

Project Zipline Key Manipulation Engine (KME)

Micro Architecture Specification

**Authors:**

**Microsoft Corporation**

**Broadcom Corporation**

**Revision History**

|  |  |
| --- | --- |
| Date | Description |
| 04/03/2019 | Version 1.0 |
|  |  |

**License**

Contributions to this Specification are made under the terms and conditions set forth in the Open Web Foundation Contributor License Agreement (“OWF CLA 1.0”) (“Contribution License”) by:

Microsoft Corporation

Broadcom Corporation

Usage of this Specification is governed by the terms and conditions set forth in Open Web Foundation Final Specification Agreement (“OWFa 1.0”) (“Specification License”).

**Note**: The following clarifications, which distinguish technology licensed in the Contribution License and/or Specification License from those technologies merely referenced (but not licensed), were accepted by the Incubation Committee of the OCP:

**None.**

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

[Project Zipline Key Manipulation Engine (KME) 1](#_Toc7443261)

[Micro Architecture Specification 1](#_Toc7443262)

[1 Overview 6](#_Toc7443263)

[1.1 Features 6](#_Toc7443264)

[1.2 Interfaces 6](#_Toc7443265)

[1.2.1 Clock, Reset & Miscellaneous 6](#_Toc7443266)

[1.2.2 AXI-4 KME Stream Slave Interface 7](#_Toc7443267)

[1.2.3 AXI-4 KME Stream Master Key TLV Interfaces (x8) 8](#_Toc7443268)

[1.2.4 APB 3.0 Slave Register Programming Interface 9](#_Toc7443269)

[2 Cipher Operations Summary 10](#_Toc7443270)

[2.1 AES Cipher Summary 10](#_Toc7443271)

[2.2 Key Derivation Function (KDF) 10](#_Toc7443272)

[2.2.1 PRF For uses with KDF 11](#_Toc7443273)

[2.3 Deterministic Random Number Generator (DRNG) 11](#_Toc7443274)

[3 Mega TLV Formats (INPUT) 12](#_Toc7443275)

[3.1 RQE TLV 13](#_Toc7443276)

[3.1.1 Word 0 (RQE0) 13](#_Toc7443277)

[3.1.2 Word 1 (RQE1) 13](#_Toc7443278)

[3.2 AUX Command TLV 14](#_Toc7443279)

[3.2.1 Word 0 (AUX\_CMD Header) 14](#_Toc7443280)

[3.2.2 Word 1 (DEBUG/FRAME) 15](#_Toc7443281)

[3.2.3 Word 2 (CRYPTO) 15](#_Toc7443282)

[3.2.4 Optional Words 16](#_Toc7443283)

[4 GUID TLV (Optional) 30](#_Toc7443284)

[4.1 Word 1/2/3/4 (GUID) 30](#_Toc7443285)

[5 Key TLV (Output) 31](#_Toc7443286)

[5.1.1 Word 0 31](#_Toc7443287)

[5.1.2 Later Words 32](#_Toc7443288)

[6 Endian Swapping 34](#_Toc7443289)

[7 Block Overview 35](#_Toc7443290)

[7.1 Internal TLV 36](#_Toc7443291)

[7.2 KOP Block Diagram 38](#_Toc7443292)

[7.3 Key Vault 39](#_Toc7443293)

[7.3.1 KIM Validations 40](#_Toc7443294)

[7.4 Label Table 40](#_Toc7443295)

[7.5 Summary of Blocks 41](#_Toc7443296)

[7.6 Key Derivation Function (KDF) 44](#_Toc7443297)

[7.6.1 SHA-256-HMAC 46](#_Toc7443298)

[7.6.2 KDF Stream Generator 48](#_Toc7443299)

[7.6.3 KDF Engine Controls 50](#_Toc7443300)

[7.6.4 KDF Test Mode 51](#_Toc7443301)

[7.7 Random GUID Generation 51](#_Toc7443302)

[7.7.1 CTR-DRBG Algorithm 51](#_Toc7443303)

[7.8 AES Core 52](#_Toc7443304)

[8 Interrupts 54](#_Toc7443305)

[9 Performance Counters 55](#_Toc7443306)

[10 Security Considerations 56](#_Toc7443307)

[11 Head of Line Decoupling FIFO 57](#_Toc7443308)

[12 ECC 58](#_Toc7443309)

[13 Operational Spreadsheet 59](#_Toc7443310)

[14 References 60](#_Toc7443311)

Table of Figures

[Figure 1 - Mega TLV Format 13](#_Toc7443312)

[Figure 2 - RQE TLV Format 14](#_Toc7443313)

[Figure 3 – Generic AUX Command TLV Format 15](#_Toc7443314)

[Figure 4 – AUX CMD Format 17](#_Toc7443315)

[Figure 5 - AUX\_CMD\_GUID-IV Format 18](#_Toc7443316)

[Figure 6 - AUX\_CMD\_GUID Format 19](#_Toc7443317)

[Figure 7 - AUX\_CMD IV Format 20](#_Toc7443318)

[Figure 8 GUID TLV Format 31](#_Toc7443319)

[Figure 9 - KME Top Level Block Diagram 36](#_Toc7443320)

[Figure 10 - Internal TLV Diagram 38](#_Toc7443321)

[Figure 11 - Key Operations Processor Block Diagram 39](#_Toc7443322)

[Figure 12 KDF Diagram 45](#_Toc7443323)

[Figure 13 - Dual SHA-256 HMAC 47](#_Toc7443324)

[Figure 14 - SHA-2 Core 48](#_Toc7443325)

[Figure 15 - NIST KDF Diagram 49](#_Toc7443326)

[Figure 16 - Internal State (Seed) of CTR-DRBG 52](#_Toc7443327)

[Figure 17 - AES Core 54](#_Toc7443328)

Table of Tables

[Table 1 - Clock, Reset and Miscellaneous Interface 8](#_Toc7443329)

[Table 2 - AXI-4 KME Stream Slave Interface 8](#_Toc7443330)

[Table 3 - AXI-4 KME Stream Master Key TLV Interfaces 9](#_Toc7443331)

[Table 4 - APB Slave Register Programming Interface 10](#_Toc7443332)

[Table 5 – Encrypted Key Blob Algorithm support 11](#_Toc7443333)

[Table 6 KDF Function Summary 11](#_Toc7443334)

[Table 7 - Information needed to process different Hash types and modes 12](#_Toc7443335)

[Table 8 - DRNG NIST Parameters 12](#_Toc7443336)

[Table 9 – RQE0 TLV Word 0 14](#_Toc7443337)

[Table 10 – RQE1 TLV Word 1 14](#_Toc7443338)

[Table 11 – AUX\_CMD Header TLV Word 0 15](#_Toc7443339)

[Table 12 – Debug/Frame TLV Word 1 16](#_Toc7443340)

[Table 13 – Crypto TLV Word 2 17](#_Toc7443341)

[Table 14 - AUX\_KEY Format 18](#_Toc7443342)

[Table 15 – GUID0/1/2/3 TLV Word 4/5/6/7 19](#_Toc7443343)

[Table 16 - IVTWEAK0/1 TLV Word 8/9 19](#_Toc7443344)

[Table 17 – GUID0/1/2/3 TLV Word 1/2/3/4 31](#_Toc7443345)

[Table 18 - Key TLV Word 0 33](#_Toc7443346)

[Table 19 - Key TLV Later Words 33](#_Toc7443347)

[Table 20 – DEK and DAK Validation Cases 41](#_Toc7443348)

[Table 21 KDF Engine Controls 51](#_Toc7443349)

[Table 22 - Interrupt Table 55](#_Toc7443350)

# Overview

The Key Manipulation Engine (KME) is a single clock domain block that provides key decryption, generation and manipulation services through one incoming and eight outgoing 64-bit wide AXI4 Streaming Interface. The register space is managed by an APB Interface with the aid of a RBUS internal translation.

