Key# crypto

Software version: v1.3

FIPS 140-2 Non-Proprietary Security Policy

RAON SECURE

Revision History

History	Contents	Date	Writer	Approver
[KS_A1]Non-Proprietary Security Policy v1.3.0	Initial registration	2018.07.31	sjpark	hwpark
[KS_A1]Non-Proprietary Security Policy v1.3.1	Responds to First comments from CSTLab	2019.04.15	sjpark	hwpark
[KS_A1]Non-Proprietary Security Policy v1.3.2	Responds to Second comments from CSTLab	2019.04.17	sjpark	hwpark
[KS_A1]Non-Proprietary Security Policy v1.3.3	Responds to Third comments from CSTLab	2019.07.29	sjpark	hwpark
[KS_A1]Non-Proprietary Security Policy v1.3.4	Responds to First comments from CMVP	2020.04.27	sjpark	ikkang
[KS_A1]Non-Proprietary Security Policy v1.3.5	Change the description within table 5	2020.05.20	sjpark	ikkang
[KS_A1]Non-Proprietary Security Policy v1.3.6	Additional updates for CMVP comments	2020.05.22	sjpark	ikkang
[KS_A1]Non-Proprietary Security Policy v1.3.7	Change the version information display of the title part	2020.06.17	sjpark	ikkang

Table of Contents

1. Cryptographic Module Specification 4
2. Ports and Interfaces 5
3. Modes of Operation and Cryptographic Functionality 7
   3.1 Critical Security Parameters and Public Keys 9
4. Roles, Authentication, and Services 10
   4.1 Roles and Services 10
   4.2 Authentication 12
5. Self-Tests 12
   5.1 Power-Up Self-Tests 12
   5.2 Conditional Tests 14
6. Operational Environment 15
   6.1 General Purpose Computer 15
   6.1.1 Test Environment 15
   6.2 General Purpose Operating System 15
7. Security Rules 16
8. Mitigation of Other Attacks 16
9. Installation and Instantiation 16

1. Cryptographic Module Specification

This document is the non-proprietary security policy for Raonsecure Key# crypto Cryptographic Module, hereinafter called the “Cryptographic Module” or “Module”.

This Cryptographic Module was created in a C-based dynamic library format and is supported by Microsoft Windows. It is a multi-chip standalone module embodiment and is composed as a pure software-only library.

The Cryptographic Module reference is as follows.

Cryptographic Module Identification	Key# crypto
Version	v1.3
Components	KeySharpCryptoFips.dll
Developer	RaonSecure Co., Ltd.

Table 1 Cryptographic Module reference

The Cryptographic Module complies with overall security level 1 of the FIPS 140-2 standard. The security level is as follows.

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	3
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

Table 2 Security Levels per FIPS 140-2 Area

The block diagram and data flow of this Cryptographic Module are as follows.

