Cryptographic Module for Intel Corporation® Platforms' Security Engine Chipset

Module Version 3.1

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.5 Last update: 2022-09-28

Contents

Copyrights and Trademarks

1. Introduction

This document is the non-proprietary FIPS 140-2 Security Policy for module version 3.1 of the Cryptographic Module for Intel® Platforms' Security Engine Chipset. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS PUB 140-2 (Federal Information Processing Standards Publication 140-2) for a Security Level 1 Firmware-Hybrid module.

2. Cryptographic Module Specification

2.1. Module Overview

The Cryptographic Module for Intel® Platforms' Security Engine Chipset (hereafter referred to as "the module") is classified as a multiple-chip standalone firmware-hybrid module for FIPS 140-2 purpose. The Security Engine Chipset consists of both hardware and firmware. The hardware portion is the Converged Security Engine (CSE) and the firmware portion is the crypto driver process. The two portions form the logical cryptographic boundary and they combine to perform cryptographic functions within the Intel® for applications executing on CSE. The module is embodied in the Intel Platform Controller Hub (PCH) chipset shown in Figure 1 below.

The components of the hybrid cryptographic module are specified in the following table:

Component	Type	Version Number	Description
Crypto Driver	Firmware	3.1	It is a firmware process running in an internal customized proprietary O/S to communicate with the hardware components of the module in the Intel PCH chipset. It includes implementations of various cryptographic algorithms such as SHA-224, SHA-384, SHA-512, HMAC, RSA, ECDSA, DRBG, and EC Diffie-Hellman implementations.
Converged Security Engine (CSE)	Hardware	2.0	CSE is a 32-bit microprocessor that includes hardware implementations of AES, SHA-1 and SHA-256 security engines, and big number arithmetic support for implementing asymmetric algorithms in firmware. It is embedded within the Intel PCH chipset.
fips_hmac	File	N/A	It is a file located in the nonvolatile RAM file system of the internal customized proprietary O/S to contain the HMAC-SHA-256 hash value for integrity check of the module.
run_bist	File	N/A	It is a file located in the nonvolatile on-chip RAM that indicates the status of self-tests, per IG 9.11.

Table 1 - Cryptographic Module Components

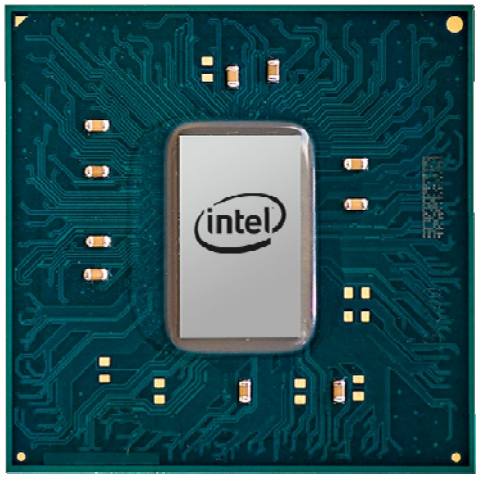

Figure 1 – Intel PCH Chipset

The module has been tested on the following multichip standalone platform:

Platform	Processor	Operating System
Intel Comet Point PCH (Chipset with CSE device firmware version 14.0.30.1115)	Lakemont 3.6 (Embedded CPU with x86 instruction set (IA-32) executing the CSE device firmware with module version 3.1)	Embedded customized proprietary O/S running on firmware version 14.0.30.1115 (Dedicated to support the functionality of the CSE)

Table 2 - Tested Platforms

The table below shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard:

FIPS 140-2 Section		Security Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services and Authentication	1
4	Finite State Model	1
5	Physical Security	1
6	Operational Environment	N/A
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	1
Overall Level		1

Table 3 - Security Levels

2.2. Block Diagrams

The physical cryptographic boundary of the module is the Intel PCH chipset, marked by the outer boundary in Figure 2. Consequently, the embodiment of the module is a multi-chip standalone cryptographic module.

The module provides cryptographic services to applications through an application program interface (API). The cryptographic logical boundary consists of the firmware component (i.e., CSE Crypto Driver), the file for integrity check, the hardware for checking if the algorithmic self-tests need to be run, and the hardware components (i.e., CSE). The logical block diagram below shows the module, its interfaces with the operational environment and the delimitation of its logical boundary which are colored in **BLUE**:

