Panasonic Corporation

Panasonic Cryptographic Module

FIPS 140-2 Non-Proprietary Security Policy

Version 1.00

February 13, 2017

Table of Contents

Modification History

Date	Version	Description
2017/02/13	1.00	Initial version

References

Reference	*Full Specification Name*
[FIPS 140-2]	Security Requirements for Cryptographic modules, May 25, 2001
[FIPS 180-4]	Secure Hash Standard, FIPS 180-4, August, 2015
[FIPS 186-2]	Digital Signature Standard (DSS), FIPS 186-2, January, 2000
[FIPS 186-4]	Digital Signature Standard (DSS), FIPS 186-4, July, 2013
[FIPS 197]	Advanced Encryption Standard (AES), FIPS 197, November 26, 2001
[FIPS 198-1]	The Keyed-Hash Message Authentication Code (HMAC), FIPS 198-1, July, 2008
[SP 800-38A]	Recommendation for Block Cipher Modes of Operation, Methods and Techniques, SP 800-38A, December 2001
[SP 800-38B]	Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005
[SP 800-38D]	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, SP 800-38D, November 2007
[SP 800-67R1]	Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, SP 800-67 Revision 1, January 2012
[SP 800-90AR1]	Recommendation for Random Number Generation Using Deterministic Random Bit Generators, SP 800-90A Revision 1, June 2015
[SP 800-131AR1]	Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths, SP 800-131A Revision 1, November 2015

Abbreviation

Abbreviation	*Full spelling*
CGK	CMAC Generate Key
CSP	Critical Security Parameters
DDK	Data Decrypt Key
DEK	Data Encrypt Key
EDK	Encrypt / Decrypt Key
GEDK	GCM Encrypt / Decrypt Key
HGK	HMAC Generate Key
KAT	Known Answer Test
SGK	Signature Generate Key
SVK	Signature Verify Key

1 Introduction

This document is the nonproprietary security policy for the Panasonic Cryptographic Module Ver. 1.04 (hereafter referred as the Module). The Module is a software library providing a C-Language application program interface (API) for Cryptographic functionality. The Module is classified by FIPS 140-2 as a software module, multichip standalone module embodiment. The physical cryptographic boundary is the general purpose computer which the Module is installed. The logical cryptographic boundary is a single shared library. The Module performs no communications other than the calling application.

The Module meets FIPS 140-2 overall Level 1 requirements. The FIPS 140-2 security levels for the Module are listed in “Table1 Security Level of Security Requirements”.

Table 1 Security Level of Security Requirements

Security Requirement	Security Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

1.1 The Module Information

Name : Panasonic Cryptographic Module

Version : Ver. 1.04

The Module Block Diagram is shown in Figure 1.