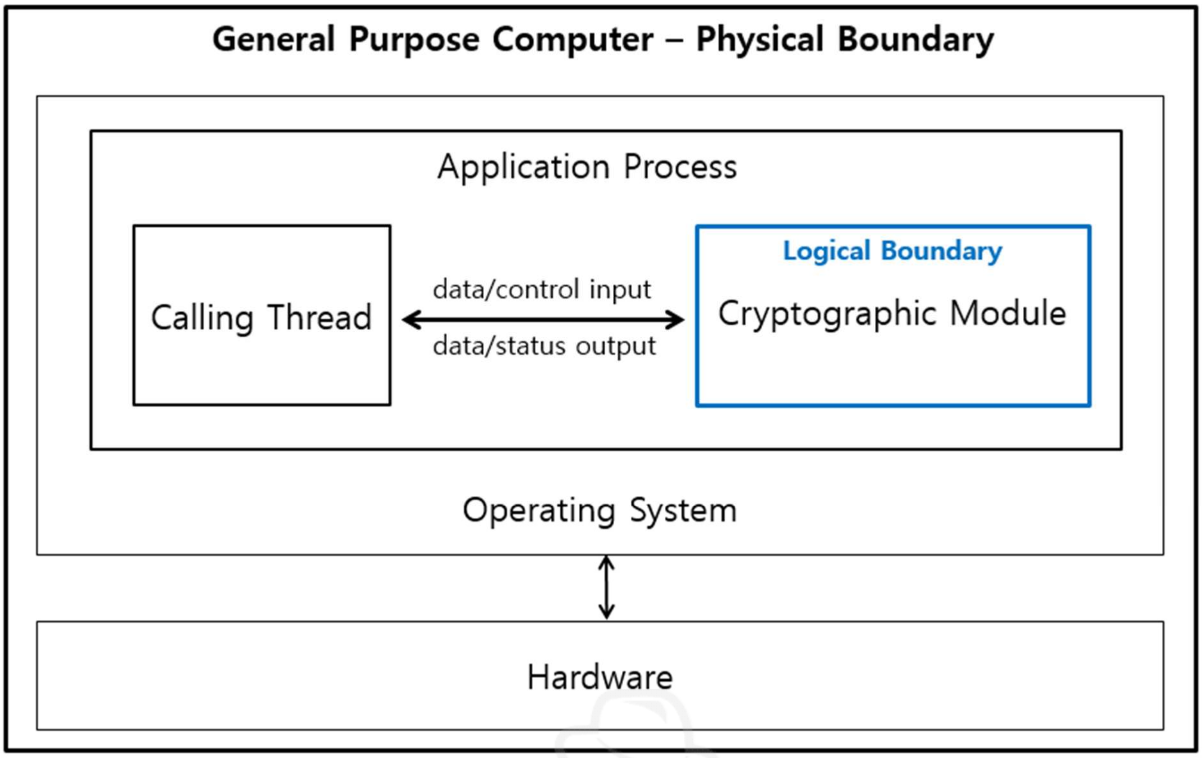

Figure 1 Module Block Diagram

The Cryptographic Module performs no communications other than with the calling application (the process that invokes the Cryptographic Module services via the API).

2. Ports and Interfaces

The physical ports of the Cryptographic Module are the same as the general purpose computer (GPC) on which it is executed. The logical interface is a C-language application program interface (API).

Logical Interface Type	Module Mapping
Data input	Parameters passed to the Cryptographic Module via API calls
Data output	Data returned from the Cryptographic Module via API calls
Control input	API Calls and/or parameters passed to API calls
Status output	Information received in response to API calls
Power	N/A

Table 3 Logical Interface Mapping

Figure 2 shows an example of the information flow for the API (C language) data input and output, control input, and status output. For example, if the application attempts to calculate the HMAC value, the application calls the HMAC API, KFC_HMAC.

At this time, the API call from the application program is sent to the Cryptographic Module through the control input interface and the Cryptographic Module receives control input to perform the HMAC computation. Also, the Cryptographic Module receives data input like a message (input in the figure), length of a message (inputLength), a secret key for HMAC (key), secret-key length (keyLength), and HMAC algorithm ID (macID).

The Cryptographic Module performs HMAC computations after receiving control/data input, and the success/failure of the computation is returned in the application. When the cryptographic operation is successful, the HMAC value (mac) and HMAC length (macLen) is returned to the application through the data output interface.