## Features

* Supports 64-bit at 800 MHz AXI Stream master and slave interface.
* Supports 32-bit at 800 MHz APB slave interface.
* Secure key storage for up to 8192 keys of length 256 bits each. It can also hold a mix of 256/512-bit keys with reduced capacity.
* Provides Physical Function (PF) and Virtual Function (VF) Validation.
* Key Decryption Services via AES-256-GCM.
* Key Generation Services using Key Derivation Function (KDF).
* KDF’s underlying Pseudo Random Function (PRF) is HMAC-SHA-256.
* Random GUID through a NIST compliant Deterministic Random Bit Generator (DRBG).
* Head of Line (HOL) blocking FIFO to accommodate any single engine stalling for 20µs.
* Supports 128ns Mega Command rate at input.
* All AES Engines are pipelined to do one round per clock cycle to meet performance.
* Two Pipelined SHA-2 Engines does four rounds per cycle to meet performance.

## Interfaces

There are ten major and one miscellaneous interface on KME.

### Clock, Reset & Miscellaneous

|  |  |  |
| --- | --- | --- |
| **Signal Name** | **Direction** | **Function** |
| clk | Input | Main clock at 800 MHz |
| rst\_n | Input | Active low main reset |
| kme\_interrupt | Output | Active High Interrupt on DRBG seed expiry |
| scan\_en | Input | Scan signals |
| scan\_mode | Input | Scan signals |
| scan\_rst\_n | Input | Scan signals |
| ovstb | Input | Memory support signals |
| lvm | Input | Memory support signals |
| mlvm | Input | Memory support signals |

Table 1 - Clock, Reset and Miscellaneous Interface

### AXI-4 KME Stream Slave Interface

|  |  |  |
| --- | --- | --- |
| **Signal Name** | **Direction** | **Function** |
| kme\_ib\_tvalid | Input | Data transfer valid  0 : Not valid  1 : Valid |
| kme\_ib\_tready | Output | Slave data acceptance for current cycle:  0 : Not Accepted  1 : Accepted |
| kme\_ib\_tdata[63:0] | Input | Input data |
| kme\_ib\_tstrb[7:0] | Input | Byte strobes corresponding to:  TSTRB[0] : TDATA[7:0]  TSTRB[1] : TDATA[15:8]  …  TSTRB[7] : TDATA[63:56]    Data Byte Valid:  0 : Not valid  1 : Valid    Only thermometer codes allowed (e.g. 0000\_0001 through 1111\_1111) |
| kme\_ib\_tlast | Input | Last data phase of the data block |
| kme\_ib\_tid[0:0] | Input | Data block identifier |
| kme\_ib\_tuser[7:0] | Input | TUSER[0] : Start of TLV (first beat)  TUSER[1] : End of TLV (last beat)  TUSER[7:2] : All 0 |

Table 2 - AXI-4 KME Stream Slave Interface

### AXI-4 KME Stream Master Key TLV Interfaces (x8)

<**dest**> = CCEIP0-3 and CDDIP0-3

|  |  |  |
| --- | --- | --- |
| **Signal Name** | **Direction** | **Function** |
| kme\_<**dest**>\_ob\_tvalid | Output | Data transfer valid  0 : Not valid  1 : Valid |
| kme\_<dest>\_ob \_tready | Input | Slave data acceptance for current cycle:  0 : Not Accepted  1 : Accepted |
| kme\_<**dest**>\_ob\_tdata[63:0] | Output | Input data |
| kme\_<**dest**>\_ob \_tstrb[7:0] | Output | Byte strobes corresponding to:  TSTRB[0] : TDATA[7:0]  TSTRB[1] : TDATA[15:8]  …  TSTRB[7] : TDATA[63:56]    Data Byte Valid:  0 : Not valid  1 : Valid    Only thermometer codes allowed (e.g. 0000\_0001 through 1111\_1111) |
| kme\_<**dest**>\_ob \_tlast | Output | Last data phase of the data block |
| kme\_<**dest**>\_ob \_tid[0:0] | Output | Data block identifier |
| kme\_<**dest**>\_ob \_tuser[7:0] | Output | TUSER[0] : Start of TLV (first beat)  TUSER[1] : End of TLV (last beat)  TUSER[7:2] : All 0 |

Table 3 - AXI-4 KME Stream Master Key TLV Interfaces

### APB 3.0 Slave Register Programming Interface

|  |  |  |
| --- | --- | --- |
| **Signal Name** | **Direction** | **Function** |
| apb\_paddr[31:0] | Input | Address Bus. |
| apb\_psel | Input | Peripheral Select. |
| apb\_penable | Input | Enable. Indicates subsequent cycles of a transfer. |
| apb\_pwrite | Input | Write Enable (Write=1/Read=0) |
| apb\_pwdata[31:0] | Input | Write Data Bus. |
| apb\_prdata[31:0] | Output | Read Data Bus. |
| apb\_pready | Output | Read Data Valid/Write has been accepted. |
| apb\_pslverr | Output | Error Status. Valid when psel, penable and pready are 1. |

Table 4 - APB Slave Register Programming Interface

# Cipher Operations Summary

This following section summarizes the crypto algorithms used in the KME.

## AES Cipher Summary

AES-GCM per [NIST SP800-38d](http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf) is implemented in the design for key wrapping and unwrapping

The table below summarizes the cipher algorithm parameters.

|  |  |  |
| --- | --- | --- |
| CiphType | CiphMode | Supported Configuration |
| AES | GCM | Block size = 128b  Key size = 256b (i.e. AES256).  Tag Length = 96bits.  IV = 96 bits.  Additional Authentication Data (AAD) = 0. |

Table 5 – Encrypted Key Blob Algorithm support

## Key Derivation Function (KDF)

KDF defined in [NIST SP 800-108](http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf). This specification uses a PRF (Pseudo Random Function) as an integral part of the KDF. The PRF used in is HMAC-SHA256.