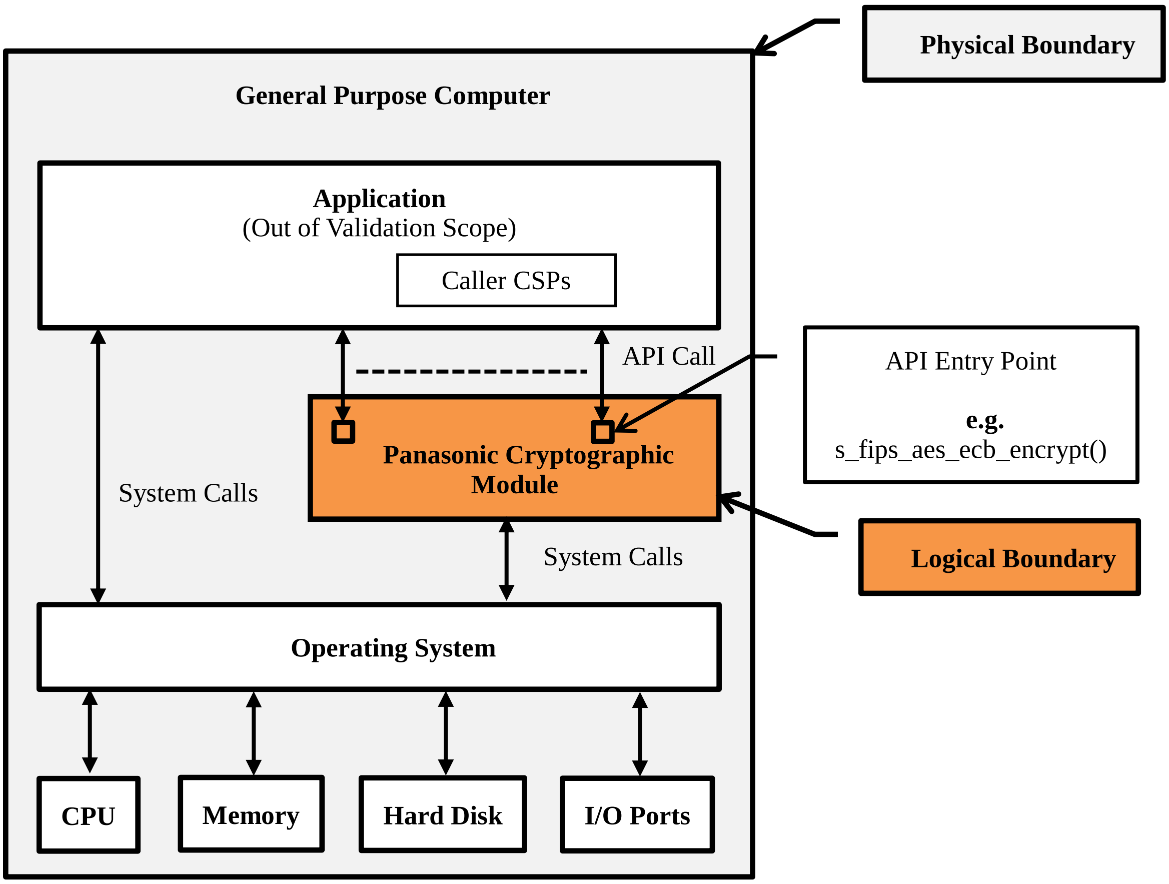

Figure 1 Module Block Diagram

2 Tested Configuration

Table 2 Tested Configuration

Operational Environment	Processor	Platform	Optimizations (Target)
Linux 3.13 (Ubuntu) 32bit	Intel® Core™ i7-4790 CPU @ 3.60GHz x 8	HP Elite Desk	Without AES-NI

3 Ports and Interfaces

Physical ports of the Module are the same as the computer system on which it is executing. The logical interface is the C-language application programming interface (API).

Table 3 Logical interfaces

Logical interface type	Description
Control input	*API entry point and corresponding stack parameters*
Data input	*API entry point data input stack parameters*
Status output	*API entry point return values*
Data output	*API entry point data output stack parameters*

The Module is a software module and control of the physical ports is out of scope. However when the Module is in “Self-test state” or “Error State”, all data output on the logical data output interface is inhibited. When the Module is in “Error State”, it returns an error value on the entry point return values of the Status output API (Not output from Data output interface).

4 Modes of Operation and Cryptographic Functionality

The Module supports one FIPS 140-2 Approved mode and one Non-Approved mode. Table 4 lists the Approved Cryptographic Functions.