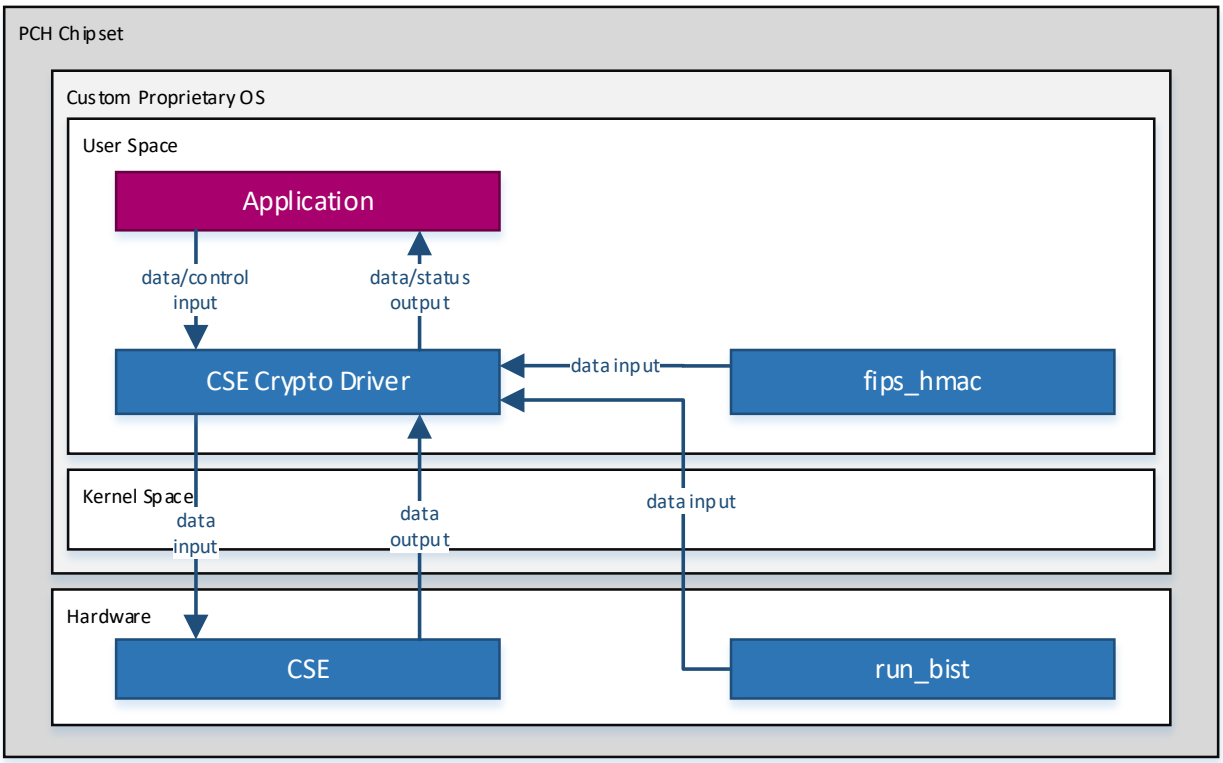

Figure 2 - Logical Block Diagram

2.3. Modes of Operation

The module supports two modes of operation:

- In "FIPS mode" (the FIPS Approved mode of operation) only approved or allowed security functions with sufficient security strength can be used.
- In "non-FIPS mode" (the non-Approved mode of operation) only non-approved security functions can be used.

When the module is powered-up and FIPS mode is enabled, the kernel of the operating environment obtains the HMAC value of the module for integrity check from the fips_hmac file and sends the HMAC value to the Crypto Driver. The Crypto Driver will read the run_bist file from battery-backed non-volatile storage. This file tracks if Crypto Driver has previously passed the full set of power-up self-tests. Pursuant with IG 9.11 requirements, the full set of power-up self-tests will not be run if the run_bist file indicates that the full set tests had previously passed. Instead, the tests executed at power up are the integrity self-test and to bootstrap the entropy self-test.

If the module fails any power-up self-test, it will enter Error state and trigger CSE reset to rerun the module and perform the power-up self-tests again. The module will store the passing test status into the run_bist file if needed. Once the

module is operational, the mode of operation is implicitly assumed depending on the security function invoked, the security strength of the cryptographic keys, and by policy of the consuming application.

Critical security parameters used or stored in FIPS mode are not used in non-FIPS mode, and vice versa.

The module supports the following Approved cryptographic algorithms:

Algorithms	Components	Standards	CAVS Cert
AES encryption and decryption with 128-bit and 256-bit key sizes and the following mode: • ECB • CBC • CMAC • Counter (CTR) • GCM (Decryption only) • GMAC (Verify only) • OFB • CFB128	CSE Crypto Driver	FIPS 197, SP 800-38A, SP 800-38B, SP 800-38D	# C1463
CKG • Section 6.1 Key Pairs for Digital Signature Schemes using unmodified DRBG output • Section 6.2 Asymmetric key establishment key generation using unmodified DRBG output • Section 7.3 Symmetric Keys Generated Using Key-Agreement Schemes using unmodified DRBG output	CSE Crypto Driver	SP 800-133	Vendor Affirmed
CVL: RSADP with a 2048-bit modulus	CSE Crypto Driver	SP 800-56B	#C1463
CVL: RSASP with a 2048-bit modulus	CSE Crypto Driver	FIPS 186-4	#C1463
CVL: Section 5.7.1.2 ECC cofactor Diffie-Hellman (ECC CDH) primitive with the following curves (EC Diffie-Hellman Key Agreement primitive, P-256, P-384, provides 128 or 192 bits of encryption strength): • NIST P256 • NIST P384	CSE Crypto Driver	SP 800-56Ar1	#C1463
DRBG based on counter mode AES-256: • With Derivation Function and without Prediction Resistance	CSE Crypto Driver	SP 800-90A	#C1463
ECDSA EC Key Pair Generation and Public Key Verification with the following curves: • NIST P256 • NIST P384	CSE Crypto Driver	FIPS 186-4	#C1463
ECDSA Signature Generation with the following curves: • NIST P256 • NIST P384 This is supported in conjunction with the following hash algorithms: • SHA-224 • SHA-256 • SHA-384 • SHA-512	CSE Crypto Driver	FIPS 186-4	#C1463

