command, CMD_INDEX points to that command within the descriptor buffer. |
| 7-5<br>CMD_STAGE | Command Stage. Each command executes in a number of steps, or stages. There are 8 possible stages. CMD_STAGE indicates which stage DECO has reached in the process of executing a command. |
| 4<br>CSA | Command Stage Aux. A refinement of the CMD_STAGE stages. Some stages may be split into two substages, and CSA will indicate which of those two substages DECO has reached. |
| 3<br>NC | No Command. This DECO is not currently executing a command. This can be because the descriptor isn't executing or DECO is doing a JUMP of some sort. |
| 2<br>BWB | Burster Write Busy. The WRITE machine in the Burster is busy. This means that the WRITE machine is scheduling DMA transactions or is waiting for the opportunity to do so. It remains busy until all the transactions required for a request have been scheduled. STORE, SEQ STORE, FIFO STORE, and SEQ FIFO STORE commands use the WRITE machine. The WRITE machine is also used to update the Shared Descriptor HEADER when propagating DNR and by the Trusted State Machine to store a computed signature. |
| 1<br>BRB | Burster Read Busy. The READ machine in the Burster is busy. This means that the READ machine is scheduling DMA transactions or is waiting for the opportunity to do so. It remains busy until all the transactions required for a request have been scheduled. LOAD, SEQ LOAD, FIFO LOAD, SEQ FIFO LOAD, and the KEY command all use the READ machine. The read to satisfy RIF in the Shared Descriptor HEADER also uses the READ machine. The SEQ FIFO STORE command can also use the READ machine when handling meta data. Jumping non-locally via any method will also use the READ machine. Commands that reference Scatter/Gather Tables will also cause the READ machine to be used to read the entries in the tables. |
| 0<br>CT | Checking Trusted. This DECO is currently generating the signature of a Trusted Descriptor. This may be to sign, re-sign, or check the signature. |

# 13.230 DECOa Debug Job Pointer (D0DJP - D2DJP)

## 13.230.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaDJP | 8_0E08h + (a × 1_0000h) | For DECOa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFGR). |

## 13.230.2 Function

The DECOa Debug Job, DECOa Debug DECO, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see Register-based service interface). Note that this register is read-only.

## 13.230.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | JDPTR | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JDPTR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | JDPTR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.230.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved |
| 39-0<br>JDPTR | Job Descriptor Pointer. |

# 13.231 DECOa Debug Shared Pointer (D0SDP - D2SDP)

## 13.231.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaSDP | 8_0E10h + (a × 1_0000h) | For DECOa. For the order that the two 32-bit halves of this register appear in memory, see the DWT bit description in Master Configuration Register (MCFG R). |

## 13.231.2 Function

The DECOa Debug Job, DECOa Debug DECO, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see Register-based service interface). Note that this register is read-only.

## 13.231.3 Diagram

| Bits | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | | | | | SDPTR | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SDPTR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | SDPTR | | | | | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.231.4 Fields

| Field | Function |
|---|---|
| 63-40<br>— | Reserved |
| 39-0<br>SDPTR | Shared Descriptor Pointer. |

## 13.232 DECOa Debug_ICID, most-significant half (D0DIR_MS - D2DIR_MS)

### 13.232.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaDIR_MS | 8_0E18h + (a × 1_0000h) | For DECOa. |

### 13.232.2 Function

The DECOa Debug Job, DECOa Debug_DBG, DECOa Debug Job Pointer, DECOa Debug ICID, and DECOa Debug Shared Pointer registers are intended to assist in debugging when a DECO appears to be hung. This register is read-only. Although the registers can be read by software at any time, software is likely to obtain inconsistent data if these registers are read while DECO continues to execute new descriptors because the registers may be updated before the software has finished reading all the registers. Another mechanism is available for debugging a descriptor once a suspect descriptor has been identified (see Register-based service interface).