Table 4 FIPS Approved Cryptographic Functions

Function	Algorithm	Options	Cert #
Random Number Generation	[SP 800-90AR1] DRBG	**Hash_DRBG**: [Prediction Resistance Tested: Enabled and Not Enabled (SHA-1 , SHA-224 , SHA-256 , SHA-384 , SHA-512) (SHS Val#3603)] **HMAC_DRBG**: [Prediction Resistance Tested: Enabled and Not Enabled (SHA-1 , SHA-224 , SHA-256 , SHA-384 , SHA-512) (HMAC Val#2905)] **CTR_DRBG**: [Prediction Resistance Tested: Enabled and Not Enabled; BlockCipher_Use_df: (AES-128 , AES-192 , AES-256) (AES Val#4366) BlockCipher_No_df: (AES-128 , AES-192 , AES-256) (AES Val#4366)] (See **NOTE** below.)	**1404**
Encryption, Decryption and CMAC	[SP 800-67R1] [SP 800-38A]	**TECB**(KO 1 e/d,) ; **TCBC**(KO 1 e/d,)	**2361**
	[FIPS 197] AES [SP 800-38A]	**ECB** (e/d; 128 , 192 , 256); **CBC** (e/d; 128 , 192 , 256); **CFB128** (e/d; 128 , 192 , 256); **CTR** (ext only; 128 , 192 , 256) **CMAC** (Generation) **(KS: 128;** Block Size(s): Full / Partial ; **Msg Len(s)** Min: 0 Max: 2^16 ; **Tag Len(s)** Min: 16 Max: 16) **(KS: 192**; Block Size(s): Full / Partial ; **Msg Len(s)** Min: 0 Max: 2^16 ; **Tag Len(s)** Min: 16 Max: 16) **(KS: 256**; Block Size(s): Full / Partial ; **Msg Len(s)** Min: 0 Max: 2^16 ; **Tag Len(s)** Min: 16 Max: 16) **GCM** (**KS: AES**_128(e/d) Tag Length(s): 128 120 112 104 96 64 32) (**KS: AES**_192(e/d) Tag Length(s): 128 120 112 104 96 64 32) (**KS: AES**_256(e/d) Tag Length(s): 128 120 112 104 96 64 32) **IV Generated:** (Internally (using Section 8.2.1)) ; **PT Lengths Tested:** (0 , 128 , 256 , 104 , 408) ; **AAD Lengths tested:** (0 , 128 , 384 , 160 , 720) ; **IV Lengths Tested:** (8 , 1024) ; **96BitIV_Supported**	**4366**
	[SP 800-38B] CMAC [SP 800-38D] GCM		

		GMAC_Supported	
Message Digests	[FIPS 180-4]	**SHA-1** (BYTE-only) **SHA-224** (BYTE-only) **SHA-256** (BYTE-only) **SHA-384** (BYTE-only) **SHA-512** (BYTE-only) **SHA-512_224** (BYTE-only) **SHA-512_256** (BYTE-only)	**3603**
Keyed Hash	[FIPS 198-1] HMAC	**HMAC-SHA1 (Key Sizes Ranges Tested: KS<BS KS=BS KS>BS) SHS Val#3573** **HMAC-SHA224 (Key Size Ranges Tested: KS<BS KS=BS KS>BS) SHS Val#3573** **HMAC-SHA256 (Key Size Ranges Tested: KS<BS KS=BS KS>BS) SHS Val#3573** **HMAC-SHA384 (Key Size Ranges Tested: KS<BS KS=BS KS>BS) SHS Val#3573** **HMAC-SHA512 (Key Size Ranges Tested: KS<BS KS=BS KS>BS) SHSVal#3573** **HMAC-SHA512_224 (Key Size Ranges Tested: KS<BS KS=BS KS>BS) SHSVal#3573** **HMAC-SHA512_256 (Key Size Ranges Tested: KS<BS KS=BS KS>BS) SHSVal#3573**	**2905**
Digital Signature and Asymmetric Key Generation	[FIPS 186-4] RSA	**FIPS186-2:** **ALG[RSASSA-PKCS1_V1_5]:** **SIG(gen) 4096 , SHS: SHA-224Val#3603 , SHA-256Val#3603 , SHA-384Val#3603 , SHA-512Val#3603 , SHS: SHA-224Val#3603 , SHA-256Val#3603 , SHA-384Val#3603 , SHA-512Val#3603** **SIG(ver): 1024 , 1536 , 2048 , 3072 , 4096 , SHS: SHA-1Val#3603 , SHA-224Val#3603 , SHA-256Val#3603 , SHA-384Val#3603 , SHA-512Val#3603** **ALG[RSASSA-PSS]: SIG(gen); 4096 , SHS: SHA-256Val#3603 , SHA-256Val#3603 , SHA-384Val#3603 , SHA-512Val#3603 , SHS: SHA-256Val#3603 , SHA-256Val#3603 , SHA-384Val#3603 , SHA-512Val#3603** **SIG(ver); 1024 , 1536 , 2048 , 3072 , 4096 , SHS: SHA-1Val#3603 , SHA-224Val#3603 , SHA-256Val#3603 , SHA-384Val#3603 , SHA-512Val#3603** **FIPS186-4:** **186-4KEY(gen): FIPS186-4_Fixed_e (10001) ;** **PGM(ProbRandom: (2048 , 3072) PPTT:(C.2)**	**2364**