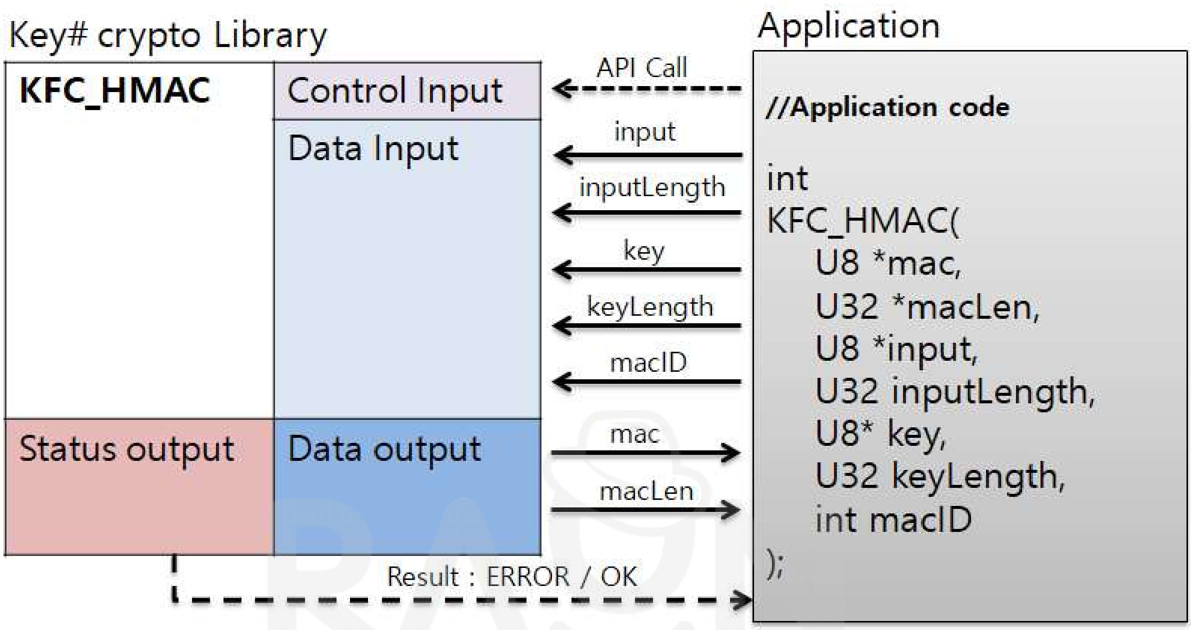

Figure 1: Logical interface of KFC_HMAC and relationship with the application

As a software module, control of the physical ports is outside the Cryptographic Module scope. However, when the Cryptographic Module is performing self-tests, or is in an error state, all output on the logical data output interface is inhibited. The Cryptographic Module is single-threaded and in error scenarios only error values are returned. (no data output is returned).

3. Modes of Operation and Cryptographic Functionality

This Cryptographic Module is designed to support both a FIPS Approved mode of operation and a non-FIPS Approved mode of operation. The mode of operation differs based on the algorithm identifier entered in a single function via an API call.

Algorithms subject to the FIPS Approved mode of operation provided by this Cryptographic Module are described in Tables 4 and 5. Table 6 lists the algorithms which are specific to the non-Approved mode of operation and these **shall not** be used when operating in the FIPS Approved mode of operation.

Function	Algorithm	Options	Standard	Cert
Random number generation & Symmetric key generation	DRBG	Hash: SHA2-256 Prediction resistance not supported	SP800-90A	#C739
Encryption & Decryption	AES	ECB, CBC, CFB8, CFB128 and OFB: Modes: Decrypt, Encrypt Key Lengths: 128, 192, 256 (bits) CTR: Counter Source: External Modes: Encrypt Key Lengths: 128, 192, 256 (bits)	FIPS 197	#C739
Message digest (HASH)	SHA	SHA-1, SHA-256, SHA-384 and SHA-512	FIPS 180-4	#C739
Keyed hash (HMAC)	HMAC	HMAC-SHA-1, HMAC-SHA2-256, HMAC-SHA2-384 and HMAC-SHA2-512	FIPS 198-1	#C739
Digital Signature & Asymmetric Key Generation	RSA	Key Gen (2048/3072) Sig GenPKCS1.5 (2048/3072 with SHA2-256, SHA2-384, SHA2-512) Sig VerPKCS1.5 (2048/3072 with SHA2-256, SHA2-384, SHA2-512) Sig GenPSS (2048/3072 with SHA2-256, SHA2-384, SHA2-512) Sig VerPSS (2048/3072 with SHA2-256, SHA2-384, SHA2-512)	FIPS 186-4	#C739
	ECDSA	PKG: P-256 PKV: P-256 Sig Gen: P-256 with SHA2-256, SHA2-384, SHA2-512 Sig Ver: P-256 with SHA2-256, SHA2-384, SHA2-512	FIPS 186-4	#C739
Key Generation	CKG	The module uses SP800-133 key	SP800-133	Vendor

		generation method using unmodified output from the Hash DRBG		Affirmed

Table 4 FIPS Approved Algorithms

This Cryptographic Module supports the following non-Approved but Allowed functions:

Function	Algorithm	Description	Standard
RSA (key transport; key establishment methodology provides between 112 and 128 bits of encryption strength)	RSA	Used by the calling application for encryption or decryption of keys. No CSPs are established into or exported out of the Cryptographic Module using these services.	PKCS#1: RSA Cryptography Standard
Entropy source	NDRNG	Used only to seed the Approved DRBG	