Table KDF Function Summary

|  |  |  |
| --- | --- | --- |
| Parameter | Value | Description |
| **Mode** | **Counter** | **Family of KDF** |
| **MACSupported** | **HMACSHA256** | **PRF used** |
| **length of L** | **256, 512, 768 bits** | **Lengths of derived key material tested** |
| **length of i** | **32 bits** | **Length of the binary representation of the counter i.** |
| **LocationCounter** | **BeforeFixedData** | **Location of counter in the data to be MACed.** |
| **Size of L** | **32 bits** | **Size of L in fixed data calculation** |
| k and i counter encoding | Big-endian | Both counters are encoded as big-endian (MSB first) binary strings when hashed |
| Context Length | 16B or 32B | Supported lengths of Context fields |
| Label Length | 0-33B (byte increments) | Supported lengths of Label field. Note byte 0 is the delimiter in the NIST spec |
| Fixed Label Registers | 8x 33B registers | Label registers. Note: 0x00 must be programmed into byte 33 for |
| Maximum number of iterations (n) | 3 | Maximum number of PRF calls |

### PRF For uses with KDF

HMAC-SHA256 is implemented per [FIPS.198-1](https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf) and [FIPS.180-4](https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf). The table below summarizes the hash algorithm parameters.

|  |  |  |
| --- | --- | --- |
| HashType | HashMode | Information needed for processing |
| SHA-256 | HMAC | Block size is 512b Key size is 256b |

Table 7 - Information needed to process different Hash types and modes

## Deterministic Random Number Generator (DRNG)

A DRNG is used to generate a random number from a seed or entropy. The DRNG as specified in [NIST SP800-90A](http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf) revision1 will be implemented. The DRNG is built using AES-256 block cipher per section 10.2.

|  |  |  |
| --- | --- | --- |
| Parameter | Value | Notes |
| Block Cipher | AES-256 | AES\_ECB (no chaining) |
| Security strength | 256b |  |
| Key size | 256b |  |
| Block length (outlen) | 128b |  |
| Seed Length (seedlen) | 384b | Key size+outlen |
| Df function | None | Not implemented. |
| Prediction resistance | None | Not supported |
| Reseed counter size | 32 bits |  |
| Health Check | Consecutive Values |  |

Table - DRNG NIST Parameters

# Mega TLV Formats (INPUT)

KME operates on Mega Command TLVs. A Mega Command TLV is made of a RQE TLV, AUX Command TLV and an optional GUID TLV. A TLV contains TLV type, TLV Length and Value.

TLV Type identifies the type of header. TLV Length is the sum of TLV Type, TLV Length and Value fields. Start and End of TLV is defined through User Bits (see tuser in Section 1.2.2).

Mega Command TLVs has **five** fixed words (in gray) and a variable number of optional words. The size of the optional words is dependent on the TLV Type and Key Type.



Figure 1 - Mega TLV Format

## RQE TLV

Two fixed words of the RQE TLV are shown below. Only the fields relevant to KME are elaborated in the table below.



Figure 2 - RQE TLV Format

### Word 0 (RQE0)

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| VFValid | 1 | VF Field is valid. |
| TLV Type | 8 | 0: RQE TLV |
| BIP2 | 2 | Parity Bit. |

Table 9 – RQE0 TLV Word 0

### Word 1 (RQE1)

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| PFNum | 4 | PF Number. |
| VFNum | 12 | VF Number. |

Table 10 – RQE1 TLV Word 1

## AUX Command TLV

Three fixed words of the AUX Command TLV are shown below. Only the fields relevant to KME are elaborated in the table below. GUID, IV and Key Blob are in Little Endian.



Figure 3 – Generic AUX Command TLV Format

### Word 0 (AUX\_CMD Header)

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| FrameNumber | 11 | 0 – Simple Command  0-2047 – Compound Commands |
| EngineID | 4 | 0-3 CCEIP, 4-7 CDDIP |
| SeqNum | 8 | Sequence Number. |
| TLVLength | 8 | Length of TLV. It’s in units of 4 bytes. |
| TLV Type | 8 | **21: AUX\_CMD**  **23: AUX\_CMD\_IV**  **24: AUX\_CMD\_GUID**  **25: AUX\_CMD\_GUID\_IV** |
| BIP2 | 2 | Parity Bit. |

Table 11 – AUX\_CMD Header TLV Word 0

### Word 1 (DEBUG/FRAME)

MSB 32-bit of this word is the Debug/Frame Word.

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| Debug | 32 | See Top Microarch for the field breakdown. Only Functional Error Insert is supported for KME. Module ID is **0x1F** for KME. Byte Mask field is overloaded to indicate operation:  0x0: Force CCEIP Key TLV Compare Failure at KME Errors[0]  0x1: Force CRC-32 calculation error on outgoing Key TLVs  **Note:**  This field is ignored when the *disable\_debug\_cmd* OTP bit is set to 1. |
| AuxFrameDataGUID | 1 | 0: GUID is present in AUX\_CMD  1: GUID is present in AUX\_FRAME\_DATA. However, GUID in AUX\_CMD will take precedence (if present) |

Table 12 – Debug/Frame TLV Word 1

### Word 2 (CRYPTO)

MSB 32-bit of this word is the Crypto Word.

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| KeyType | 6 | Key Type Encoded as:  0: No AUX\_KEY  1: AUX\_KEY Only  2: 256b DEK  3: 512b DEK  4: 256b DAK  5: 256b DEK/256b DAK  6: 512b DEK/256b DAK  7: 256b DEK Encrypted  8: 512b DEK Encrypted  9: 256b DAK Encrypted  10: 256b DEK Encrypted, 256b DAK Encrypted  11: 512b DEK Encrypted, 256b DAK Encrypted  12: 256b DEK/DAK Encrypted Combo  13: 512b DEK Encrypted, 256b DAK Encrypted Combo  This field determines the size of Key Blob (see section 3.2.4.5). |

Table 13 – Crypto TLV Word 2

### Optional Words

Below we will expound upon four distinct AUX CMD TLV Types.

#### AUX\_CMD

Only the fields relevant to KME are elaborated in the diagram below. The exact size of this TLV Type is depended on Key Blob size.



Figure 4 – AUX CMD Format

##### Word 3 (AUX\_KEY, Key Blob)

Key Blob is discussed in Section 3.2.4.5.

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| DekKeyOp | 1 | **NOOP**: Do not use KDF.  **KDF**: Use KDF output as DEK Key. |
| KDFMode | 2 | **GUID**: Use GUID passed in AUX\_CMD.  **RGUID**: Generate a random GUID and use it in KDF.  **COMB\_GUID**: Use GUID passed in AUX\_CMD to generate DEK and DAK.  **COMB\_RGUID**: Use random GUID to generate DEK and DAK. |
| DekKeyRef | 14 | Reference to KIM Entry for DEK. |
| DakKeyOp | 1 | **NOOP**: Do not use KDF.  **KDF**: Use KDF output as DAK Key. |
| DakKeyRef | 14 | Reference to KIM Entry for DAK. |

Table 14 - AUX\_KEY Format

#### AUX\_CMD\_GUID\_IV

Only the fields relevant to KME are elaborated in the diagram and table below. The exact size of this TLV Type is depended on Key Blob size.



Figure 5 - AUX\_CMD\_GUID-IV Format

##### Word 3/4/5/6 (GUID)

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| GUID0/1/2/3 | 64 | GUID[255:0] is used in KME.  GUID[255:192] = GUID0[63:0]  GUID[191:128] = GUID1[63:0]  GUID[127:64] = GUID2[63:0]  GUID[63:0] = GUID3[63:0] |

Table 15 – GUID0/1/2/3 TLV Word 4/5/6/7

##### Word 7/8 (IVTWEAK)

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| IVTWEAK0/1 | 64 | IVTWEAK[127:0] is used in KME.  IVTWEAK [127:64] = IVTWEAK0[63:0]  IVTWEAK [63:0] = IVTWEAK1[63:0] |

Table 16 - IVTWEAK0/1 TLV Word 8/9

##### Word 9 (AUX\_KEY, Key Blob)

This is the same as Section 3.2.4.1.1.

#### AUX\_CMD\_GUID

Only the fields relevant to KME are elaborated in the diagram and table below. The exact size of this TLV Type is depended on Key Blob size.



Figure 6 - AUX\_CMD\_GUID Format

##### Word 3/4/5/6 (GUID)

This is the same as Section 3.2.4.2.1.