		ALG[RSASSA-PKCS1_V1_5] **SIG(gen) (2048 SHA(1 , 224 , 256 , 384 , 512 , 512-224 , 512-256)) (3072 SHA(1 , 224 , 256 , 384 , 512 , 512-224 , 512-256))** **SIG(gen) with SHA-1 affirmed for use with protocols only.** **SIG(Ver) (1024 SHA(1 , 224 , 256 , 384 , 512 , 512-224 , 512-256)) (2048 SHA(1 , 224 , 256 , 384 , 512 , 512-224 , 512-256)) (3072 SHA(1 , 224 , 256 , 384 , 512 , 512-224 , 512-256))** **ALG [RSASSA-PSS]:** **Sig(Gen): (2048 SHA(1 SaltLen(10) , 224 SaltLen(15) , 256 SaltLen(20) , 384 SaltLen(25) , 512 SaltLen(33) , 512-224 SaltLen(15) , 512-256 SaltLen(20))) (3072 SHA(1 SaltLen(10) , 224 SaltLen(15) , 256 SaltLen(20) , 384 SaltLen(25) , 512 SaltLen(33) , 512-224 SaltLen(15) , 512-256 SaltLen(20)))** **SIG(gen) with SHA-1 affirmed for use with protocols only.** **Sig(Ver): (1024 SHA(1 SaltLen(10) , 224 SaltLen(15) , 256 SaltLen(20) , 384 SaltLen(25))) (2048 SHA(1 SaltLen(10) , 224 SaltLen(15) , 256 SaltLen(20) , 384 SaltLen(25) , 512 SaltLen(33) , 512-224 SaltLen(15) , 512-256 SaltLen(20))) (3072 SHA(1 SaltLen(0) , 224 SaltLen(0) , 256 SaltLen(0) , 384 SaltLen(0) , 512 SaltLen(0) , 512-224 SaltLen(0) , 512-256 SaltLen(0)))** **SHA Val#3603** **DRBG: Val# 1404**	

NOTE: Following cryptographic functions can not be used in FIPS mode, but can be used in Non-FIPS mode.

- DRBG
  - **Hash_DRBG**: [Prediction Resistance Tested: Enabled and Not Enabled
    (SHA-1 , SHA-224)]
  - **HMAC_DRBG**: [Prediction Resistance Tested: Enabled and Not Enabled
    (SHA-1 , SHA-224)]
  - **CTR_DRBG**: [Prediction Resistance Tested: Enabled and Not Enabled;
    BlockCipher_Use_df: (AES-128 , AES-192)
    BlockCipher_No_df: (AES-128 , AES-192 , AES-256)]

Following is the Non-FIPS Approved But Allowed Cryptographic Functions in FIPS mode.

- NDRNG (entropy source for SP 800-90A DRBG). Entropy source is the /dev/random function in the operating environment as shown in Table 2. The minimum entropy per 8bit sample of the entropy source is estimated as 6.536 bits.

- Non SP 800-56B Compliant RSA Key Transport (RSA may be used to encrypt / decrypt key for key transport. No Keys are established into the Module using RSA. [SP 800-131AR1] “Through December 31, 2017, the use of this scheme is deprecated if len(n)>= 2048. The use of this scheme is disallowed after December 31, 2017.”)

Following functions are the Non-Approved Cryptographic Functions only available in non-FIPS mode.

- RSA Signature Generation with other than 2048 bit key or 3072 bit key
- RSA Signature Generation with SHA-1
- RSA encryption, decryption (RSAES-OAEP, RSAES-PKCS1-v1_5, with no internal data padding)
- DES
- AES PKCS5padding CBC encryption, decryption

4.1 Critical Security Parameters and Public Keys

This section describes all CSPs used by the Module. All services of the Module which access to this CSP is described in in Section 4.