Table 5 Non-FIPS Approved but Allowed Cryptographic Functions

Function	Algorithm	Options	Standard
Encryption & Decryption	ARIA (non-compliant)	ECB, CBC, CFB8, CFB128 and OFB : Modes: Decrypt, Encrypt Key Lengths: 128, 192, 256 (bits) CTR: Counter Source: External Key Lengths: 128, 192, 256 (bits)	KS X 1213-1(2009)
	SEED (non-compliant)	ECB, CBC, CFB8, CFB128 and OFB : Modes: Decrypt, Encrypt Key Lengths: 128, 256 (bits) CTR: Counter Source: External Key Lengths: 128, 192, 256 (bits)	TTAS. KO-12.0004/R1(2005)
	LEA (non-compliant)	ECB, CBC, CFB8, CFB128 and OFB: Modes: Decrypt, Encrypt Key Lengths: 128, 192, 256 (bits) CTR: Counter Source: External Key Lengths: 128, 192, 256 (bits)	TTAK.KO-12.0223(2013)
Message Digest (HASH)	HAS (non-compliant)	160	TTAS.KO-12.0011/R2
Keyed Hash (HMAC)	HMAC (non-compliant)	HAS-160	FIPS 198-1
Digital Signature & Asymmetric Key Generation	RSA	Non-compliant SigGen-PKCS1.5, SigGenPSS, SigVer-PKCS1.5, SigVerPSS (2048/3072 with SHA-1)	FIPS 186-4
	KCDSA (non-compliant)	PQG Gen Key Pair Gen, Sig Gen, Sig Ver when	TTAS, KO-12.000/R1

		using: - TTAK.KO-12.0001_R4 RNG - 1024 bit key size with HAS160 - 2048 bit key size with SHA-256	

Table 6 Non-Approved Algorithms (Non-Approved Mode Only)

3.1 Critical Security Parameters and Public Keys

This part explains all CSPs used in the Cryptographic Module. All access to CSPs through module services is explained in Section 4.

CSPs	Description
RSA SGK	RSA (2048 and 3072 bits) signature generation key
RSA KDK	RSA (2048 and 3072 bits) key decryption key
ECDSA SGK	ECDSA (P-256) signature generation key
AES EDK	AES (128 / 192 / 256) encrypt / decrypt key
HMAC Key	Keyed hash key (160 / 256 / 384 / 512)
Hash_DRBG	V (440 / 888 bits) and C (440 / 888 bits) entropy input (length dependent on security strength)

Table 7 Critical Security Parameters

Key Name	Description
RSA SVK	RSA (2048 and 3072 bits) signature verification public key
RSA KEK	RSA (2048 and 3072 bits) key encryption key
ECDSA SVK	ECDSA (P-256) signature verification key

Table 8 Public Keys

For all CSPs and Public Keys:

Random Number Generation - The module employs an Approved SP 800-90A Hash_DRBG for the creation of random numbers. The DRBG is instantiated during module initialization and the module loads the DRBG using the Hash_DRBG mechanism with SHA2-256 and derivation function without prediction resistance.

The Cryptographic Module uses the Windows Random Number Generator (RNG) as the entropy source for seeding the DRBG. The entropy is provided by the operational environment using the Microsoft CNG (Cryptography, Next Generation) API via a BCryptGenRandom function call from the Windows 10 bcrypt.dll library, which is within the module's physical boundary but outside the module's logical boundary. The Windows entropy source provides at least 112 bits of entropy to the DRBG during initialization (seed) and reseeding (reseed).

The Cryptographic Module performs a continuous self-test on the output of Windows RNG to ensure that consecutive random numbers do not repeat. Also, the module performs the DRBG health tests as defined in section 11.3 of [SP800-90A]

Storage - RAM, is associated to entities by memory location. The Cryptographic Module stores DRBG state values for the lifetime of the DRBG instance. The Cryptographic Module uses CSPs passed in by the calling application on the stack. The Cryptographic Module does not store any CSP persistently (beyond the lifetime of an API call), with the exception of DRBG state values used for the Cryptographic Module's key generation service.

Generation - The cryptographic module implements a NIST SP 800-133 key generation function using a NIST SP 800-90A compliant DRBG for the generation of symmetric keys and asymmetric ECDSA and RSA keys, as shown in the table of CSPs above. The calling application is responsible for storage of generated keys returned by the cryptographic module

Key Entry - All CSPs enter the Cryptographic Module's logical boundary in plaintext as API parameters, associated by memory location. However, none cross the physical boundary.

Output - The Cryptographic Module does not output CSPs, other than as explicit results of key generation services. However, none cross the physical boundary.