ECDSA Signature Verification with the following curves: • NIST P256 • NIST P384 This is supported in conjunction with the following hash algorithms: • SHA-1 (legacy use only) • SHA-224 • SHA-256 • SHA-384 • SHA-512	CSE Crypto Driver	FIPS 186-4	#C1463
HMAC with: • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512	CSE Crypto Driver	FIPS 186-4	#C1463
KAS: Section 6.2.2.2 (Cofactor) One-Pass Diffie-Hellman, C(1e, 1s, ECC CDH) Scheme using the concatenation KDF with the following curves (EC Diffie-Hellman Key Agreement with concatenation KDF, P-256, P-384, provides 128 or 192 bits of encryption strength): • NIST P256 • NIST P384 KAS: Section 6.1.1.2 (Cofactor) Full Unified Model, C (2e, 2s, ECC CDH) Scheme with the following curves (EC Full Unified Key Agreement primitive, P-256, P-384, provides 128 or 192 bits of encryption strength): • NIST P256 • NIST P384	CSE Crypto Driver	FIPS 198-1	#C1463
KBKDF: Counter mode Key Derivation Function using: • HMAC SHA-1 • HMAC SHA-224 • HMAC SHA-256 • HMAC SHA-384 • HMAC SHA-512	CSE Crypto Driver	SP 800-108	#C1463
KTS: RSA-based Key Transport with OAEP and 2048, 3072, and 4096 bits modulus size (CVL Cert. #C1463; RSA key wrapping using encryption and decryption primitives, 2048, 3072, and 4096 bits, provides between 112 and 150 bits of strength)	CSE Crypto Driver	SP 800-56B	Vendor Affirmed
KTS: Support for the following key wrapping/unwrapping algorithms: • KW key wrapping/unwrapping algorithm with support for 128-bit and 256-bit KEKs (AES Key Wrap, key establishment methodology provides 128 or 256 bits of encryption strength) • Key unwrap using AES-GCM as an authenticated symmetric decryption algorithm with support for 128-bit and 256-bit wrapping keys. (GCM, Key Unwrapping, key establishment methodology provides 128 or 256 bits of encryption strength.) • Key wrap/unwrap using the combination of an approved authentication algorithm (RSA PKCS, RSA PSS, ECDSA, HMAC, or CMAC) and an approved symmetric encryption algorithm (AES).	CSE Crypto Driver	SP 800-38F	#C1463

(AES, Key Wrapping (with CMAC, HMAC, RSA, and ECDSA authentication, key establishment methodologies provides 128 or 256 bits of encryption strength)			
RSA Key Generation with 2048-bit modulus sizes.	CSE Crypto Driver	SP 800-186-4 Appendix B.3.3	#C1463
RSA Signature Generation based on PKCS#1 v1.5 with 2048, 3072, and 4096† modulus bit sizes. Hash functions available are: • SHA-224 • SHA-256 • SHA-384 • SHA-512 RSA Signature Generation based on PSS with 2048, 3072, and 4096† modulus bit sizes. Hash functions available are: • SHA-224 • SHA-256 • SHA-384 • SHA-512 † RSA Signature Generation with 4096-bit modulus is per FIPS 186-2.	CSE Crypto Driver	FIPS 186-4 Appendix B.3.3	#C1463
RSA Signature Verification based on PKCS#1 v1.5 with 2048, 3072, and 4096 modulus bit sizes. For legacy use, RSA with 1024-bit modulus sizes are supported. Hash functions available are: • SHA-1 (legacy use only) • SHA-224 • SHA-256 • SHA-384 • SHA-512 RSA Signature Verification based on PSS with 2048, 3072, and 4096 modulus bit sizes. For legacy use, RSA with 1024-bit modulus sizes are supported. Hash functions available are: • SHA-1 (legacy use only) • SHA-224 • SHA-256 • SHA-384 • SHA-512	CSE Crypto Driver	FIPS 186-4	#C1463
SHA: • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512	CSE Crypto Driver	FIPS 186-4	#C1463

Table 4 – Validated Cryptographic Algorithms

Note: The module only uses the general purpose AES engine from the CSE for AES cryptographic operations (i.e., encryption and decryption). Any other AES engine from the hardware component of the module is considered as dead path since it is not callable via the interface of crypto driver.

The module supports three Non-Approved but Allowed elliptic curves. Both KAS (Cert. #C1463) and ECDSA (Cert. #C1463) signature generation and verification are using the following Non-Approved but Allowed elliptic curves, per IG A.2:

- Barreto Naehrig Curve. 256-bit long curve offers 128 bits of encryption strength.
- SECG P256k1 curve. 256-bit long curve offers 128 bits of encryption strength.
- Brainpool384 curve. 384-bit long curve offers 192 bits of encryption strength.

The above approved algorithms are only approved in the modes, sizes, etc. as specified above. As a level 1 FIPS module it is the user's responsibility to ensure adherence to the requirements for each algorithm.