##### Word 7 (AUX\_KEY, Key Blob)

This is the same as Section 3.2.4.1.1.

#### AUX\_CMD\_IV

Only the fields relevant to KME are elaborated in the diagram and table below. The exact size of this TLV Type is depended on Key Blob size.



Figure 7 - AUX\_CMD IV Format

##### Word 3/4 (IVTWEAK)

This is same as Section 3.2.4.2.2.

##### Word 5 (AUX\_KEY, Key Blob)

This is same as Section 3.2.4.1.1.

#### Key Blob

Below are diagrams for 12 key types. Key Type 0-1 does not have a Key Blob. Here are some things to keep in mind as you read the diagram.

**DEK** = Data Encryption Key. This can be 256 or 512-bits in size.

**DAK** = Data Authentication Key. This can only be 256-bits in size.

**eDEK** = Encrypted Data Encryption Key. This can be 256 or 512-bits in size.

**eDAK** = Encrypted Data Authentication Key. This can only be 256-bits in size.

**eIV** = IV to be used in GCM operation to decrypt DEK. This is always 96-bits.

**eAuthTag** = Tag to verify the GCM decryption of DEK. This is always 96-bits.

**aIV** = IV to be used in GCM operation to decrypt DAK. This is always 96-bits.

**aAuthTag** = Tag to verify the GCM decryption of DAK. This is always 96-bits.

**aeIV** = IV to be used in GCM operation to decrypt DEK/DAK in combo mode. This is always 96-bits.

**aeAuthTag** = Tag to verify the GCM decryption of DEK/DAK in combo mode. This is always 96-bits.

Combo mode is where a single GCM operation is used decrypt both DEK and DAK. Therefore, combo modes have only one IV and one Tag.

Key Types 2 through 6 are unencrypted Key Blobs. Support for these key types is disabled via an OTP bit, as shown below:

*disable\_unencrypted\_keys* = 0 : Key Types 2-6 supported

*disable\_unencrypted\_keys* = 1 : Key Types 2-6 not supported

If the KME encounters an unencrypted key type when *disable\_unencrypted\_keys* is set to 1, the KME will generate an Unsupported Key Type Error (138) in the KME Errors field of the Key TLV.

##### Key Type 2



##### Key Type 3



##### Key Type 4



##### Key Type 5



##### Key Type 6



##### Key Type 7



##### Key Type 8



##### Key Type 9



##### Key Type 10



##### Key Type 11



##### Key Type 12



##### Key Type 13



##### NIST Vector Example

The following example illustrates the mapping between a NIST vector and the Key Blobs. The 256-bit NIST vector shown below is in big-endian byte ordering with byte 0 being first in time (highlighted in yellow) and byte 31 being last in time (highlighted in blue).

Key = b52c\_505a\_37d7\_8eda\_5dd3\_4f20\_c225\_40ea\_1b58\_963c\_f8e5\_bf8f\_fa85\_f9f2\_4925\_05b4

Since the Key Blob is in little endian format, the NIST vector key is delivered to the KME in the following order for Key Type 2:



Internally, the KME performs byte swapping on each 64-bit word of the key blob, as described in Endian Swapping, and the key is restored to the NIST vector big-endian format.

# GUID TLV (Optional)

Frame Word in AUX Command TLV indicates whether this TLV will arrive into KME. This TLV is made up of five fixed words. Only the fields relevant to KME are elaborated in the table below.



Figure 8 GUID TLV Format

## Word 1/2/3/4 (GUID)

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| GUID0/1/2/3 | 64 | GUID[255:0] is used in KME.  GUID[255:192] = GUID0[63:0]  GUID[191:128] = GUID1[63:0]  GUID[127:64] = GUID2[63:0]  GUID[63:0] = GUID3[63:0] |

Table 17 – GUID0/1/2/3 TLV Word 1/2/3/4

# Key TLV (Output)



### Word 0

|  |  |  |
| --- | --- | --- |
| Field | Width | Description |
| TLVType | 8 | TLV Type. This must be set to 2. |
| FrameNumber | 11 | Frame Number. Copied from AUX\_CMD Header. |
| TLVEngineID | 4 | Engine ID Number. Copied from AUX\_CMD Header. |
| TLVSeqNum | 8 | Sequence ID Number. Copied from AUX\_CMD Header. |

Table 18 - Key TLV Word 0

### Later Words

The size of the header is fixed.

|  |  |
| --- | --- |
| Field | Size |
| GUID0…GUID3 | Always 256-bits long. Copied from GUID fields of AUX\_CMD or DRBG Generated. |
| IncrIV0…IncrIV1 | Always 128-bits long. Copied from IVTWEAK fields of AUX\_CMD. |
| CiphKey0…CiphKey7 | If DEK is 256-bits, it occupies CiphKey4-7. |
| HashKey0…HashKey3 | Always 256-bits long. |
| KME Errors | Errors Generated during Key Generation at KME. Only LSB 16-bits are valid. Name in parentheses in from CAS.  **130: KIM entry for DakKeyRef is invalid** (KME\_DAK\_INV\_KIM)  **131: VF/PF Validation failed for DAK** (KME\_DAK\_PF\_VF\_VAL\_ERR)  **132: KIM entry for DekKeyRef is invalid** (KME\_DEK\_INV\_KIM)  **133: VF/PF Validation failed for DEK** (KME\_DEK\_PF\_VF\_VAL\_ERR)  **134: KME DRBG Seed Expired on Random GUID** (KME\_SEED\_EXPIRED)  135: Encrypted DEK keyblob decryption failed (ICV Tag miscompare) (KME\_DEK\_GCM\_TAG\_FAIL)  136: Encrypted DAK keyblob decryption failed (ICV Tag miscompare) (KME\_DAK\_GCM\_TAG\_FAIL)  **137: Multibit ECC Uncorrectable Memory Error** (KME\_ECC\_FAIL)  **138: Unsupported Key Type (Overloaded for multiple errors. See below)** (KME\_USUPPORTED\_KEY\_TYPE)  **139: DEK marked as KEK can only be used to decrypt keys or KDF or both** (KME\_DEK\_ILLEGAL\_KEK\_USAGE)  **140: DAK marked as KEK can only be used to decrypt keys or KDF or both** (KME\_DAK\_ILLEGAL\_KEK\_USAGE)  **141: KME DRBG Health Check Error on Random GUID Generation** (KME\_DRBG\_HEALTH\_CHECK\_FAIL)  Items in bold will zero out the Cipher Key and Hash Key in Key TLV.  DEK Tag miscompare will zero out the Cipher Key. (Code 135)  DAK Tag miscompare will zero out the Hash Key. (Code 136) |
| CRC-32 | CRC-32 over Words 1-19. Only LSB 32-bit is valid. |

Table 19 - Key TLV Later Words

The Unsupported Key Type Error is overloaded with the following errors:

1. Unencrypted Key Types 2-6 and disable\_encrypted\_keys OTP bit set to 1.
2. Key Type is NONE (Key Type 0) and key required (i.e. AUX\_CMD.crypto\_ctrl.cipher\_op != NONE or AUX\_CMD.crypto\_ctrl.auth\_op != NONE or AUX\_CMD.crypto\_ctrl.raw\_auth\_op != NONE).
3. Missing IV error (see 7.5 paragraph C)
4. Missing GUID error (see 7.5 paragraph C)

# Endian Swapping

All GUID, IV and Key Blobs are Little Endian. Crypto algorithms operate on Big Endian. Therefore, GUID, IV and Key Blobs need to be converted to Big Endian before Crypto operations.