### 13.232.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | NON_SEQICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | Reserved | | | | SEQICID | | | | | | | | | | | |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.232.4 Fields

| Field | Function |
|---|---|
| 31-28<br>— | Reserved |
| 27-16<br>NON_SEQICID | DECO Non-SEQ ICID. This is the ICID value that is asserted during a DMA transaction that is related to normal descriptor operations other than Input or Output SEQ sequences.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |
| 15-12<br>— | Reserved. |
| 11-0<br>SEQICID | DECO SEQ ICID. This is the ICID value that will be asserted during a DMA transaction that is related to a normal descriptor Input or Output SEQ sequence.<br>Note, in this SoC the valid range of this field is limited to the least signficant 8 bits. |

## 13.233 Sequence Output Length Register (SOL0 - SOL2)

### 13.233.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| SOLa | 8_0E20h + (a × 1_0000h) | For DECOa. Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.233.2 Function

The Sequence Out Length Register is used to specify the amount of data for an Output Sequence (i.e., a series of SEQ STORE or SEQ FIFO STORE commands within a single descriptor). See SEQ vs non-SEQ commands for a discussion of sequences. See Using sequences for fixed and variable length data for a discussion of the use of the SOL register in Output Sequences. The SEQ OUT PTR command can be used to load the SOL register. The SOL Register can be read or written via the MATH Command (see SRC0 and DEST fields in MATH and MATHI Commands). When the DECO is under direct control of software (see Register-based service interface) this register is accessible at the addresses shown above.

### 13.233.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SOL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SOL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.233.4 Fields

| Field | Function |
|---|---|
| 31-0<br>SOL | Output Sequence Length. This value is used in output data sequences.<br>SOL can also be used as a general purpose math register. |

## 13.234 Variable Sequence Output Length Register (VSOL0 - VSOL2)

### 13.234.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| VSOLa | 8_0E24h + (a × 1_0000h) | For DECOa. Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.234.2 Function

The Variable Sequence Out Length Register is used to specify a variable amount of data for an Output Sequence (i.e., a series of SEQ STORE or SEQ FIFO STORE commands within a single descriptor). See SEQ vs non-SEQ commands for a discussion of sequences. See Using sequences for fixed and variable length data for a discussion of the use of the VSOL register in Output Sequences. The VSOL Register can be read or written via the MATH Command (see SRC0, SRC1 and DEST fields in MATH and MATHI Commands). When the DECO is under direct control of software (see Register-based service interface this register is accessible at the addresses shown above.Note that VSOL is actually a 64-bit register when accessed via a descriptor, but the 32 most-significant bits are accessible from the IP bus as the UVSOL register, located at offset E34h.

### 13.234.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | VSOL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | VSOL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.234.4 Fields

| Field | Function |
|---|---|
| 31-0<br>VSOL | This value is used in variable-length output data sequences. VSOL/UVSOL can also be used as a general purpose math register. See UVSOL register. |

## 13.235 Sequence Input Length Register (SIL0 - SIL2)

### 13.235.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| SILa | 8_0E28h + (a × 1_0000h) | For DECOa. Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.235.2 Function

The Sequence In Length Register is used to specify the amount of data for an Input Sequence (i.e., a series of SEQ LOAD or SEQ FIFO LOAD commands within a single descriptor). See Section SEQ vs non-SEQ commands for a discussion of sequences. See Using sequences for fixed and variable length data for a discussion of the use of the SIL register in Input Sequences. The SIL Register can be read or written via the MATH Command (see SRC0 and DEST fields in MATH and MATHI Commands). When the DECO is under direct control of software (see Register-based service interface this register is accessible at the addresses shown above. This register can also be loaded by the SEQ IN PTR command.