The module supports the following Non-Approved but Allowed cryptographic algorithms:

Non-Approved but Allowed Algorithms
NDRBG: The module uses a HW NDRNG as its major source for entropy. This entropy source is used during the seeding / reseeding process of the module's internally managed DRBG. The NDRNG is used to generate 576 bits of data (per IG 7.14 1a) to seed the DRBG. There is an option where the seed is passed into the module via API input parameters for DRBG with external entropy. When entropy is externally loaded, no assurance of the minimum strength of generated keys.

Table 5 – Non-Approved but Allowed Algorithms

The module implements the following non-Approved algorithms:

Algorithms
AES • GCM (Encryption only) • GMAC (Generate only)
AES-GCM Key Wrapping
MD5 including its use in the following algorithms: • Hashing • HMAC • ECDSA signing, verification • SP 800-108 KDF • SP 800-56A KDF • RSA signing, verification
RSA encryption / decryption key transport with modulus byte sizes in the range [256, 512]. Only byte lengths aligned to 8 bytes are supported. RSA key generation, digital signature verification and generation, key wrapping with non-compliant modulus sizes (32 bytes to 248 bytes)
SHA1 hash algorithm usages: • RSA signature generation • ECDSA signature generation
Trusted Computing Group (TCG) EC algorithms: • EC Schnoor • EC commit computation • EC DAA

Table 6 – Non-Approved Cryptographic Algorithms

Regarding the services available in FIPS mode of operation and non-FIPS mode of operation, please refer to Table 8 - Services in FIPS mode of operation and Table 9 - Services in non-FIPS mode of operation in 4.2 Services.

3. Cryptographic Module Ports and Interfaces

As the module is a firmware-hybrid, the data enters and exits the module via the logical interface through firmware.

The logical interfaces are the application program interface (API) through which applications request services from the firmware (i.e., CSE Crypto Driver). The hardware interfaces are the interface for data in and out of the hardware components of the module. The following table summarizes the four logical interfaces, where DMA denotes Direct Memory Access and IOSF denotes Intel On-Chip System Fabric:

FIPS 140-2 Interface	Logical Interface	Hardware Interface
Data Input	API input parameters pointing to data stored in RAM, fips_hmac file.	DMA
Data Output	API output parameters pointing to RAM locations for storing output data.	DMA
Control Input	API function calls.	IOSF
Status Output	API return codes.	IOSF
Power Port		PCH power pin

Table 7 - Ports and Interfaces

4. Roles, Services and Authentication

4.1. Roles

The module supports the following roles:

- **User role**: performs all services (in both FIPS mode and non-FIPS mode of operation), except module initialization.
- **Crypto Officer role**: performs module initialization and enables FIPS mode.

The User and Crypto Officer roles are implicitly assumed by the entity accessing the module services.

4.2. Services

The module provides services to users that assume one of the available roles. All services are described in detail in the user documentation.

The following table lists the Approved services and the non-Approved but allowed services in FIPS mode of operation, the roles that can request the service, the Critical Security Parameters involved and how they are accessed. The numbers in parentheses maps the key/CSP entry to the corresponding entry in Table 10.

Service	Role	Key/CSP	Access
Symmetric encryption and decryption	User	AES Keys (#1): AES with 128-bit and 256-bit keys	Read
RSA key generation	User	RSA Key Pair (#3): RSA public-private keys with modulus size 2048.	Create
RSA digital signature generation based on PKCS#1 v1.5 and PSS scheme	User	RSA public-private keys (#3) with modulus size 2048, 3072, and 4096 bits.	Read
RSA digital signature verification based on PKCS#1 v1.5 and PSS scheme	User	RSA public-private keys (#3) with modulus size 1024, 2048, 3072, and 4096 bits.	Read
RSA-based Key Transport using RSAOAEP (SP 800-56B Section 9)	User	RSA public-private keys (#3) with modulus size 2048, 3072, and 4096 bits.	Read
RSA-based Key Wrapping using RSA Encryption and Decryption Primitives (SP 800-56B Section 7.1)			
RSA key validation	User	RSA prime numbers and private key (#3)	Read
ECDSA key generation	User	ECDSA Key Pair (#4): ECDSA public-private keys with P256 and P384 curve	Create
ECDSA signature generation			Read
ECDSA public key verification	User	ECDSA public keys (#4) with P256 and P384 curve	Read
ECDSA signature verification			Read
Message digest generation	User	None	None
MAC generation and verification	User	HMAC Keys (#2): At least 112 bits HMAC key	Read
Random Number Generation	User	Entropy Input String for DRBG Seed (#6): DRBG seed (consisting of Entropy Input String); DRBG internal V and Key (#7)	Read, Update
EC Diffie-Hellman Key Agreement Primitive	User	ECC public-private keys with P256 curve, P384 curve, shared secret (#12); Shared Secret (#5)	Read, Create
EC Diffie-Hellman Full Unified Scheme			

EC Diffie-Hellman One-Pass with concatenation KDF Scheme	User	Concatenation KDF Shared Secret (#9): ECDSA public-private keys with P256 curve, P384 curve, derived keying material	Read, Create
SP 800-108 counter mode HMAC KDF	User	Counter Mode HMAC KDF Key (#8): HMAC key	Read
SP 800-56A concatenation KDF	User	Shared secret value (#9)	Read
AES-KW wrapping/unwrapping algorithm	User	AES 128 bit and 256-bit keys (#10)	Read
Show status	User	None	None
Self-Tests	User	According to IG 7.4, the keys only used to perform Power-Up Tests are not considered CSPs.	None
Zeroization	User	All CSPs	Zeroize
Module Initialization	Crypto Officer	None	None

Table 8 - Services in FIPS Mode of Operation

The following table lists the services only available in non-FIPS mode of operation.