Key Blobs are made of IV, Data and Tags and individually they need to be converted to Big Endian.

This is how a 64-bit data is converted from Little Endian to Big Endian. This applies to Data, Keys, Tags and 128-bit IVs. Assembling 128-bit data blocks from 64-bit stream: **{Word 0, Word 1}** where Word 0 comes first in time.



This is how a 96-bit IV is converted from Little Endian to Big Endian.



# Block Overview



Figure 9 - KME Top Level Block Diagram

## Internal TLV

The **Parser** in CKV Pipeline processes different flavors of Mega Commands into an Internal TLV. This is the structure that will be manipulated by downstream blocks within KME to generate a Key TLV.



Figure 10 - Internal TLV Diagram

## KOP Block Diagram



Figure 11 - Key Operations Processor Block Diagram

## Key Vault

The Key Vault (CKV) is a programmable 2Mbit memory capable of being partitioned in up to 8K unique segments holding a single 256bit key (or fewer segments holding larger combinations of keys). In addition to the CKV there is a secondary table used to provide indirection and metadata for the keys, called the Key Indirection Memory (KIM). The KIM is used to indirectly map a pointer reference from the AUX\_CMD into the CKV, as well as to provide length and other metadata information.

The Software view for these tables is:

1. The KIM appears as a table with 16,384 entries; this is double the number of actual keys that are available in the Key Vault and has been designed with the intent that multiple labels can be attached to the same key reference. Each KIM entry has a flag (*pf\_num[3]*) that can mark it as KEK (Key Encryption Key) or KDF Key only. If set, the associated CKV can only be used to decrypt keys or used as a KDF key. If **unset**, you cannot use this key to decrypt keys, but it can be used as a KDF key or to encrypt/decrypt user data.
2. The CKV appears as a flat 2Mbit memory that software can read and write over the APB interface, it will be organized from a SW point of view as a 32Kx64bit memory.

Both data structures are memory mapped into the APB interface of the KME. Secure protection is handled outside of the KME, thus the KME will not check any security state (nor is it present on the APB interface).

The data path view of these tables is as follows:

1. Hardware will read the KIM entries corresponding to DakKeyRef and DekKeyRef.
2. Hardware will then perform a series of reads to the CKV using the CKV Pointers from the KIM. The length of the reads will be based on the CKV length

The design needs to have arbiters which will implement a fixed priority access control scheme such that hardware will always be granted access to the tables and hold off the software interface.

The KIM and CKV need to have ECC protected memories.

* The KIM should be physically organized as a 16Kx38 bit memory + ECC
* The CKV should be physically organized as an 32Kx64 bit memory + ECC

### KIM Validations

KIM entries for DEK and DAK are validated according to the cases shown in Table 20, which is an excerpt from the spreadsheet included in Section 13 Validation depends on aux\_cmd.crypto\_ctrl parameters and aux\_cmd.keys\_ctrl.kdf\_mode.

KIM validations can be required for the DEK/DAK of every AUX\_CMD, regardless of the settings shown in Table 20, by setting CR\_KME\_SPARE\_CONFIG[5] to 1.

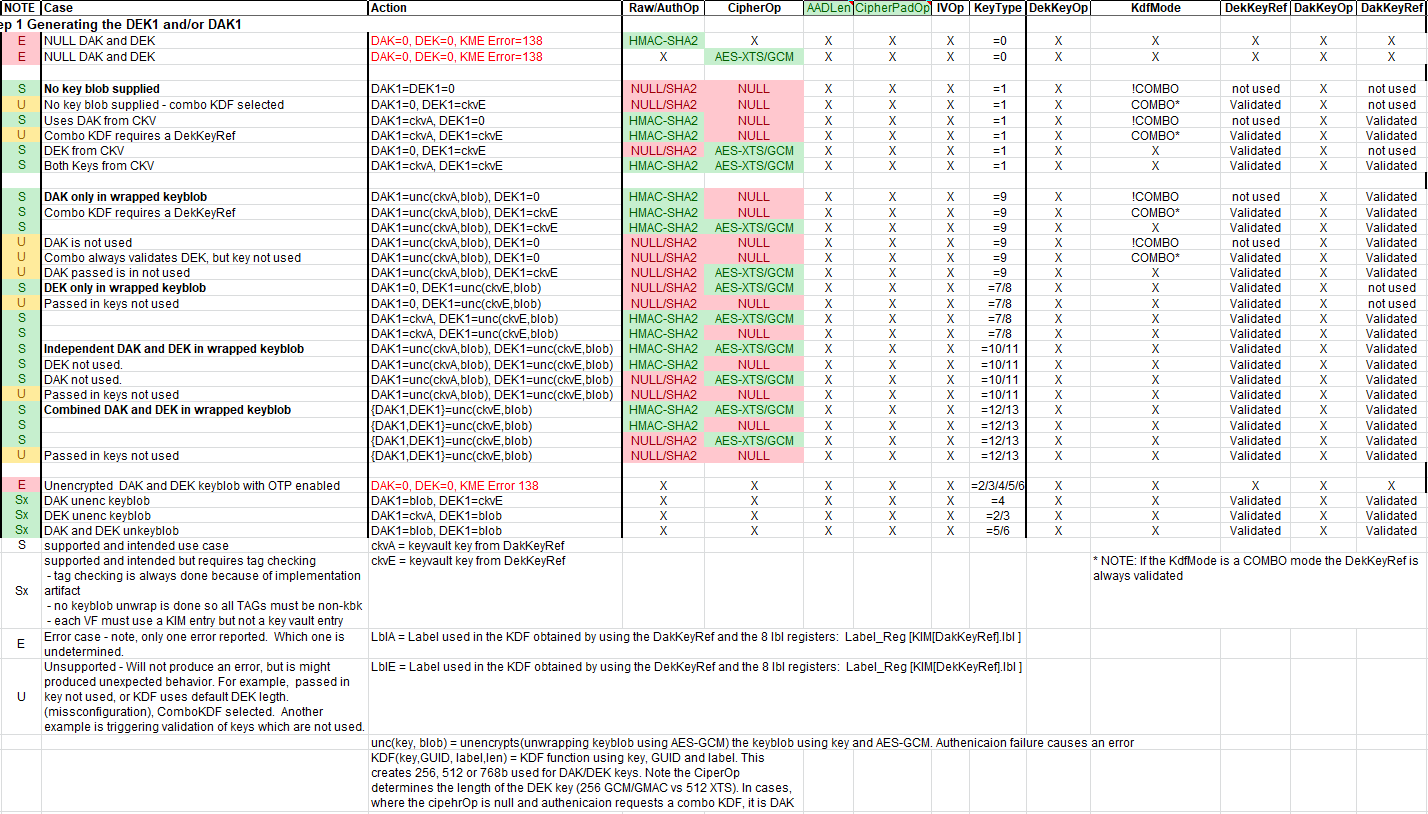


Table 20 – DEK and DAK Validation Cases

## Label Table

There are eight labels that can be programmed through the registers on APB interface. A KIM entry points to one of the eight labels.