### 13.235.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SIL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | SIL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.235.4 Fields

| Field | Function |
|---|---|
| 31-0<br>SIL | This value is used in input data sequences. SIL can also be used as a general purpose math register. |

## 13.236 Variable Sequence Input Length Register (VSIL0 - VSIL2)

### 13.236.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| VSILa | 8_0E2Ch + (a × 1_0000h) | For DECOa. Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.236.2 Function

The Variable Sequence In Length Register is used to specify a variable amount of data for an Input Sequence (i.e., a series of SEQ LOAD or SEQ FIFO LOAD commands within a single descriptor). See Section SEQ vs non-SEQ commands for a discussion of sequences. See Using sequences for fixed and variable length data for a discussion of the use of the VSIL register in Input Sequences. This register is also loaded when RIF is executed for a Shared Descriptor. The VSIL Register can be read or written via the MATH Command (see SRC0, SRC1, and DEST fields in MATH and MATHI Commands). When the DECO is under direct control of software (see Register-based service interface) this register is accessible at the addresses shown above. Note that VSIL is actually a 64-bit register when accessed via a descriptor, but the 32 most-significant bits are accessible from the IP bus as the UVSIL register, located at offset E38h.

### 13.236.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | VSIL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | VSIL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.236.4 Fields

| Field | Function |
|---|---|
| 31-0<br>VSIL | This value is used in variable-length input data sequences. VSIL/UVSIL can also be used as a general purpose math register. See UVSIL register. |

## 13.237 Protocol Override Register (D0POVRD - D2POVRD)

### 13.237.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| DaPOVRD | 8_0E30h + (a × 1_0000h) | For DECOa. Accessible only when RQDa and DENa are asserted in DECORR. |

## 13.237.2 Function

The DECO Protocol Override Register is used to override the PDB options for certain built-in network protocols. can be read or written via the MATH Command (see SRC0, SRC1 and DST fields in MATH and MATHI Commands) and it can be written via a LOAD IMMEDIATE command (see DST value 07h, Class=11 in LOAD commands). Note that DPOVRD can also be used as a general purpose math register. The format of the register is specific to the protocol (see table below). When the DECO is under direct control of software (see Register-based service interface) this register is accessible at the addresses shown above.

**Table 13-9. DECO Protocol Override Register - Formats for Various Networking Protocols**

| Protocol | Format Diagram |
|---|---|
| IPsec ESP Encapsulation | See Overriding ESP Transport (and legacy Tunnel) PDB content with the DECO Protocol Override Register |
| IPsec ESP Decapsulation | See Overriding ESP Transport (and legacy Tunnel) PDB content with the DECO Protocol Override Register |
| SSL 3.0 / TLS 1.0 Encapsulation | See Overriding the PDB for SSL, TLS, and DTLS Encapsulation |
| TLS 1.1 / 1.2 Encapsulation | See Overriding the PDB for SSL, TLS, and DTLS Encapsulation |
| DTLS Encapsulation | See Overriding the PDB for SSL, TLS, and DTLS Encapsulation |

## 13.237.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | DPOVRD | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | DPOVRD | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.237.4 Fields

| Field | Function |
|---|---|
| 31-0<br>DPOVRD | This register can be written to override various PDB settings. The format used for the DPOVRD register depends on the particular protocol operation that is executed. Table 13-9 contains links to the different format diagrams. |

## 13.238 Variable Sequence Output Length Register; Upper 32 bits (UVSOL0 - UVSOL2)

### 13.238.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| UVSOLa | 8_0E34h + (a × 1_0000h) | For DECOa. Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.238.2 Function

VSOL is actually a 64-bit register when accessed via a descriptor, but when accessed via the IP bus the least-significant 32 bits are accessed as the VSOL register, located at offset E24h, and the most-significant 32 bits are accessible as the UVSOL register, located at offset E34h.