Service	Role	Key/CSP	Access
Message digests using MD5	User	None	Create
MAC generation using MD5	User	HMAC key (#2)	Read
MAC generation using less than 112-bits HMAC key	User	Less than 112-bits HMAC key (#2)	Read
Symmetric encryption / decryption using RC4	User	RSA public/private keys of any size (#3)	Read
RSA encryption/decryption with no padding algorithm.	User	RSA public/private keys of any size (#3)	Read
RSA key generation with 1024 bits modulus size	User	RSA public-private keys with modulus size 1024 bits (#3)	Create
RSA signature generation, key wrapping with any modulus size rather than 2048, 3072 and 4096 bits	User	RSA public-private keys with any modulus size rather than 2048, 3072 and 4096 bits (#3)	Read
RSA signature generation with MD5 or SHA-1 for any modulus size	User	RSA public-private keys with any modulus size (#3)	Read
Trusted Computing Group EC Schnoor signature generation algorithm	User	ECDSA Key Pair (#4): ECDSA public/private keys	Read
Trusted Computing Group EC Schnoor signature verification algorithm			
Trusted Computing Group ECDAA signature generation algorithm			
Trusted Computing Group EC commit computation algorithm			

Table 9 - Services in Non-FIPS Node of Operation

4.3. Operator Authentication

The module does not implement authentication. The role is implicitly assumed based on the service requested.

5. Physical Security

The Cryptographic Module for Intel® Platforms' Security Engine Chipset is a firmware hybrid module that operates on a multi-chip standalone platform which conforms to the Level 1 requirements for physical security. The hardware portion of the cryptographic module is a production grade component. The cryptographic module must be used in a commercial off the shelf (COTS) computer device. The computer device shall be comprised of production grade components with standard passivation (a sealing coat applied over the chip circuitry to protect it against environmental and other physical damage) and a production grade enclosure that completely surrounds the cryptographic module.

6. Operational Environment

6.1. Applicability

The module operates in a limited operational environment per FIPS 140-2 level 1 specifications. The module runs on an internal customized proprietary O/S within the Intel PCH chipset.

6.2. Policy

The operating system is restricted to a single operator; concurrent operators are explicitly excluded.

The application that requests cryptographic services is the single user of the module.

7. Cryptographic Key Management

The following table summarizes the Keys and Critical Security Parameters (CSPs) that are used by the cryptographic services implemented in the module:

#	Name	Generation / Entry	Storage	Zeroization
1	AES Keys	The key is passed into the module via API input parameters. The key can also be loaded from the Secure Key Storage (SKS) directly to the register.	Stored as plaintext in the RAM or SKS.	Called memset_secure() to replace zero in the memory.
2	HMAC Keys	The key is passed into the module via API input parameters. The key can also be loaded from the SKS directly to the register.	Stored as plaintext in the RAM or SKS.	Called memset_secure() to replace zero in the memory.
3	RSA Key Pair	The prime number is generated using SP 800-90A DRBG. The RSA public-private keys are generated using FIPS 186-4 RSA Key Generation method. The key pair is passed into the module via API input parameters.	Stored as plaintext in the RAM.	Called memset_secure() to replace zero in the memory.
4	ECDSA Key Pair	The ECDSA public-private keys are generated using FIPS 186-4 ECDSA Key Generation method and the random value used in key generation is generated using SP 800-90A DRBG. The key pair is passed into the module via API input parameters.	Stored as plaintext in the RAM.	Called memset_secure() to replace zero in the memory.
5	Shared Secret	The shared secret is generated in the EC Diffie-Hellman key agreement function.	Stored as plaintext in the RAM.	Called memset_secure() to replace zero in the memory.
6	Entropy Input String for DRBG seed	Obtained from hardware DRBG outside of the module's logical boundary for DRBG with internal entropy. Passed into the module via API input parameters for DRBG with external entropy. When entropy is externally loaded, no assurance of the minimum strength of generated keys	Stored as plaintext in the RAM.	Zeroized during the power cycle of the module. External entropy DRBG can also be zeroized upon request of caller API.
7	DRBG internal V and Key	Generated internally in the DRBG.	Stored as plaintext in the RAM.	Zeroized during the power cycle of the module.
8	Counter mode HMAC KDF Key	HMAC key comes into the module via API input.	Stored as plaintext in the RAM.	Called memset_secure() to replace zero in the memory

9	Concatenation KDF Shared Secret	Calculated inside module during One-Pass ECDH operation.	Stored as plaintext in the RAM.	Called memset_secure() to replace zero in the memory
10	AES-KW KEK	The key is passed into the module via API input parameters.	Stored as plaintext in the SKS	Zeroized by replacing new values.
11	RC4 Keys	The key is passed into the module via API input parameters.	Stored as plaintext in the RAM.	Called memset_secure() to replace zero in the memory.
12	ECC Key Pair	The ECC public-private keys are generated using NIST SP 800-56A Section 5.6.1.2.2 ECC Key Generation method and the random value used in key generation is generated using SP 800-90A DRBG. The key pair can also be passed into the module via API input parameters.	Stored as plaintext in the RAM.	Called memset_secure() to replace zero in the memory.