Labels are only used during KDF operations. A Label configuration provides essential parameters to the KDF algorithm:

1. GUID Size: This can be 128-bit or 256-bit. The GUID coming on the wire or generated by DRBG is always 256-bits. This configuration will select the full or LSB 128-bit of GUID field.
2. Label Size: This can be between 0-32 bytes.
3. Label Delimiter: For NIST compliance, this needs to be set to 0x00. It is one byte long.

## Summary of Blocks

We will summarize the functionality of each block in this section.

1. Input FIFO
   1. Common Block that stores the incoming TLVs as GUID is stitched.
   2. Generates TLV start and end strobes.
   3. Extract sideband TLV Signals such as tid and user bits.
2. GUID Stitcher
   1. Stitch GUID TLV Content (if it has precedence) into the AUX Command TLV.
   2. Verifies BIP2 of all TLV Word 0.
3. CKV Pipeline – Parser
   1. Maps the incoming Mega Command to Internal TLV.
   2. If a field is not defined in the incoming Mega Command, it is set to 0.
   3. DEK/DAK KIM entry is set to 0. PF and VF number are copied from RQE1.
   4. CKV Data and KME Errors are set to 0.
   5. Set missing\_guid field in DEK KIM entry when the KME configuration requires a GUID and none exists. The configurations that cause this are shown below:

* GUID is required for KDF (i.e. (aux\_cmd.keys\_ctrl.dek\_op=KDF || aux\_cmd.keys\_ctrl.dak\_op=KDF) && (aux\_cmd.keys\_ctrl.kdf\_mode=GUID || aux\_cmd.keys\_ctrl.kdf\_mode=COMB\_GUID)
* AUX\_CMD does not have a GUID (i.e. AUX\_CMD0 || AUX\_CMD1 || no GUID TLV)

This condition is reported as an Unsupported Key Type error in KME Errors (KME\_USUPPORTED\_KEY\_TYPE (138))

* 1. Set missing\_iv field in DEK KIM entry when the KME configuration requires an IV and none exists. The configurations that cause this are shown below:
* IV is required (aux\_cmd.crypto\_ctrl.iv\_op=INCREMENT && aux\_cmd.crypto\_ctrl.cipher\_op!=NONE.)
* AUX\_CMD does not have an IV (i.e. AUX\_CMD0 || AUX\_CMD2)

This condition is reported as an Unsupported Key Type error in KME Errors (KME\_UNSUPORTED\_KEY\_TYPE (138))

* 1. For Key 12 and 13, eIV holds the IV and aTAG holds the tag.
  2. 256-bit DEK occupies [e]DEK-1 slots. [e]DEK-0 is set to 0.

1. CKV Pipeline – KIM/DRBG Reader
   1. KIM references for DEK and DAK in AUX\_KEY is always valid.
   2. If random GUID is requested, DRBG is read to replace the GUID entries in Internal TLV. If the seed is expired, errors are reported in KME Errors (KME\_SEED\_EXPIRED (134)).
   3. DEK/DAK KIM entries are read from KIM Memory.
   4. If the read KIM entry is invalid, errors are updated accordingly in KME Errors (KME\_DAK\_INV\_KIM (130) and KME\_DAK\_INV\_KIM (132)).
   5. PF/VF Validation is performed per the cases shown in Table 20, and any errors are updated accordingly in KME Errors (KME\_DAK\_PF\_VF\_VAL\_ERR (131) and KME\_DEK\_PF\_VF\_VAL\_ERR(133)). VF is only checked if VF field is valid.
2. CKV Pipeline – CKV Reader
   1. KIM Entries are used to read the CKV.
   2. If DEK is 256-bits, DEK-0 is filled with 0s and DEK-1 is filled with CKV entries.
3. CKV Pipeline - KOP Assigner
   1. Maps the Internal TLV to 3 KOPs.
   2. If Engine ID is 0-3, this Internal TLV is forwarded to CCEIP-Encrypt and CCEIP-Validate.
   3. If Engine ID is 4-7, this Internal TLV is forwarded to CCDIP-Decrypt.
4. Key Operations Processor – TLV Inspector
   1. [e]DEK-0/1 and [e]DAK is sent to Upsizer.
   2. A GCM command that has
      1. eIV and aIV for DEK and DAK GCM operations.
      2. CKV keys that will be used as GCM key.
      3. Map Key Type to GCM operation. GCM operation can be a Decrypt, Decrypt in Combo mode, send CKV key (ex. AUX\_KEY\_ONLY) or send e[DEK]-0/1 and [e]DAK (ex. DEK256\_DAK).
   3. Send Tag to be compared against. eAuthTag and aAuthTag are for DEK and DAK.
   4. Interpret KDF Command based on KDFMode field in AUX\_KEY Header.
   5. Send GUID and Label Index for KDF Stream Generation.
   6. TLV Sideband will carry Word0/GUID/IVTWEAK to form Key TLV.
5. Key Operations Processor – Upsizer
   1. Upgrades datapath from 64-bit to 128-bit for AES Engine.
6. Key Operations Processor – GCM
   1. Uses the key and IV from Command FIFO to run GCM operation.
   2. It runs GCM regardless of whether decryption is needed.
   3. Post-Decryption, Command FIFO will inform on whether to transmit decrypted GCM data, incoming data from Upsizer or CKV keys to KDF.
   4. Tag is compared if decrypted data is forwarded to KDF. The result of the comparison is sent to Key Builder via GCM Status FIFO.
7. Key Operations Processor – KDF Stream Generator
   1. Using GUID and Label, it will generate the data stream for KDF’s HMAC-SHA-256. See section 7.6)
8. Key Operations Processor – KDF
   1. It runs KDF regardless of whether KDF is needed.
   2. Post-KDF, Command FIFO will inform on whether to use the output of KDF or the incoming data from GCM to Key Builder.
9. Key Operations Processor - Key TLV Builder
   1. Using the Sideband FIFO and the DEK and DAK from KDF, it will form the Key TLV.
   2. It will also calculate CRC-32 and insert errors (if any) from GCM operation in KME Errors field (KME\_DEK\_GCM\_TAG\_FAIL(135) and KME\_DAK\_GCM\_TAG\_FAIL(136)).
10. Key TLV Compare
    1. KOP for CCEIP-Encrypt and Validate are redundant paths.
    2. If the two TLVs match, one TLV is sent via RSM (Re-assembly).
    3. If TLVs do not match, words from CCEIP-Encrypt are chosen and CRC from CCEIP-Encrypt is inverted to make it fail downstream.
11. Splitter
    1. Engine ID is used to direct Key TLV to the appropriate RSM.
12. RSM + FIFO
    1. RSM is a common block that forms a Key TLV with proper BIP2.
    2. RSM also has a large Head of the Line Blocking Output FIFO.

## Key Derivation Function (KDF)



Figure 12 KDF Diagram

KDF is made up of two parts: Stream Generator and the PRF Function HMAC-SHA-256.

Keys from GCM are coming in beats of 128-bits. A bypass path is created for these keys via Key FIFO. These keys are also upgraded to 256-bit by the Upsizer since HMAC accepts keys in that format.

**Key Filter removes upper 256-bits of DEK as the lower 256-bits of DEK are used as KDF Key.** DAK is optionally removed if we are operating in combo mode (KDFMode in AUX\_KEY).

**In combo mode, lower 256-bits of DEK are used by KDF to generate DEK and DAK.**

Using HMAC-SHA-256 Engine from Crypto block and the KDF Stream Generator, pseudorandom keys are generated.

Merger chooses between the keys from GCM and KDF output according to DekKeyOp and DakKeyOp.