### 13.238.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | UVSOL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | UVSOL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 13.238.4 Fields

| Field | Function |
|---|---|
| 31-0<br>UVSOL | This value is used in variable-length output data sequences. VSOL/UVSOL can also be used as a general purpose math register. See VSOL register. In some older versions of SEC the UVSOL register did not exist, i.e. the VSOL register was only 32 bits. In those older versions when VSOL was the destination of a right-shift MATH command the source was first truncated to 32 bits and then 0 bits were shifted in from the left. For backward compatibility, that will continue to be the case for Math lengths of 1, 2 or 4 bytes. But when the Math length is 8 bytes, all 64 bits of the source will be copied into UVOL/VSOL and then 0 bits will be shifted in from the left. |

## 13.239 Variable Sequence Input Length Register; Upper 32 bits (UVSIL0 - UVSIL2)

### 13.239.1 Offset

For a = 0 to 2:

| Register | Offset | Description |
|---|---|---|
| UVSILa | 8_0E38h + (a × 1_0000h) | For DECOa. Accessible only when RQDa and DENa are asserted in DECORR. |

### 13.239.2 Function

VSIL is actually a 64-bit register when accessed via a descriptor, but when accessed via the IP bus the least-significant 32 bits are accessed as the VSIL register, located at offset E2Ch, and the most-significant 32 bits are accessible as the UVSIL register, located at offset E38h.

## 13.239.3 Diagram

| Bits | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | UVSIL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | UVSIL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 13.239.4 Fields

| Field | Function |
|---|---|
| 31-0<br>UVSIL | This value is used in variable-length input data sequences. VSIL/UVSIL can also be used as a general purpose math register. See VSIL register. In some older versions of SEC the UVSIL register did not exist, i.e. the VSIL register was only 32 bits. In those older versions when VSIL was the destination of a right-shift MATH command the source was first truncated to 32 bits and then 0 bits were shifted in from the left. For backward compatibility, that will continue to be the case for Math lengths of 1, 2 or 4 bytes. But when the Math length is 8 bytes, all 64 bits of the source will be copied into UVIL/VSIL and then 0 bits will be shifted in from the left. |

# Appendix A
# Revision History

Because this is the initial publication of the document, there are no changes.

# Appendix B
# Acronyms and abbreviations

### Table B-1. Acronyms and abbreviated terms

| Term | Meaning |
|---|---|
| AAD | Additional Authenticated Data |
| AES | Advanced Encryption Standard - 128-bit block encryption algorithm, using a 128, 192 or 256-bit key. |
| AXI | AMBA Advanced eXtensible Interface (AXI) Protocol Specification. Defined by ARM Ltd. |
| BMan | Buffer Manager<br>A companion block to SEC that manages the buffers that supply data and receive results through SEC's Queue Manager Interface |
| CBC | Cipher Block Chaining<br>An encryption mode of operation. This is one of the official modes of operation specified for DES and AES. |
| CCB | Cryptographic Control Block<br>A logic module within SEC |
| CCM | Counter with CBC-MAC Mode<br>An authenticated encryption mode of operation. (Also known as CBC-MAC for CTR mode.) |
| CFB | Cipher FeedBack<br>An encryption mode of operation. This is one of the official modes of operation specified for DES and AES. |
| CHA | Cryptographic Hardware Accelerator<br>One of the hardware accelerators used in SEC |
| CRJD | Control Replacement Job Descriptor |
| CSP | Critical Security Parameter<br>Security related information (such as secret and private cryptographic keys or authentication data such as passwords and PINs) whose disclosure or modification can compromise the security of a cryptographic module. (See FIPS140-2) |
| CTR | Counter mode<br>An encryption mode of operation used with AES |
| DECO | Descriptor Controller<br>A logic module within SEC |
| DEK | Data Encryption Key. |
| DES | Data Encryption Standard<br>64-bit block encryption algorithm, using a 64-bit key. |
| 3DES | Triple DES<br>64-bit block encryption algorithm, using a 128 or 196-bit key. |