Table 5-1 Critical Security Parameters (DRBG)

CSP	Description
Hash_DRBG CSP	V (440/888 bits) and C (440/888 bits)
HMAC_DRBG CSP	V (160/224/256/384/512 bits) and Key (160/224/256/384 /512 bits)
CTR_DRBG CSP	V (128 bits) and Key (AES 128/192/256 bits)

Generation/Input: These CSPs are generated by calling internal DRBG instantiate or reseed function with entropy from the internal entropy acquisition function. When the acquired entropy becomes unnecessary, it is automatically erased by the internal entropy elimination function.

Output: These CSPs are not output.

Storage: The DRBG status value which is in plaintext is stored in allocated memory for application by OS.

Zeroization: Zeroization by calling internal DRBG uninstantiate function is executed by calling an API of a zeroize service.

Table 5-2 Critical Security Parameters (Other than DRBG)

CSP	Description	Generation/Input
RSA SGK	RSA (2048bit, 3072bit) private key for signature generation	The Module implements the asymmetric key generation service. The calling application is responsible for the storage of generated keys returned from this module.
RSA DDK	RSA (2048bit, 3072bit) private key for data decryption	
AES EDK	AES (128/192/256bits) encrypt/decrypt key	The Module implements the symmetric encrypt / decrypt key generation service. The symmetric encrypt / decrypt key also enters the Module's logical boundary in plaintext as API parameters, associated by memory location. However, none cross the physical boundary. The calling application is responsible for the storage of generated keys returned from this module.
AES CGK	AES (128/192/256bits) CMAC generate key	
AES GEDK	AES (128/192/256bits) GCM encrypt/decrypt key	
TDES EDK	TDES (3-key, 168bit) encrypt/decrypt key TDES (2-key, 112bit) decrypt key	
HGK	Keyed hash key	The Module implements the keyed hash key generation service. The keyed hash key also enters the Module's logical boundary in plaintext as API parameters, associated by memory location. However, none cross the physical boundary. The calling application is responsible for the storage of generated keys returned from this module.

Output: These CSPs are output to the calling API. However, none cross the physical boundary.

Storage: These CSPs are stored as plaintext in RAM as specified by the application program. The Module uses these CSPs stored in the data area such as the stack area of the calling application of the Module API. This Module does not permanently store any CSP (beyond the lifetime of the API call).

Zeroization: The application that invokes the API is responsible for the parameters that are input / output to / from this Module.

Table 5-3 Public Keys

Keys	Description	Generation/Input
RSA SVK	RSA (1024bit, 2048bit, 3078bit) public key for signature verification	The Module implements the asymmetric key generation service. The public key also enters the Module's logical boundary in plaintext as API parameters, associated by memory location. However, none cross the physical boundary. The calling application is responsible for the storage of generated keys returned from this module.
RSA DEK	RSA (2048bit, 3078bit) public key for data encryption	

Output: These keys are output to the calling API. However, these keys do not cross the physical boundary.

Storage: Is stored as plaintext in RAM as specified by the application program. The Module uses these keys stored in the data area such as the stack area of the calling application of the Module API. This module does not permanently store any key (beyond the lifetime of the API call).

Zeroization: The application that invokes the API is responsible for the parameters that are input / output to / from this module.

5 Roles, Authentication and Services

The Module supports the required User and Crypto Officer roles. Only one role is active at the same time, and the Module does not allow multiple operations simultaneously. The Role is assumed by the service being used.

- Crypto Officer Role (CO): Installation of the Module on the host computer system and calling of administrative API functions.
- User Role (User): Calling of non-administrative API functions.
- All services implemented in the Module are described in "Table 6-1 FIPS mode Services and CSP Access", with their API functions and the access to CSPs.