Table 10 - Life Cycle of Keys and Critical Security Parameters (CSP)

The following sections describe how CSPs, in particular cryptographic keys, are managed during its life cycle.

7.1. Random Number Generation

The module employs three instantiations of the Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of asymmetric keys, and for providing a Random Number Generation service to calling applications. The module implements each instantiation as a CTR_DRBG with AES-256 with the derivation function and without prediction resistance. The CTR_DRBG is implemented in the firmware (i.e., CSE Crypto Driver) and provides between 1 and 65536 bytes of output data per each request.

The first instantiation is used when the module is operating in FIPS mode as the default source for randomness for all cryptographic operations. The module uses the raw entropy source of a hardware DRNG as the entropy source for seeding the CTR_DRBG. The hardware DRNG is implemented outside of the module's logical boundary within the Intel PCH chipset physical boundary. The module collects 72 bytes of data from the hardware DRNG for generating the initial seed during initialization of the CTR_DRBG, and reseeding which occurs less than 2^{28} times of DRBG service request.

The second instantiation is similar to the first instantiation in that it also uses the HW DRNG for seeding and reseeding. The difference is that this DRBG will only be instantiated and de-instantiated upon request from within an API.

The third instantiation does not use the HW DRNG for seeding and reseeding. Rather the entropy is provided as input to the module from an API. When entropy is externally loaded, no assurance of the minimum strength of generated keys.

The module performs Continuous Random Number Generator Test (CRNG) on the output of the hardware DRNG to ensure that the hardware DRNG is not degraded. The CRNG evaluates if the current 72 bytes of entropy are binary identical to the previous 72 bytes of entropy.

7.2. Key Generation

For generating RSA and ECDSA keys the module implements asymmetric key generation services compliant with [SP800-133] and [SP800-90A].

The module does not offer a dedicated service for generating keys for symmetric algorithms. However, the module offers a DRBG compliant to [SP800-90A] to allow a caller to obtain random numbers which can be used as key material for symmetric algorithms or HMAC.

7.3. Key Establishment/Key Derivation

In approved mode the module supports the [SP800-56A] EC Diffie-Hellman Key Agreement primitive, EC Full Unified Key Agreement primitive, and EC Diffie-Hellman Key Agreement with concatenation KDF for the P-256 and P-384 curves. Additionally, in approved mode the module supports the non-approved Barreto Naehrig, SECG P256k1, and Brainpool384 curves which are allowed in approved mode. The caller may use the module's DRBG to generate EC Diffie-Hellman private keys.

The module also supports [SP800-56B] RSA Key Transport using OAEP with the modulus size of 2048, 3072 and 4096-bits in FIPS mode Other modulus sizes are only available in non-FIPS mode.

CAVEAT:

- EC Diffie-Hellman key establishment methodology provides 128 bits of encryption strength for 256-bit curves and 192 bits of encryption strength for 384-bit curves.
- RSA key wrapping provides between 112 and 150 bits of encryption strength.

7.4. Key Entry / Output

The module does not support manual key entry or intermediate key generation key output. In addition, the module does not produce key output in plaintext format outside its physical boundary.

7.5. Key / CSP Storage

The symmetric keys and HMAC keys are provided to the module via API input parameters and are destroyed by the module before they are released in the memory.

Asymmetric public and private keys are provided to the module via API input parameters and are destroyed by the module before they are released in the memory.

The Secure Key Storage (SKS) is a hardware block of CSE for storing 128-bit and 256-bit keys for AES or HMAC. Keys in SKS are invisible to the Crypto Driver firmware or calling application. The keys in SKS are directly loaded into CSE's AES or HMAC engine. The calling application may request the Crypto Driver to store caller-provided keys in SKS and use the keys for AES or HMAC operations later.

The HMAC key used for integrity test is stored in the module's source code. The HMAC value used for integrity test is stored in the fips_hmac file and relies on the operating system for protection.

7.6. Key / CSP Zeroization

The memory occupied by keys is stored in static arrays or allocated by regular memory allocation operating system calls. The firmware of the module (i.e., CSE Crypto Driver) calls the memset_secure() functions to overwrite the memory occupied by keys with "zeros" before deallocating the memory with the regular memory deallocation operating system call.

At the hardware level, two Memory-Map I/O (called "MMIO" in short) registers are accessible from CSE Crypto Driver to store the keys temporarily: HCU_KEY and AES_KEY. These two registers are write-only (i.e., user cannot read the keys from the registers) and they are mapped to the CSE Crypto Driver only (i.e., no other process is able to access these registers); therefore, the keys are protected by the hardware architecture before the key zeroization occurs. The keys are zeroized during the power-cycle of the module or replacing by other new keys. All other keys in the hardware components for the cryptographic operations are provided via the SKS which is wired hardware-internally to the cipher engines.