*Table continues on the next page...*

## Table B-1. Acronyms and abbreviated terms (continued)

| Term | Meaning |
|---|---|
| DPAA | Datapath Acceleration Architecture, version 1 |
| DRBG | Deterministic Random Bit Generator<br>A deterministic algorithm that generates a sequence of numbers whose values are statistically random. Sometimes called "PRNG" (pseudorandom number generator). |
| ECB | Electronic Code Book<br>An encryption mode of operation. This is one of the official modes of operation specified for DES and AES. |
| FQ | Frame Queue |
| HAB | High Assurance Boot software |
| HMAC | A hashing mode of operation used to implement a Message Authentication Code |
| ICID | Isolation Context Identifier |
| IPAD | Inner padding defined for HMAC |
| ICV | Integrity Check Value<br>A checksum or message digest that allows detection of errors or changes in data. |
| IJD | Inline Job Descriptor |
| IV | Initialization Vector<br>A value used to initialize some encryption modes of operation |
| JD | Job Descriptor |
| JDKEK | Job Descriptor Key Encryption Key |
| JQC | Job Queue Controller<br>The hardware that schedules jobs received from the Job Rings, Queue Manager Interface and RTIC |
| JR | Job Ring |
| LIODN | Logical Input Output Device Number |
| MD5 | A message digest algorithm returning a 128-bit hash value |
| MDHA | Message Digest Hardware Accelerator (hashing accelerator block) |
| NS | Non-secure indication<br>NS = 0 is secure. This signal is generated by the TrustZone feature implemented in some ARM processors. The Central Security Unit (CSU) generates this signal for other bus masters. |
| NVTK | Non-volatile Test Key |
| OFB | Output FeedBack<br>An encryption mode of operation. This is one of the official modes of operation specified for DES and AES. |
| OPAD | Outer padding defined for HMAC |
| OTPMK | One-time-programmable Master Key |
| PKHA | Public Key Hardware Accelerator (ECC, RSA, DH, DSA) |
| POR | Power On Reset. |
| PRNG | Pseudo Random Number Generator<br>A deterministic algorithm that generates a sequence of numbers whose values are statistically random. See DRBG. |
| PSP | Public Security Parameter<br>Security-related public information whose modification can compromise the security of a cryptographic module. |

*Table continues on the next page...*

## Table B-1. Acronyms and abbreviated terms (continued)

| Term | Meaning |
|---|---|
| QMan | Queue Manager<br>A companion block to SEC that organizes data streams into queues of PDUs that are processed through SEC's Queue Manager Interface |
| QI | Queue (Manager) Interface<br>A logic block within SEC that interacts with the BMan and QMan blocks external to SEC |
| RNG | Random Number Generator<br>A hardware module within SEC that generates random numbers based on the interaction of two free running ring oscillators and uses these random numbers to seed a DRBG. |
| RJD | Replacement Job Descriptor |
| RTIC | Run-Time Integrity Checker<br>A logic block within SEC that generates a tamper alarm if the integrity of selected memory areas have been compromised |
| SD | Shared Descriptors |
| SEC | Security Engine (Also known as Cryptographic Acceleration and Assurance Module) |
| SecMon | Security Monitor<br>A companion block to SEC that detects security violations and maintains security state. |
| SHA-1 | A message digest algorithm defined in FIPS 180-2 returning a 160-bit hash value. |
| SHA-224 | A message digest algorithm defined in FIPS 180-2 returning a 224-bit hash value. |
| SHA-256 | A message digest algorithm defined in FIPS 180-2 returning a 256-bit hash value. |
| SHA-384 | A message digest algorithm defined in FIPS 180-2 returning a 384-bit hash value. |
| SHA-512 | A message digest algorithm defined in FIPS 180-2 returning a 512-bit hash value. |
| SSP | Sensitive Security Parameter<br>Data whose integrity must be protected |
| SWRST | Software Reset<br>Register resets caused by writing 1 to the SWRST field in the MCFGR register. |
| TD | Trusted Descriptor |
| TDSK | Trusted Descriptor Signing Key |
| TDKEK | Trusted Descriptor Key Encryption Key |
| TRK | Trusted Root Key |
| ZMK | Zeroizable Master Key |