Table 6-1 FIPS mode Services and CSP Access

Service	Description	Role		Input	Output	CSPs	Type of access to CSP
		CO	User				
Initialize module	Used to setup and instantiate the DRBG. Transit to a state where other services can be executed.	Y	N	Calling an API and initialization information	Execution result	Hash_DRBG CSP HMAC_DRBG CSP CTR_DRBG CSP	Write access to CSP
Zeroize	Used to zeroize internal DRBG CSP.	Y	N	Calling an API	Execution result	Hash_DRBG CSP HMAC_DRBG CSP CTR_DRBG CSP	Write access to CSP which needs to be zeroized
Random number generation	Used to obtain random data. Uses and updates Hash_DRBG CSPs, HMAC_DRBG CSPs, CTR_DRBG CSPs.	N	Y	Calling an API and control inputs for random number	Random bit sequence, Execution result	Hash_DRBG CSP HMAC_DRBG CSP CTR_DRBG CSP	Read and write access to CSP

				generation			
	Used to reseed a DRBG instance. Uses and updates Hash_DRBG CSPs, HMAC_DRBG CSPs, CTR_DRBG CSPs..	N	Y	Calling an API and control inputs for reseeding	Execution result	Hash_DRBG CSP HMAC_DRBG CSP CTR_DRBG CSP	Read and write access to CSP
	Used to execute DRBG health test.	N	Y	Calling an API	Health Test result	N/A	N/A
Set mode	Used to set module operation mode	Y	N	Calling an API and set mode information	Execution result	N/A	N/A
Show module status and mode	Used to retrieve module status and mode	Y	N	Calling an API	Status Mode	N/A	N/A
Show version	Used to retrieve module version	Y	N	Calling an API	Version	N/A	N/A
Asymmetric key generation	Used to generate RSA keys.	N	Y	Calling an API and control inputs for asymmetric key generat	RSA SGK	RSA SGK	Write access to CSP

				ion			
Symmetric encrypt/decrypt key generation	Used to generate symmetric encrypt/decrypt keys:	N	Y	Calling an API and control inputs for symmetric key generation	AES EDK, TDES EDK, AES GEDK	AES EDK, TDES EDK, AES GEDK,	Write access to CSP
Symmetric digest key generation	Used to generate symmetric digest keys:	N	Y	Calling an API and control inputs for symmetric digest key generation	AES CGK	AES CGK,	Write access to CSP
Keyed hash key generation	Used to generate keyed hash keys:	N	Y	Calling an API and control inputs for keyed hash key generation	HGK	HGK	Write access to CSP
Digital signature	Used to generate RSA digital signatures. Executes using RSA	N	Y	Calling an API,	Signature	RSA SGK	Read access

	SGK; (passed in by the calling process).			RSA SGK, and data to be signed			to CSP
	Used to verify RSA digital signatures. Executes using RSA SVK; (passed in by the calling process).	N	Y	Calling an API, RSA SVK, and data to be signed signature	Verification result	RSA SVK	Read access to CSP
Key Transport	Used to encrypt key. Executes using RSA DEK; (passed in by the calling process).	N	Y	Calling an API, RSA DEK, and plain text key	Cipher text key	RSA DEK	Read access to CSP
	Used to decrypt key. Executes using RSA DDK; (passed in by the calling process).	N	Y	Calling an API, RSA DDK, and cipher text key	Plain text key	RSA DDK	Read access to CSP
Symmetric encrypt/decrypt	Used to encrypt data. Executes using TDES EDK (passed in by the calling process).	N	Y	Calling an API, TDES EDK,	Cipher text	TDES EDK	Read access to CSP

				and plain text			
	Used to decrypt data. Executes using TDES EDK (passed in by the calling process).	N	Y	Calling an API, TDES EDK, and cipher text	Plain text	TDES EDK	Read access to CSP
	Used to encrypt data. Executes using AES EDK (passed in by the calling process).	N	Y	Calling an API, AES EDK, and plain text	Cipher text	AES EDK	Read access to CSP
	Used to decrypt data. Executes using AES EDK (passed in by the calling process).	N	Y	Calling an API, AES EDK, and cipher text	Plain text	AES EDK	Read access to CSP
	Used to encrypt data. Executes using AES GEDK (passed in by the calling process).	N	Y	Calling an API, AES GEDK, and plain text	Cipher text and authentication tag	AES GEDK	Read and write access to CSP
	Used to decrypt data. Executes using AES GEDK (passed in by the calling process).	N	Y	Calling an API, AES GEDK, cipher	Plain text	AES GEDK	Read and write access to CSP