Zeroization - Zeroization of sensitive data is performed automatically by API function calls for temporarily stored CSPs. In addition, the Cryptographic Module provides a function called KSC_CM_StateFinal to explicitly destroy CSPs related to random number generation services. The calling application is responsible for parameters passed in and out of the Cryptographic Module.

CSPs, like a secret key or private key, exists only between the start of the operation function call and function call returns, and when the function call returns, zeroization is automatically performed. The method for zeroizing the CSP is to set the memory space as a 0 value, by using the memset function provided by C language.

4. Roles, Authentication, and Services

4.1 Roles and Services

There are two roles used in this Cryptographic Module – The User Role and the Crypto-Officer Role. The roles are logically separated according to the performed services.

The details of each role are explained as follows:

- ✓ ***User Role(User)***: This refers to the entities that can access all services offered by the module. The role of the user is to summon API calls offered by this Cryptographic Module to use the service.
- ✓ ***Crypto-Officer Role(CO)***: This is the entity for installing the Cryptographic Module and setup the operating system in a secure manner. The Crypto-Officer role also has access to the services provided by the module. The Crypto-Officer has the same access to services available to the User, but the Crypto-Officer does not have the authority to access keys or data.

The roles of Crypto-Officer and User are implicitly assumed as the Cryptographic Module provides no authentication. The services offered by this Cryptographic Module can be categorized per role as follows.

Role	Service	Details
CO	Initialize	Module initialization. Does not access CSPs.
CO	Self-test	Perform self-tests. Does not access CSPs.
CO	Show status	Functions that provide module status information: Does not access CSPs.
CO	Zeroize	Functions that destroy Hash_DRBG CSP.
CO,User	Random number generation	Used for random number and symmetric key generation. • Uses and updates Hash_DRBG CSP.
CO,User	Symmetric Key Generation	Used to generate AES keys using the Approved DRBG
CO,User	Asymmetric key generation	Used to generate ECDSA and RSA keys: RSA SGK, RSA SVK; ECDSA SGK, ECDSA SVK
CO,User	Symmetric encryp t/decrypt	Used to encrypt or decrypt data (passed in by the calling process).
CO,User	Message Digest (HASH)	Used to generate a SHA-1 or SHA-2 message digest. Does not access CSPs.
CO,User	Keyed Hash	Used to generate or verify data integrity with HMAC. Executes using HMAC Key (passed in by the calling process).
CO,User	Key transport	Used to encrypt or decrypt a key value on behalf of the calling pr ocess (does not establish keys into the module). Executes using RSA KDK, RSA KEK (passed in by the calling pro cess).
CO,User	Digital signature	Used to generate or verify RSA, or ECDSA digital signatures. Executes using RSA SGK, RSA SVK; ECDSA SGK, ECDSA SVK (passed in by the calling process)

Table 9 Services and CSP Access

The Cryptographic Module interface per service is as follows.

Service	Function
Initialize	KFC_CM_StateInit
Zeroize	KFC_CM_StateFinal
Change Status	KFC_CM_StateChange
Show Status	KFC_CM_StateInfo
Get Version	KFC_CM_Version
Get ErrorString	KFC_CM_ErrorString
Self-test	KFC_CM_SelfTest
Symmetric Key Generation	KFC_KEY_GenSecKey
Asymmetric key generation	KFC_KEY_GenKeyPair KFC_KEY_CheckKeyPair
Symmetric encrypt/decrypt (AES CTR only supports symmetric encrypt)	KFC_SYM_Encrypt KFC_SYM_Encrypt_Init KFC_SYM_Encrypt_Update KFC_SYM_Encrypt_Final

	KFC_SYM_Decrypt KFC_SYM_Decrypt_Init KFC_SYM_Decrypt_Update KFC_SYM_Decrypt_Final
Digital signature	KFC_ASYM_GenParam
	KFC_ASYM_Sign
	KFC_ASYM_Verify
Key transport	KFC_ASYM_Enrypt
	KFC_ASYM_Decrypt
Keyed Hash	KFC_HMAC KFC_HMAC_Init KFC_HMAC_Update KFC_HMAC_Final
Message Digest	KFC_Hash KFC_Hash_Init KFC_Hash_Update KFC_Hash_Final
Random number generation	KFC_Rand

Table 10 API Calls by Service

The Cryptographic Module defines CSPs in chapter 3.1. A number of the service APIs are for functions that perform cryptographic operations. Some of these accept CSPs (like a secret or private key) as parameters. There are also APIs for functions that generate keys and pass them back to the calling application. These CSPs are ephemeral and are not stored within the Cryptographic Module. After these CSPs have been used by the API functions, they are zeroized within the Cryptographic Module.