# Appendix C Glossary

### Table C-1. Glossary of terms

| Term | Description |
|---|---|
| Alleged RC4 | A stream cipher that is compatible with RC4. |
| Black blob | A blob whose input data when exporting and whose output when importing was assumed to be a black key. When exporting a black blob, the input data is first decrypted using the JDKEK (if the BLOB Command was in a job descriptor) or TDKEK (if the BLOB command was in a trusted descriptor) before being encrypted with the blob key. When importing a Black Blob, the data blob is first decrypted with the Blob Key before being encrypted using the JDKEK (if the BLOB Command was in a job descriptor) or TDKEK (if the BLOB Command was in a trusted descriptor). (See *Red Blob*.) |
| Black key | A key that has been encrypted using either the JDKEK or the TDKEK. (See *Red Key*.) |
| Blob | As used in this Block Guide the term 'blob' refers to a cryptographically protected data object consisting of a Blob Key encrypted with a Blob Key Encryption Key, a Data Blob encrypted with a Blob Key, and the MAC Tag resulting from the AES-CCM encryption of the Data Blob. |
| Blob key | The 256-bit random number used for AES-CCM encryption of the data portion of a blob. |
| Blob key encryption key | The Blob Key Encryption Key (BKEK) is a 256-bit key used when encrypting cryptographic Blobs exported from memory. It is intended for use in protecting the confidentiality and integrity of this data. The BKEK is derived from the Master Key or Non-volatile Test Key, a constant embedded in the SEC Descriptor that initiated the Blob operation, the Security mode and the Blob type. (See *Master Key.*) |
| Critical security parameter | A critical security parameter (CSP) is security-related information (e.g., secret and private cryptographic keys, and authentication data such as passwords and PINs) whose disclosure or modification can compromise the security of a cryptographic module. [from FIPS PUB 140-3 (DRAFT)] |
| Data encryption key | A data encryption key is a key that can be referenced in a descriptor as a cryptographic key and that is not one of other keys defined in this glossary. Some examples are: a symmetric key used for encryption or decryption of session data, a private key used for signing data, a public key used for verifying a signature, a private or public key used in a key establishment operation, an HMAC key. |
| Decrypt key | A decrypt key is used for decrypting data to yield plaintext (unencrypted data). Some cryptographic algorithms (e.g. AES) successively modify the cryptographic key during the steps of the cryptographic operation; therefore the decrypt form of the key is different from the encrypt form of the key. |
| Descriptor | A descriptor is a sequence of commands that causes SEC to perform cryptographic functions. There are three types of descriptors: job descriptors, shared descriptors, and trusted descriptors. Shared descriptors and trusted descriptors are actually special forms of job descriptors. Note that this usage of the term 'descriptor' is not related to the term 'frame descriptor'. |
| Fail mode | SEC clears its CSP registers (e.g. key registers) upon entrance to *Fail Mode*. SEC enters Fail Mode when the SecMon's security state machine enters its Fail State. This could be due to the detection of tampering, scan or JTAG testing or due to the failure of a security module. |
| Frame descriptor | SEC inputs and outputs data over its Queue Interface in data structures called 'frame descriptors'. Note that the term 'frame descriptor' is not related to the terms 'job descriptor', 'trusted descriptor', or 'shared descriptor'. |