				text and authentication tag			
Symmetric digest	Used to generate data integrity with CMAC. Executes using AES CGK (passed in by the calling process).	N	Y	Calling an API, AES CGK, and plain text	Digest	AES CGK	Read and write access to CSP
Keyed hash	Used to generate or verify data integrity with HMAC. Executes using HGK(passed in by the calling process).	N	Y	Calling an API and plain text	Digest	HGK	Read and write access to CSP
Key zeroize	Used to zeroize TDES EDK.	N	Y	Calling an API, and CSP	Execution result	TDES EDK,	Read and write access to CSP which needs to be zeroized
	Used to zeroize AES EDK.					AES EDK	
	Used to zeroize AES GEDK.					AES GEDK	
	Used to zeroize AES CGK.					AES CGK	
	Used to zeroize AES HGK.					HGK	
	Used to zeroize RSA SGK.	N	Y	Calling an API and key	Execution result	RSA SGK	Write access to CSP which needs to be zeroized
	Used to zeroize RSA public key.					N/A	
Message digest	Used to generate a SHA1 or SHA2 message digest. Does not access CSPs.	N	Y	Calling an API and	Digest	N/A	N/A

				plain text			
Self-test	Used to excecute power-up self-test.	N	Y	Calling an API	Self-test result	N/A	N/A

Table 6-2 Non-FIPS mode services

Service	Description
Set mode	Used to set module operation mode.
Show module status and mode	Used to retrieve module status and mode.
Show version	Used to retrieve module status and mode
Non-FIPS random number generation	Used for random number generation which operates in non-FIPS mode.
Non-FIPS asymmetric key generation	Used to generate RSA keys.
Non-FIPS symmetric encrypt/decrypt key generation	Used to generate symmetric encrypt/decrypt keys.
Non-FIPS keyed hash key generation	Used to generate keyed hash keys.
Non-FIPS digital signature	Used to generate RSA digital signatures.
	Used to verify RSA digital signatures.
Non-FIPS key Transport	Used to encrypt key.
	Used to decrypt key.
Non-FIPS asymmetric encrypt/decrypt	Used for RSAES-OAEP encryption/decryption which operates in non-FIPS mode.
	Used for RSAES-PKCS1-v1_5 encryption/decryption which operates in non-FIPS mode.
	Used for RSA encryption/decryption with nopadding which operates in non-FIPS mode.
Non-FIPS symmetric encrypt/decrypt	Used to encrypt data.
	Used to decrypt data.
	Used to encrypt data.
	Used to decrypt data.
	Used to encrypt data.
	Used to decrypt data.
Non-FIPS symmetric digest	Used to generate data integrity with CMAC.
Non-FIPS keyed Hash	Used to generate or verify data integrity with HMAC.
Non-FIPS key zeroize	Used to zeroize symmetric encrypt/decrypt keys.
	Used to zeroize symmetric digest keys.
	Used to zeroize keyed hash keys.
	Used to zeroize RSA secret key.
	Used to zeroize RSA public key.
Non-FIPS message digest	Used to generate a SHA1 or SHA2 message digest.

6 Self-test

The Module performs self-tests listed in “Table 7a Power Up Self-Tests” when the Module is loaded.