4.2 Authentication

Module services are logically separated through the API and the roles of Crypto-Officer and User are implicitly assumed, as the module does not provide authentication. Instead, the operator must authenticate to the underlying Windows OS in order to use the module.

5. Self-Tests

Correct operation of the Cryptographic Module is assured through the implemented power-up and conditional self-tests.

5.1 Power-Up Self-Tests

All power-up self-tests explained below, are invoked automatically by the module library at load time. The KSC_CM_StateInit function instantiates the Cryptographic Module. The Cryptographic Module defines the DllMain function and calls the KSC_CM_StateInit function in it. Should any self-test fail, the state of the Cryptographic Module transitions to a hard error state. The Cryptographic Module returns the error, which must be resolved by the Crypto-Officer in order to restore correct operation.

The following items are tested as part of the self-tests:

- ✓ ***Cryptographic algorithm test***: The cryptographic algorithm test performs the Known Answer Test (KAT) and the Pair-wise Consistency Test. The KAT that compares the result of performing the cryptographic operation with the known value. It is executed for the Approved algorithms in the Cryptographic Module to check for correct operation and when a failure occurs, the module transitions to the Hard Error state. The Pair-wise Consistency Test is performed using the public and private key pairs. If the pairwise test fails, the module transitions to the Hard Error state.
- ✓ ***Software integrity test***: In order to test the integrity of the module, the RSASSA-PSS signature attached to the Cryptographic Module library is verified by the public key at power-up to ensure the module has not been modified.
- ✓ ***Critical Functions Test***: As one of the critical functions, the health test of NIST SP 800-90A DRBG, Section 11.3 (instantiate, reseed and generate) are executed. The other critical function is the continuous RNG test, which continuously gathers entropy for comparison between the previous and current data block. The test checks to see if the entropy bit strings of the two blocks are the same, and proceeds to the error state if they are.

The following table specifies the error conditions which are possible for self-tests:

Test Item	Error Conditions
Cryptographic Algorithm Test	When the test results are not consistent with the known values and fail the known answer / pairwise consistency tests.
Software Integrity Test	When verification of the RSA signature fails
Critical Function Test	When the gathered entropy blocks continuously have the same values or the health test of the NIST SP 800-90A DRBG have failed.

Table 11 Self-Test Error Conditions

The following table contains the module's power-up self-tests:

Cryptographic Algorithm	Test Method
AES-128-ECB encrypt AES-128-ECB decrypt AES-192-ECB encrypt AES-192-ECB decrypt AES-256-ECB encrypt AES-256-ECB decrypt AES-128-CBC encrypt AES-128-CBC decrypt AES-192-CBC encrypt AES-192-CBC decrypt AES-256-CBC encrypt AES-256-CBC decrypt AES-128-CFB8 encrypt AES-128-CFB8 decrypt AES-192-CFB8 encrypt AES-192-CFB8 decrypt AES-256-CFB8 encrypt AES-256-CFB8 decrypt	Known Answer Test

AES-128-CFB128 encrypt AES-128-CFB128 decrypt AES-192-CFB128 encrypt AES-192-CFB128 decrypt AES-256-CFB128 encrypt AES-256-CFB128 decrypt AES-128-OFB encrypt AES-128-OFB decrypt AES-192-OFB encrypt AES-192-OFB decrypt AES-256-OFB encrypt AES-256-OFB decrypt AES-128-CTR encrypt AES-192-CTR encrypt AES-256-CTR encrypt	
RSA	SigGen-PKCS1.5 Known Answer Test SigVer-PKCS1.5, SigVerPSS Known Answer Test Pairwise consistency test with 2048 bit and 3072 bit key
ECDSA	Pairwise consistency test with P-256 curve
DRBG	SP 800-90A Hash DRBG known answer test SP 800-90A Section 11.3 Health Tests (instantiate, reseed and generate)
SHA-1	Known Answer Test
SHA-256	Known Answer Test
SHA-384	Known Answer Test
SHA-512	Known Answer Test
HMAC-SHA-1	Known Answer Test
HMAC-SHA-256	Known Answer Test
HMAC-SHA-384	Known Answer Test
HMAC-SHA-512	Known Answer Test
Software integrity test	RSASSA-PSS 2048-bit digital signature using SHA-256