### SHA-256-HMAC

HMAC does two back-to-back HASH operations to produce a result.



Figure 13 - Dual SHA-256 HMAC

The first HASH operation requires data to be padded with IPAD-KEY and SHA padding with length. If our data has M blocks, padding will result in at most M+2 blocks due to IPAD and SHA padding.

SHA-2A will process the first M blocks and output a partial result to SHA-2B. The last two unprocessed blocks of the first HASH operation will be pushed into SHA-2B FIFO.

Additionally, OPAD-KEY is pushed into SHA-2B FIFO. OPAD is used in the second HASH operation of HMAC.

Once SHA-2B receives the partial result from SHA-2A, it will process the two unprocessed blocks in SHA-2B FIFO. The resulting hash is pushed into SHA-2B FIFO thereby creating two blocks for the second and final HASH operation of HMAC. The result will be forwarded to the Frame Footer Stitcher.

In the worst case, **SHA-2B FIFO** is holding three complete SHA Blocks. Therefore, it is sized at 16x128.

#### SHA-2 Core

SHA-2 is a secure hashing standard. There are many variants of SHA-2 but the requirement is to support SHA-256.

SHA-256 is a 64 round algorithm. It operates on 512-bit block size. SHA-256 processes one block at a time. The result of each block operation is accumulated into a single register.

There are two key steps for SHA-256: Message Schedule and Rounds.



Figure 14 - SHA-2 Core

Number of SHA Round Operations performed per cycle is configurable. We chose four rounds per cycle. Please look at the Performance section of this document for an analysis.

Low number of SHA Round Operations will easily meet timing and produce a smaller area. However, latency and bandwidth will be impacted as SHA-256 is a recursive algorithm. The whole pipeline is stalled while the current block is processed.

### KDF Stream Generator

Below is the KDF operation. PRF is HMAC-SHA-256. The output of the HMAC-SHA-256 PRF is 256-bits. Therefore, if you would like to generate a 512-bit pseudorandom key, two PRF operations are required.

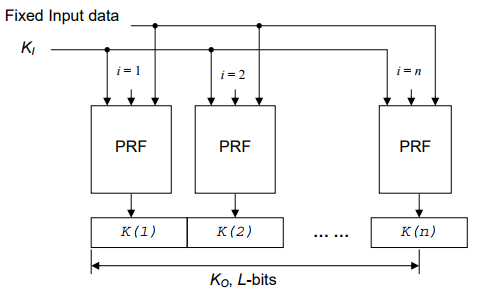
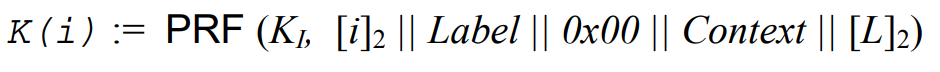


Figure 15 - NIST KDF Diagram

There are three inputs to PRF

1. Fixed Input Data.
2. Iteration variable (i).
3. PRF Key.

The output of the PRF is



where II is concatenation operation.

In the KDF specific to KME,

* Iteration variable, i, is 32-bits wide.
* Label ranges from 0-32 bytes.
* 0x00 is delimiter specified by the Label Configuration Register.
* Context is GUID which can be 128 or 256-bits wide as indicated by Label Configuration Register.
* L is the output key size in bits. It is 32-bits wide.

Suppose we are generating a 512-bit DEK, then

512-bit DEK = {K(1), K(2)}

K(1) = HMAC-SHA-256 ( 256-bit Key, 0x0000\_0001 || **Label || 0x00 || GUID || 0x000\_0200** )

K(2) = HMAC-SHA-256 ( 256-bit Key, 0x0000\_0002 || **Label || 0x00 || GUID || 0x000\_0200** )

### KDF Engine Controls

AUX\_KEY Control provides the following bits to control the KDF engine.

|  |  |  |
| --- | --- | --- |
| **AUX\_KEY Control** | | |
| KDF Mode | DEK Op | DAK Op | GUID Source | Cipher  Mode | Notes |
| GUID | KDF | KDF | Inline | AES-XTS/XEX | Run PRF twice for DEK and once for DAK. |
| KDF | KDF | Inline | !AES-XTS/XEX | Run PRF once for DEK and once for DAK. |
| NOOP | KDF | Inline | AES-XTS/XEX | Run PRF once for DAK. |
| NOOP | KDF | Inline | !AES-XTS/XEX | Run PRF once for DAK. |
| KDF | NOOP | Inline | AES-XTS/XEX | Run PRF twice for DEK. |
| KDF | NOOP | Inline | !AES-XTS/XEX | Run PRF once for DEK. |
|  | | | | | |
| RGUID | KDF | KDF | Random | AES-XTS/XEX | Run PRF twice for DEK and once for DAK. |
| KDF | KDF | Random | !AES-XTS/XEX | Run PRF once for DEK and once for DAK. |
| NOOP | KDF | Random | AES-XTS/XEX | Run PRF once for DAK. |
| NOOP | KDF | Random | !AES-XTS/XEX | Run PRF once for DAK. |
| KDF | NOOP | Random | AES-XTS/XEX | Run PRF twice for DEK. |
| KDF | NOOP | Random | !AES-XTS/XEX | Run PRF once for DEK. |
|  | | | | | |
| COMB-GUID | KDF | KDF | Inline | AES-XTS/XEX | Run PRF thrice. Use 1st and 2nd iterations as DEK and 3rd iteration as DAK. |
| KDF | KDF | Inline | !AES-XTS/XEX | Run PRF twice. Use 1st iteration as DEK and 2nd iteration as DAK. |
| NOOP | KDF | Inline | AES-XTS/XEX | Run PRF thrice. Use 3rd iteration as DAK. |
| NOOP | KDF | Inline | !AES-XTS/XEX | Run PRF twice. Use 2nd iteration as DAK. |
| KDF | NOOP | Inline | AES-XTS/XEX | Run PRF thrice. Use 1st and 2nd iterations as DEK. |
| KDF | NOOP | Inline | !AES-XTS/XEX | Run PRF twice. Use 1st iteration as DEK. |
|  |  |  |  |  |  |
| COMB-RGUID | KDF | KDF | Random | AES-XTS/XEX | Run PRF thrice. Use 1st and 2nd iterations as DEK and 3rd iteration as DAK. |
| KDF | KDF | Random | !AES-XTS/XEX | Run PRF twice. Use 1st iteration as DEK and 2nd iteration as DAK. |
| NOOP | KDF | Random | AES-XTS/XEX | Run PRF thrice. Use 3rd iteration as DAK. |
| NOOP | KDF | Random | !AES-XTS/XEX | Run PRF twice. Use 2nd iteration as DAK. |
| KDF | NOOP | Random | AES-XTS/XEX | Run PRF thrice. Use 1st and 2nd iterations as DEK. |
| KDF | NOOP | Random | !AES-XTS/XEX | Run PRF twice. Use 1st iteration as DEK. |

Table 21 KDF Engine Controls

### KDF Test Mode

In order to demonstrate that the KME KDF function is compliant with industry standard specifications, the inputs to the KDF function need to be configurable via software. The only input that is not configurable is the Key Size, which hardware controls per the number of KDF function iterations.

A KDF Test Key Size register (CR\_KME\_KDF\_TEST\_KEY\_SIZE\_CONFIG) is provided to support compliance testing. Its use is enabled by setting CR\_KME\_SPARE\_CONFIG[6] to 1.