8. Self-Tests

The module performs power-up self-tests and conditional tests to ensure the correctness of the cryptographic algorithm implementations within the module boundary. If any self-test fails, the module enters Error state by calling the CSE reset to restart the module and rerun the power-up self-test to recover from Error state. No data output and cryptographic operation are allowed in Error state.

Once the FIPS mode has been enabled on the module the system will not boot unless the power-up tests have passed. IG9.11 requires that the integrity self-test must pass every boot and that the other power-up tests are known to have passed on a prior boot cycle. This is a FIPS enabled system being fully booted is evidence that all required power-up self-tests have passed.

8.1. Power-Up Tests

The module performs power-up tests automatically when the module is loaded into memory; power-up tests ensure that the module is not corrupted and that the cryptographic algorithms work as expected.

While the module is executing the power-up tests, services are not available, and input and output are inhibited. The module does not return control to the calling application until the power-up tests are completed.

Once the power-up tests are completed successfully, the module will be operational.

8.1.1. Integrity Tests

The integrity of the module is verified by comparing an HMAC-SHA-256 value calculated at run time with the HMAC value stored in the fips_hmac file that was computed at build time. If the HMAC values do not match, the test fails, and the module enters the Error state.

8.1.2. Cryptographic algorithm tests

The module performs self-tests on all FIPS-Approved cryptographic algorithms supported in the approved mode of operation, using the known answer tests (KAT) and pair-wise consistency test (PCT), shown in the following table:

Algorithm	Power-Up Tests
AES	• KAT AES ECB, encrypt with 128-bit key • KAT AES ECB, decrypt with 128-bit key • KAT AES GCM, encrypt with 128-bit key
SHS	• KAT SHA-1 is covered in the KAT for HMAC-SHA-1 as allowed with IG 9.1 • KAT SHA-224 is not required per IG 9.4 • KAT SHA-256 is covered in the Integrity Test which is allowed with IG 9.3 • KAT SHA-384 is not required per IG 9.4 • KAT SHA-512 is covered in the KAT for HMAC-SHA-512 as allowed with IG 9.1
HMAC	• KAT HMAC-SHA-1 and HMAC-SHA-512
ECDSA	• PCT ECDSA (NIST P-256) signature generation • PCT ECDSA (NIST P-256) signature verification
RSA	• KAT RSA 2048-bit key PKCS#1 v1.5 with SHA-256 signature generation • KAT RSA 2048-bit key PKCS#1 v1.5 with SHA-256 signature verification • 800-56B KAT with an RSA 2048-bit key using OAEP encrypt • 800-56B KAT with an RSA 2048-bit key using OAEP decrypt

DRBG	• KAT CTR_DRBG (Instantiate, Generate, and Reseed Health Test Functions)
KBKDF	• KAT KBKDF with SHA-256 HMAC and a 256-bit key
Algorithm	**Power-Up Tests**
ECC Diffie-Hellman	Primitive "Z" Computation KAT: • KAT for ECC Cofactor Diffie-Hellman primitive (NIST P-256)

Table 11 - Self-Tests

For the KAT, the module calculates the result and compares it with the known value. If the answer does not match the known answer, the KAT is failed, and the module enters the Error state.

For the PCT, if the signature generation or verification fails, the module enters the Error state.

8.2. On-Demand Self-Tests

The on-demand self-tests is invoked by the user running the self-test command.

The power-up self-tests can be run on demand by the crypto officer disabling the module from within BIOS, rebooting the platform, re-enabling the module from within BIOS, and rebooting the platform again. At this point the module will run the full set of power-up tests again. During the execution of the power-up self-tests, services are not available, and no data output or input is possible.

8.3. Conditional Tests

The module performs conditional tests on the cryptographic algorithms, using the pair-wise consistency test (PCT) and the Continuous Random Number Generator Test (CRNGT), shown in the following table:

Algorithm	Test
ECDSA key generation	• PCT signature generation and verification
RSA key generation	• PCT signature generation and verification
DRBG (Not Implemented)	• CRNGT is not required per IG 9.8
NDRNG	• CRNGT

Table 12 - Conditional Tests

9. Guidance

9.1. Operator's Guidance

The following security guidance for User role is described below:

- When the module switches between FIPS and non-FIPS mode or vice versa the following must be considered by the user:
    - The DRBG engine must be reseeded.
    - The CSPs and keys shall not be shared between the modes.

- The operator of the module can call the crypto_drv_fips_mode_status() API to check if the module is an FIPS 140-2 validated module. If the API call returns 1, the module is an FIPS 140-2 validated module; it returns 0 if the module is not an FIPS 140-2 validated module.
- To enable/disable FIPS mode, the user will enter the BIOS and select the relevant BIOS menu setting (“PCH-FW Configuration” to “FIPS Configuration”) as defined by the platform manufacturer. The user will then enable or disable FIPS mode and save the changes. Afterwards, the system will be reset in order to transition to/from FIPS mode after the setting is configured.
- Once the operator enables FIPS in the configuration settings of the operating environment, the module is initialized as an FIPS 140-2 validated module.
- One RSA key pair shall be used for one cryptographic purpose, such as digital signature and key transport. A new RSA key pair is required for different cryptographic purpose.

9.2. Delivery Procedure

The firmware component of the module is distributed as part of the CSE Device Driver firmware. Firmware is released on VIP site https://platformsw.intel.com. Only Original Equipment Manufacturers (OEM) with signed Intel agreements are able to download this firmware.

The hardware component of the module is contained within the Intel Sunrise Point Platform Controller Hub (PCH). The Intel Comet Point PCH is a tightly coupled component of Intel® Core™ and Intel® Core™ platforms. The Intel Comet Point PCH can be bundled with CPU as a kit, or outside the CPU packages as a discreet component mounted on the Printed Circuit Board (PCB). Intel requires their Original Equipment Manufacturer (OEM) partners that create, market, and sell brand systems to meet the brand validation requirements and testing to ensure the systems have been designed and constructed with the proper components including CPU and Intel Comet Point PCH. Intel‘s brand validation tool would detect any mismatch of CPU and PCH for any system being designed.

Intel manages and implements security best practices throughout every step of their supply chain and works closely with their partners (i.e., Original Design Manufacturer and Original Equipment Manufacturer) to ensure that they meet Intel’s requirements for secure supply chain processes as specified in partner contract agreements. Therefore, end customers can be assured that any system with a badge had been designed and tested to conform to Intel’s requirements and systems will always have the Intel Comet Point PCH that contains the module.

10. Mitigation of Other Attacks

The module provides a mechanism to guard against an RSA timing attack.

The CSE's big number arithmetic is able to perform the modular exponentiation operations in constant time. This means, the time taken is only dependent on the size of operands and not dependent on the value of the operands. During modular exponentiation, an extra mathematical step is needed when the processed bit of the key is 1 compared to a zero bit. The CSE's big number arithmetic implements a "dummy" step when processing a zero bit from the key such that this processing time is identical to the processing time of a set bit. Using this approach, an observer is unable to determine the number of set and unset bits from observing the timing behavior of the modular exponentiation operation.

The CSE Crypto Driver takes advantage of this feature by enabling the aforementioned functionality in the CSE for private key operations. This implies that the computation time using the private key is constant, hence mitigating timing attacks.

Appendix A. Glossary and Abbreviations

AES	Advanced Encryption Standard
API	Application Program Interface
CAVS	Cryptographic Algorithm Validation System
CBC	Cipher Block Chaining
CMVP	Cryptographic Module Validation Program
COTS	Commercial Off The Shelf
CRNGT	Continuous Random Number Generator Test
CSE	Converged Security Engine
CSP	Critical Security Parameter
CTR	Counter Mode
CVL	Component Validation List
DES	Data Encryption Standard
DMA	Direct Memory Access
DSA	Digital Signature Algorithm
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
FIPS	Federal Information Processing Standards Publication
HMAC	Hash Message Authentication Code
IG	Implementation Guidance
IOSF	Intel® On-Chip System Fabric
KAS	Key Agreement Schema
KAT	Known Answer Test
MAC	Message Authentication Code
ME	Management Engine
MMIO	Memory-Mapped I/O
NIST	National Institute of Science and Technology
OAEP	Optimal Asymmetric Encryption Padding
OEM	Original Equipment Manufacturers
O/S	Operating System
PBKDF	Password-Based Key Derivation Function
PCH	Platform Controller Hub
PCT	Pair-wise Consistency Test
PSS	Probabilistic Signature Scheme
RNG	Random Number Generator
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SKS	Secure Key Storage
SKU	Stock Keeping Unit

Appendix B. References

FIPS140-2	**FIPS PUB 140-2 - Security Requirements For Cryptographic Modules** May 2001 http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
FIPS140-2	**IG Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program** September 15, 2015 http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf
FIPS180-4	**Secure Hash Standard (SHS)** March 2012 http://csrc.nist.gov/publications/fips/fips180-4/fips 180-4.pdf
FIPS186-4	**Digital Signature Standard (DSS)** July 2013 http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** November 2001 http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** July 2008 http://csrc.nist.gov/publications/fips/fips198 1/FIPS-198 1_final.pdf
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography** Specifications Version 2.1 February 2003 http://www.ietf.org/rfc/rfc3447.txt
SP800-38A	**NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques** December 2001 http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
SP800-56A	**NIST Special Publication 800-56A Revision 1 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** May 2013 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-56ar.pdf
SP800-56B	**NIST Special Publication 800-56B Revision 1 - Recommendation for Pair Wise Key Establishment Using Integer Factorization Cryptography** September 2014 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br1.pdf
SP800-67	**NIST Special Publication 800-67 Revision 1 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** January 2012 http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf

SP800-90A	**NIST Special Publication 800-90A Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-90B	**NIST Draft Special Publication 800-90B - Recommendation for the Entropy Sources Used for Random Bit Generation** August 2012 http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90b.pdf
SP800-131A	**NIST Special Publication 800-131A - Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths** January 2011 http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf
SP800-132	**NIST Special Publication 800-132 - Recommendation for PasswordBased Key Derivation - Part 1: Storage Applications** December 2010 http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf
SP800-133	**NIST Special Publication 800-133 - Recommendation for Cryptographic Key Generation** December 2012 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-133.pdf