Table 12 Test Method by Cryptographic Algorithm

5.2 Conditional Tests

The following conditional tests are automatically performed by the Cryptographic Module:

- ✓ ***Pair-wise consistency test*** **:** After RSA and ECDSA keys are generated, the consistency of the generated key pairs is automatically tested.
  - RSA checks whether the two-key encryption/decryption was successfully performed and whether digital signature verification was properly tested.
  - ECDSA checks whether digital signature verification is valid.
- ✓ ***Continuous RNG tests*** **:** The OE contains an RNG which seeds the NIST SP 800-90A DRBG and performs the required continuous RNG test. It records the previously gathered (512-bytes) of entropy in a safe memory space and compares it with the entropy gathered afterward. The test fails if the newly gathered entropy is the same as the recorded 512-bytes. A continuous test is also performed on the output of the NIST SP 800-90A DRBG

The following are the error conditions for each test item:

Test Item	Error Conditions
Pair-wise Consistency Test	When after signing as a private key, the verification as a public key fails
Software Load Test	N/A
Manual Key Entry Test	N/A
Continuous RNG Tests	When the results generated by the entropy source (RNG) or NIST SP 800-90A DRBG has repeatedly consistent values
Bypass Test	N/A

Table 13 Error Conditions of Conditional Self-Tests

6. Operational Environment

The operational environment of the Cryptographic Module is the general-purpose computer specified in Section 6.1, with the Windows operating system specified in Section 6.2

Note: The Cryptographic Module will operate correctly on other GPC platforms running Microsoft Windows; however, no claim can be made as to the correct operation of the module or the security strengths of the generated keys when ported to an operational environment which is not listed on the validation certificate.

6.1 General Purpose Computer

The Cryptographic Module executes on the hardware of a general-purpose computer. The Cryptographic Module was tested on the following general-purpose computer as per the requirements of FIPS 140-2, Level 1:

6.1.1 Test Environment

- PC: HP Pavilion Slimline s5-1250kr Desktop PC
- CPU: Intel(R) Core (TM) i5-2400 CPU @ 3.10GHz 3.10 GHz
- Memory: 12.00GB
- SSD: 256GB
- HDD: 1TB

6.2 General Purpose Operating System

The Cryptographic Module was tested on the following general-purpose operating system:

- Windows 10 (64-bit)

The Windows operating system segregates user processes into separate process spaces. Each process space is logically separated from all other processes by the operating system. The Cryptographic Module functions entirely within the process space of the calling application and implicitly satisfies the FIPS 140-2 requirement for a single user mode of operation.

7. Security Rules

The Cryptographic Module was designed with the following security rules in mind:

1. The Cryptographic Module shall provide two distinct operator roles. These are the User role, and the Crypto-Officer role.

2. The Cryptographic Module does not provide any operator authentication.

3. The operator shall be capable of commanding the Cryptographic Module to perform the Power-Up Self-Test using recycling power or KFC_CM_SelfTest function.

4. The module functions entirely within the process space of the calling application, and implicitly satisfies the FIPS 140-2 requirement for a single-user mode of operation.

5. The Cryptographic Module does not output intermediate key generation values.

6. The Cryptographic Module is available to perform services only after successfully completing the Power-Up Self-Tests.

7. Security functions listed in Table 6 in this Security policy are not allowed for use in the FIPS Approved mode of operation. When these algorithms are used, the Cryptographic Module is no longer operating in the FIPS Approved mode of operation. It is the responsibility of the calling application to zeroize all keys and CSPs prior to and after utilizing these non-Approved algorithms. As per CMVP IG 1.2: **CSPs defined in the Approved mode of operation, <u>shall not</u> be accessed or shared while in the non-Approved mode of operation.**

8. Mitigation of Other Attacks

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

9. Installation and Instantiation

The Cryptographic Module supports Approved and non-Approved modes of operation and there are no specific installation steps that need to be taken. The module binary can simply be placed in a folder on a GPC disk drive and accessed by a calling application.

All power-up self-tests are invoked by the operating system at library load time. Therefore, the state of the Cryptographic Module is already in the Approved mode of operation upon the successful invocation of the KFC_CM_StateInit function. The application can call the KFC_CM_StateInfo function and check the return code = KC_STATE_ID_CMVP (5) to verify that the module has been initialized in FIPS mode e.g. passed self-tests.