*Table continues on the next page...*

## Table C-1. Glossary of terms (continued)

| Term | Description |
|---|---|
| Hash | A hash is the message digest resulting from a hashing operation, such as SHA-1, SHA-256 or MD5. A cryptographic hashing operation is a collision-resistant one-way function that yields a fixed-length bit string from a variable length input. A function is collision-resistant if it is difficult to find two input strings that yield the same Message Digest. A function is one-way if it is computationally infeasible to calculate the input, given only the Message Digest. |
| Job descriptor | The term 'job descriptor' means a descriptor that is not a shared descriptor or a trusted descriptor. Unlike a shared descriptor, a job descriptor can reference another descriptor, and unlike a trusted descriptor, a job descriptor is not signed. |
| Job descriptor key encryption key | The Job Descriptor Key Encryption Key (JDKEK) is a 256-bit key used to protect the confidentiality of Data Encryption Keys (DEK) referenced by job descriptors. A new JDKEK value is generated by the SEC's RNG at each POR, and is used throughout the current power-on cycle to encrypt or decrypt DEKs "on-the-fly" during job descriptor processing. (See *Trusted Descriptor Key Encryption Key*. |
| Key encryption key | A Key Encryption Key (KEK) is a cryptographic key used to encrypt other cryptographic keys. SEC supports various KEKs that are used in different circumstances. (See *JDKEK, TDKEK*) |
| Link table | A link table is also referred to by the term "Scatter/Gather Table". |
| Master key | The master key is a 256-bit secret value that SEC receives from the SecMon. (See *Non-volatile Test Key*, *OTP Master Key*, *Zeroizable Master Key*, and *Blob Key Encryption Key*.) |
| Message digest | A message digest (also called a hash) is a fixed-size string that is the result of computing a cryptographic one-way function of some input data. |
| Non-volatile test key | The Non-volatile Test Key (NVTK) is a 256-bit key hardwired into SEC. When SEC is in the Non-Secure Mode SEC will use the NVTK to derive Blob key encryption keys, rather than using the secret Master Key. The NVTK value is public knowledge, and is the same in every SOC. It is used for known-answer tests when testing the SEC cryptographic hardware. |
| Non-secure mode | SEC's Non-secure Mode is intended to allow SEC to be tested without compromising the security of sensitive data. In this mode a known version of the BKEK (based on the Non-volatile Test Key) is used for exporting and importing Blobs. Therefore any Blobs exported while in Secure Mode or Trusted Mode cannot be successfully imported while in Non-secure Mode. |
| OTP master key | The OTP Master Key (OTPMK) is a 256-bit secret value stored in one-time-programmable storage on the SOC. The value is generally written to the one-time-programmable storage while the SOC is in the factory. The OTPMK bits are protected with a lock that, when set, prevents modifying the value. In some configurations SecMon will use the OTPMK to derive the value of the master key that the SecMon supplies over a private bus to SEC. Its value cannot otherwise be read, sensed or scanned. |
| Processor | A processor is a bus master capable of executing software. |
| Public security parameter | A public security parameter (PSP) is security-related public information whose modification can compromise the security of a cryptographic module. [from FIPS PUB 140-3 (DRAFT)] The Trusted Root Key is a PSP. |
| Read safe | A read-safe transaction reads a full aligned burst of data, even if not all of the data is needed. |
| Red blob | A blob whose data input when exporting is assumed to be not encrypted, and whose data output when importing is not encrypted. (See *Black Blob*) |
| Red key | A key that is not encrypted. (See *Black Key*) |
| Replay | Replay is a type of security attack in which old data is presented by a hacker as if it were new data. For instance, a hacker could replace a new Blob that shows that a software license has expired with an old Blob that indicates that the license is still valid. The term "replay" is sometimes also used to refer to a denial of service attack based upon flooding the system with the same message over and over. If this message is encrypted or cryptographically authenticated, then the attacker may not be able to generate new messages and instead would "replay" a legitimate message that the attacker had snooped from the network. |
| Secure mode | Secure Mode is the normal operating mode of SEC. The Security State Machine within the SecMon determines when SEC is operating in Secure Mode. |