## Random GUID Generation

KDF algorithm requires a Context. Context in KME is a 128/256-bit GUID. KME can be asked to generate this GUID through a DRBG.

Since 256-bits of GUID are reserved in Key TLV structure, DRBG always outputs a 256-bit GUID and label configuration will determine if the full GUID or the 128-bit LSB of GUID participates in KDF.

### CTR-DRBG Algorithm

This is a Deterministic Random Bit Generator algorithm based on a block cipher. AES-256 is the chosen block cipher for KME.

There are two major steps in this algorithm: Initialization and Generation. Initialization is performed by software and the Internal State of the KME is updated accordingly. KME only performs the Generate function.

The Internal State (Seed) of the KME consists of a Key, Value and a reseed limit.



Figure 16 - Internal State (Seed) of CTR-DRBG

Software should program two seeds.

In KME, GUID is 256-bits. Therefore, a Generate function will:

* Increment Value by 1.
* Feed the Key and the Incremented Value to AES-256.
* The 128-bit result is used as GUID [255:128].
* Increment Value by 1.
* The 128-bit result is used as GUID [127:0].
* Increment Value by 1 three more time (VAL0/1/2).
* Feed VAL0/1/2 to AES-256.
* The result of VAL0/1 will be the new Key of the Internal State.
* The result of VAL2 will be the new Value of the Internal State.
* Increment Reseed Counter by 1.

If the Reseed Counter reaches the Reseed Limit, KME will assert an interrupt. Software will then update the seed. In the meantime, KME will move to the next programmed seed and continue generating GUIDs. If no seeds are available, random GUIDs will be reported as errors in Key TLV (KME\_SEED\_EXPIRED(134)).

Whenever the DRBG generates a new value, it is compared to the previous value as a health check. If these 2 values match, the KME will generate a KME\_DRNG\_HEALTH\_FAIL error (141) in the Key TLV.

**Note:**

This error can be tested by setting the CR\_KME\_TOP\_KME\_KDF\_DRBG\_SEED\_0\_RESEED\_INTERVAL register to the max value of 0xffff\_ffff\_ffff.

## AES Core

AES is a block encryption standard. There are two main parts to AES Algorithm: Key Expansion and Cipher. AES operates on 128-bit block size.



Figure 17 - AES Core

Key Expansion for a 256-bit key takes 60 steps. AES Core does four steps per cycle. Once key expansion has taken place, each 128-bit block of data needs to go through 14 cipher rounds for AES-256 to produce the final encrypted block.

Number of Cipher Rounds performed per cycle is configurable. We chose one round per cycle. Please look at the Performance section of this document for an analysis.

Low number of Cipher Rounds will easily meet timing and produce a smaller area. However, latency will be impacted.

# Interrupts

The following interrupts are supported with the ability to mask.

|  |  |
| --- | --- |
| Interrupt Name | Notes |
| TLV Miscompare | Key TLVs coming out of CCEIP-Encrypt KOP and CCEIP-Validate KOP does not match. |
| GCM Tag Fail | GCM Decryption failed on DEK or DAK. |
| TXC Backpressure | An unexpected backpressure to TXC occurred. |
| ECC/Parity Error | This includes uncorrectable ECC multibit errors, BIMC interrupt, and TLV BIP2 errors. |
| DRBG Seed Expired | One of the two seeds of DRBG expired. |

Table 22 - Interrupt Table

# Performance Counters

There will be a small group of performance counters build into KME. These are provided for debug/diagnostic use.

These 50-bit counters are based on the Statistic Accumulator (SA) Block in CCEIP/CDDIP. Their register definitions and operations are identical.

|  |  |
| --- | --- |
| Counter Number | Counter Description |
| 0 | Number of Reseed requests by DRBG. |
| 1 | Number of AUX Commands requesting a Random GUID with expired seeds. |
| 2 | Number of AUX Command with Key Type 0. |
| 3 | Number of AUX Command with Key Type 1 |
| 4 | Number of AUX Command with Key Type 2. |
| 5 | Number of AUX Command with Key Type 3. |
| 6 | Number of AUX Command with Key Type 4. |
| 7 | Number of AUX Command with Key Type 5. |
| 8 | Number of AUX Command with Key Type 6. |
| 9 | Number of AUX Command with Key Type 7. |
| 10 | Number of AUX Command with Key Type 8. |
| 11 | Number of AUX Command with Key Type 9. |
| 12 | Number of AUX Command with Key Type 10. |
| 13 | Number of AUX Command with Key Type 11. |
| 14 | Number of AUX Command with Key Type 12. |
| 15 | Number of AUX Command with Key Type 13. |
| 16 | Number of Stall Clocks from CCEIP0 while Key TLV Words are ready in KME. |
| 17 | Number of Stall Clocks from CCEIP1 while Key TLV Words are ready in KME. |
| 18 | Number of Stall Clocks from CCEIP2 while Key TLV Words are ready in KME. |
| 19 | Number of Stall Clocks from CCEIP3 while Key TLV Words are ready in KME. |
| 20 | Number of Stall Clocks from CDDIP0 while Key TLV Words are ready in KME. |
| 21 | Number of Stall Clocks from CDDIP1 while Key TLV Words are ready in KME. |
| 22 | Number of Stall Clocks from CDDIP2 while Key TLV Words are ready in KME. |
| 23 | Number of Stall Clocks from CDDIP3 while Key TLV Words are ready in KME. |
| 24 | Number of AUX Commands with failed PF/VF Validation. |

# Security Considerations

The APB Interface into the KME processor is expected to be a secure master. Since APB does not provide Trust-Zone protection, the security must reside in the PRIMATE CPU complex that is part of Zipline.

* It is expected that the NIC-4xx default slave registers are properly configured to allow only secure masters to communicate with the KME.
* No other security measures are being designed at the KME level to prevent unauthorized accesses.

# Head of Line Decoupling FIFO

KME needs to accommodate the latency for the CCEIP/CDDIP. It also needs to keep up with a burst of AUX\_CMDs coming from TXC without asserting backpressure. Hence,

AUX Input FIFO is 168 entries (33 AUX\_CMDs).

Each KOP Input FIFO is 2048 entries (47 Internal Key TLVs).

Each RSM Output FIFO is 168 entries (8 Key TLVs)

# ECC

ECC is added to the following memories:

* AUX Input FIFO
* CKV
* KIM
* All 3 KOP Input FIFOs

If an ECC error is detected in any of these memories the following actions are taken:

* An ECC error is reported in the Key TLV (KME\_ECC\_FAIL (137)).
* The CR\_KME\_interrupt\_status.mbe bit is set to 1 and if enabled, an external interrupt is generated to the embedded CPU.

# Operational Spreadsheet

Attached below is a spreadsheet that describes the operation of the KME.



# References

[[2104](http://www.rfc-editor.org/rfc/pdfrfc/rfc2104.txt.pdf)] HMAC: Keyed-Hashing for Message Authentication

[[FIPS-180-2](http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf)] Secure Hash Standard

[[FIPS-197](http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf)] Advanced Encryption Standard (AES)

[[GCM](http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf)] The Galois/Counter Mode of Operation (GCM)

[[DRBG](http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf)] NIST SP800-90A Recommendation for Random Number Generation (Section 10.2)

[[KDF](http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf)] NIST SP800-108 Recommendation for Key Derivation Using Pseudorandom Functions (Section 5.1 - KDF in Counter Mode)