Table 7a Power Up Self-Tests

Algorithm	Type	Test Attributes
Software integrity	-	HMAC SHA-256
HMAC	KAT	One KAT per SHA-1, SHA-512
AES	KAT	Separate encrypt and decrypt, CBC mode, 128 bit key length
AES GCM	KAT	Separate encrypt and decrypt, 128 bit key length
AES CMAC	KAT	Generate MAC, 128 bit key lengths
TDES	KAT	Separate encrypt and decrypt, CBC mode, 3-Key
RSA	KAT	Sign and verify using 2048 bit key, SHA256, PKCS #1 v2.1 RSASSA-PKCS1-v1_5 Encrypt and Decrypt using 2048 bit key, SHA256, PKCS #1 v2.1 RSAES-PKCS1-v1_5 Encrypt and Decrypt using 2048 bit key, with no internal data padding
DRBG	KAT	CTR_DRBG: AES, 256 bit with derivation function HASH_DRBG: SHA-256 HMAC_DRBG: SHA-256

The Known answer test and Software integrity test described in table “7a” is executed automatically during module startup, without operator intervention. When the Power up self-test fails, all further cryptographic functions are disabled and the Module enters an error state.

The Known answer test and Software integrity test described in table “7a” is executed by calling Self-test API. When the self-test fails, all further cryptographic functions are disabled and the Module enters an error state.

Table 7b Conditional Tests

Algorithm	Test
DRBG	Tested as required by [SP 800-90AR1] Section 11 Tests the DRBG mechanism set by s_fips_init() to be used, and the KAT test and error check of the Cryptographic primitive settings. The test is executed on the following conditions. • When the calling application invoke s_fips_init()

	• When the internally called DRBG generate function is executed 0xFFFFFFF times.
RSA	Pairwise consistency test on each generation of a key pair
NDRNG	FIPS 140-2 continuous random number generator test

In case where Conditional Test fails, the Module transits to Error state according to [FIPS 140-2] requirement.

7 Operational Environment

The OS restricts and separates each user process in its own process space. Each process space is logically separated by the OS software and Hardware. The function of “the Module” is restricted to the calling applications process space, and implicitly fulfills the FIPS 140-2 requirement of single user mode operation.

8 Mitigation of other Attacks

The Module is not designed to mitigate against attacks which are outside of the scope of FIPS 140-2.

9 Physical Security

The FIPS 140-2 Area 5 Physical Security requirements do not apply because the Module is a software module.

10 Security Rules

10.1 Distribution

The Module is distributed as a digital data “fips_crypto.tgz” on a CD-R media. Fips_crypto.tgz is in a tar+gzip format.

10.2 Installation

The fips_crypto.tgz file creates the following directory structure in the current working directory.

```
|-include
|   |-p_fips.h                  “Header Files”
|-lib
|   |-libfipscrypto.so          “Shared librarys
|   |-libfipscrypto.so.sha256   “HMAC-SHA256 DATA”
|-doc                           “Documentation“
```

There are no restrictions on the installation location of the file “p_fips.h” and “libfipscrypto.so”. The file “libfipscrypto.so.sha256” must be installed in the same directory as “libfipscrypto.so”.

To reflect the installation information, execute the following command after the installation of the library.

% sudo ldconfig *(The directory where libfipscrypto.so is installed)*

All files must retain the original name for proper operation.

10.3 Linking the runtime executable application

To build an application embedding the Module the following steps must be taken.

- Specify the directory where the file “p_fips.h” is installed by using the “-I” option of the compiler at compile time.
- Specify the installed directory of the “libfipscrypto.so” using the “-L” option of the linker.
- Specify the use of the fipscrypto and supporting libraries by setting the linker options “-lfipscrypto -pthread” at link time.

10.4 Main operation

I. FIPS mode initialization

  i. After the startup of the Module execute the “s_fips_set_mode(S_FIPS_MODE_FIPS)” call and transit to FIPS mode.

  ii. Execute the “s_fips_init()”call and initialize FIPS mode.

II. Mode verification

i. To verify that the Module is operating in FIPS mode execute “s_fips_get_status()” and verify that the lower bits of the result is set to “S_FIPS_STATUS_FIPS_MODE”.

III. Error state recovery

i. If the 8 higher bits of the return code of “s_fips_get_status()” call is not “S_FIPS_ERRORCODE_NONE” the Module is in error state. In such cases, execute “s_fips_reset()”call to reset the FIPS mode. After reset, execute the “s_fips_init()”call for proper FIPS mode operation.