*Table continues on the next page...*

## Table C-1. Glossary of terms (continued)

| Term | Description |
|---|---|
| Security monitor | Security Monitor (SecMon) is a companion logic block to SEC. It implements a security alarm and maintains a security state machine. |
| Sensitive data | Sensitive data is data that should be protected against unauthorized disclosure. |
| Sensitive security parameter | The term 'sensitive security parameters' (SSP) encompasses critical security parameters and public security parameters. [from FIPS PUB 140-3 (DRAFT)] |
| SEQ | Sequence. For most memory referencing descriptor commands SEC implements an auto-incrementing addressing mode using sequence input address and sequence output address registers. This is intended to faciliate the processing of cryptographic networking protocols. |
| Shared descriptor | A shared descriptor is a special type of job descriptor that can be executed only when it is referenced by another descriptor. Shared descriptors are intended to contain data, such as keys and sequence numbers, that are shared by two or more other descriptors. |
| Trusted descriptor | A trusted descriptor is a special type of job descriptor that has some additional access privileges and some additional security protections. Trusted descriptors are protected from modification by means of a signature over the descriptor. SEC verifies the signature before executing the trusted descriptor, and aborts execution if the signature is incorrect. (See *Trusted Descriptor Signing Key*) |
| Trusted descriptor signing key | The *Trusted Descriptor Signing Key* (TDSK) is a 256-bit key used in HMAC-SHA-256 to sign and verify the signature over trusted descriptors. A new TDSK value is generated by the SEC RNG at each POR, and is used throughout the current power-on cycle. SEC will allow TDSK to be used to sign a new trusted descriptor only if the descriptor is submitted via a Job Ring that has AMTD set in its JRaICID register. Otherwise, SEC will use TDSK only to verify the signature over a trusted descriptor, or to update the signature on an existing trusted descriptor that has modified itself during its execution. |
| Trusted descriptor key encryption key | The Trusted Descriptor Key Encryption Key (TDKEK) is a 256-bit key that can be used to protect the confidentiality of Data Encryption Keys (DEKs) referenced by trusted descriptors. A new TDKEK value is generated by the SEC's RNG at each POR, and is used throughout the current power-on cycle to encrypt or decrypt DEKs "on-the-fly" during trusted descriptor processing. (See *Job Descriptor Key Encryption Key*) |
| Trusted mode | Trusted Mode is a special operating mode of SEC. The Security State Machine within the SecMon determines when SEC is operating in Trusted Mode. This mode is implemented so that trusted boot-time software, or a hypervisor or TrustZone Secure World software can store data in and retrieve data from Trusted Mode Blobs that are not accessible to software running while SEC is in Secure Mode or Non-Secure Mode. |
| Trusted root key | The Trusted Root Key is a public signature key used by HAB to verify the signature over the Command Sequence File. The key could be RSA (probably 2048 bits) or ECC-DSA (probably 511 bits). The integrity and authenticity of this key is protected by placing a SHA-256 hash of this key in fuses on the SOC. The fuses are located in a bank with a lock fuse that, when set, prevents any changes to the hash value. |
| Word | A word of memory or a one-word register contains 32 bits. |
| Write safe | A write-safe transaction writes 0s to addresses past the targeted locations up to the next 8, 16, 32 or 64-byte address boundary, depending upon the offset within the cacheline. |
| Zeroizable master key | The Zeroizable Master Key (ZMK) is a 256-bit key stored in a register in the low-power domain of SecMon. In some configurations and security states SecMon will use the ZMK to derive the value of the Master Key that SecMon supplies over the snvs_master_key signal to SEC. Its value cannot otherwise be read or scanned. The value can be generated by the SEC RNG, and can be loaded automatically by hardware. The value can be zeroized when a tamper event is detected. (See *Master Key*.) |
| Zeroize | A set of data storage locations is *zeroized* by overwriting the storage locations with a value (not necessarily 0) that is independent of the previous content of the storage locations. |

***How to Reach Us:***

**Home Page:**
nxp.com

**Web Support:**
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, Freescale, the Freescale logo, and Layerscape are trademarks of NXP B.V. All other product or service names are the property of their respective owners.  ARM, AMBA, ARM Powered, Cortex, and TrustZone are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. ARM7 is a trademark of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates.