Mocana Cryptographic Suite B Module

Version 7.0.0f_u1

Non-Proprietary FIPS 140-3 Security Policy

Document Version: 1.2

Date: July 23, 2024

DigiCert, Inc.

2801 North Thanksgiving Way

Suite 500

Lehi, UT 84043

+1 800-896-7973

Table of Contents

List of Tables

List of Figures

1 General

This document defines the non-proprietary Security Policy for the Mocana Cryptographic Suite B Module version 7.0.0f_u1 hereafter denoted the Module. It contains the security rules under which the module must operate and describes how this module meets the requirements specified in FIPS 140-3 for a Security Level 1 module.

Table 1 – Security Level of Security Requirements

ISO/IEC 24759 Section	Security Requirement	Security Level
1	**General**	1
2	**Cryptographic Module Specification**	1
3	**Cryptographic Module Interfaces**	1
4	**Roles, Services and, Authentication**	1
5	**Software/Firmware Security**	1
6	**Operational Environment**	1
7	**Physical Security**	N/A
8	**Non-Invasive Security**	N/A
9	**Sensitive Security Parameter Management**	1
10	**Self-Tests**	1
11	**Life-Cycle Assurance**	1
12	**Mitigation of Other Attacks**	N/A
Overall		1

The Module design corresponds to the Module security rules. The security rules enforced by the Module are described in this document.

2 Cryptographic Module Specification

The primary purpose of this Module is to provide approved cryptographic routines to consuming applications via an Application Programming Interface (API).

The Module conforms to [ISO/IEC 19790:2012] Section 7.2 Cryptographic Module Specification.

The Module is a software, multi-chip standalone cryptographic module that runs on a general-purpose computer which is the Tested Operational Environment's Physical Perimeter (TOEPP).

The cryptographic boundary of the Module is the single shared object (SO), libmss.so, associated signature file, a persistent status file, and the CPU when PAA is enabled.

No components are excluded from the [ISO/IEC 19790:2012] A.2.2 requirements.

The Module supports the normal mode of operation under which all of the algorithms, security functions, and services are available; degraded operation is not supported.

The Module is intended for use by US Federal agencies or other markets that require FIPS 140-3 validated Security Level 1 software modules. The Module is intended to be used in dedicated purpose IOT (Internet of Things) devices and general-purpose computer systems.

2.1 Operational Environment

The Mocana Suite B Cryptographic Module is tested on the following operational environment(s):

Table 2 – Tested Operational Environments - Software

#	Operating System	Hardware Platform	Processor	PAA/PAI	Hypervisor and Host OS
1	Yocto Linux 3.1 (32-bit)	Xerox Explorer 6.5	Intel Atom E3950	with PAA	NA
2	Yocto Linux 3.1 (32-bit)	Xerox Explorer 6.5	Intel Atom E3950	without PAA	NA
3	Yocto Linux 3.1 (32-bit)	Xerox Explorer 8.0	Intel Atom x6413E	with PAA	NA
4	Yocto Linux 3.1 (32-bit)	Xerox Explorer 8.0	Intel Atom x6413E	without PAA	NA
5	Yocto Linux 3.1 (32-bit)	Xerox Alexandra Platform	ARM Cortex A53	without PAA	NA
6	Buildroot Linux 2020.02.01 (32-bit)	Ultra Charge & Communication Base	Ingenic X1600E	without PAA	NA
7	Buildroot Linux 2020.02.01 (32-bit)	Honeywell Cordless Ultra	Ingenic X2000	without PAA	NA

DigiCert also performed the testing of the Module on the following Operational Environment(s) and claims vendor affirmation on them:

Table 3 – Vendor Affirmed Operational Environment

#	Operating System	Hardware Platform
1	Ubuntu Linux 4.15 (64-bit)	Intel NUC with processor: i7-8650U with and without PAA
2	Yocto Linux 4.4.11	DVF99 using ARM926EJ-S
3	Yocto Linux 4.4.11	DVF101 using ARM Cortex-A9 with Neon
4	Wind River Linux 2.6.27.18	BCM111XX using ARM1176JZF-S

The CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when ported to an operational environment that is not listed on the validation certificate.

2.2 Cryptographic Boundary

The software block diagram in Figure 1 shows the module, interfaces with the Tested Operational Environment, and the delimitation of its cryptographic boundary, shown shaded in blue. The Cryptographic Boundary is shown in Figure 1 and is comprised of the shared library files (libmss.so) and the integrity check signature file (libmss.so.sig), the POST status file (mssp.bin), and the CPU when PAA is enabled. The dashed line in Figure 2 indicates the Logical Object.

The Module's operations occur via API calls from calling applications running within the same process as the Module.

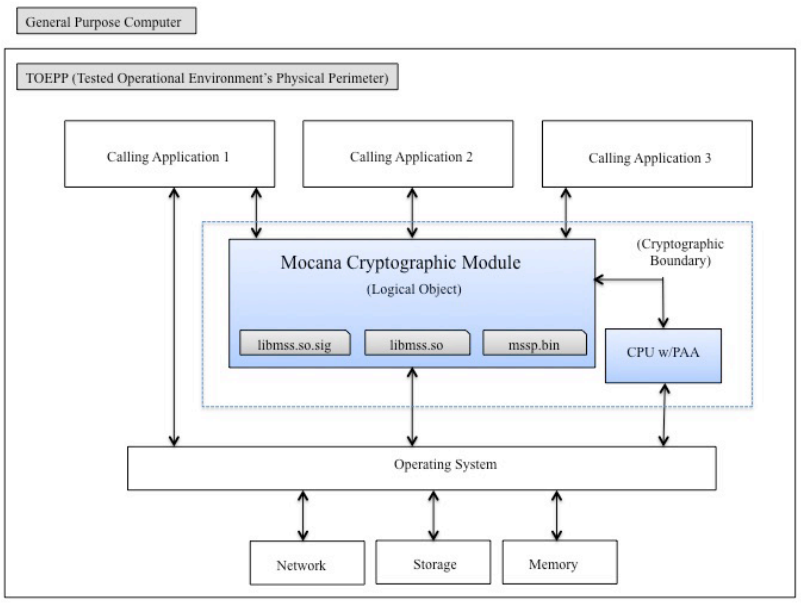

Figure 1 – Cryptographic Module Interface Design

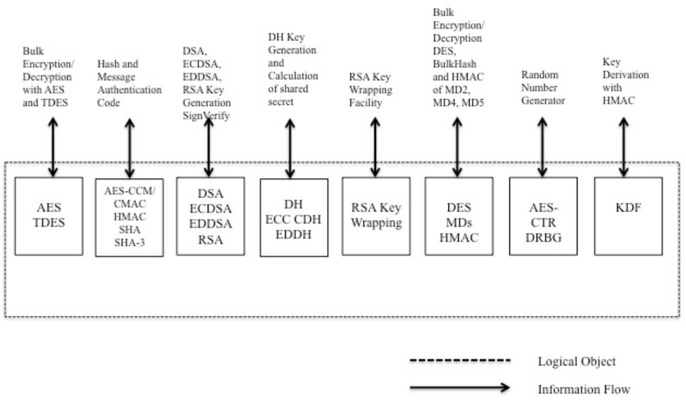
Bulk Encryption/ Decryption with AES and TDES
Hash and Message Authentication Code
DSA, ECDSA, EDDSA, RSA Key Generation SignVerify
DH Key Generation and Calculation of shared secret
RSA Key Wrapping Facility
Bulk Encryption/ Decryption DES, BulkHash and HMAC of MD2, MD4, MD5
Random Number Generator
Key Derivation with HMAC
AES TDES
AES-CCM/ CMAC HMAC SHA SHA-3
DSA ECDSA EDDSA RSA
DH ECC CDH EDDH
RSA Key Wrapping
DES MDs HMAC
AES-CTR DRBG
KDF
Logical Object
Information Flow

Figure 2 – Logical Object

2.3 Modes of Operation

The Module supports approved and non-approved modes of normal operation:

1. Approved mode: only approved or allowed security functions with sufficient key security strength can be used.
2. Non-approved mode: When non-approved security functions or approved security functions with insufficient key security strength are used.

The Module enters approved mode after pre-operational self-tests have successfully completed. Once the Module is operational, the mode of operation is implicitly assumed depending on the security function invoked and the security strength of the cryptographic keys. To provide an indicator of the current mode of operation, an asynchronous callout event mechanism is provided.

The application using the Module can register for a FIPS_eventLog call-out function to be used as the approved-mode/non-approved-mode indicator. The application's FIPS_eventLog function will be called from the Module at the beginning and end of each approved or non-approved service or algorithm. This FIPS_eventLog and the FIPS_EventType enumeration value provided as a parameter serves as a thread-safe status indicator of the current approved or non-approved mode of operation.

The Module uses the scenario of a shared indicator for multiple services, per IG 2.4.C example #3. It uses a dedicated status output interface that is a software callback function. The calling application using the Module registers a callback function to be called at the beginning and end of all services and algorithm implementations. The callback function FIPS_eventLog() is called with an enumeration that specifies the start and stop of services and algorithms. It also specifies whether they are approved or non-approved. The specific crypto-algorithm ID is also provided to this callback function. By interpreting the enumeration and algo-ID in the callback function, the callback function is the software analog of a hardware LED. Whether that function actually turns on and off a H/W LED or logs these FIPS indicator events, is left to the calling application as appropriate for the application domain.

Keys and CSPs shall not be shared between the approved and non-approved mode of operation.

2.3.1 Configuration of the Approved Mode of Operation

The approved mode of operation is configured at instantiation of the Module by the Cryptographic Officer role by execution of an application or protocol operating system process that uses the Module's cryptographic functions.

2.3.2 Configuration of the Non-Approved Modes of Operation

The Module transitions to the non-approved mode of operation when one of the non-approved security functions is utilized. The Module can transition back to the approved mode of operation by utilizing an approved security function.

2.4 Security Functions

The Module implements the approved and non-approved but allowed cryptographic functions listed in the table(s) below.

Table 4 – Approved Algorithms

CAVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A4653	AES [197]	CBC [38A]	Key Sizes: 128, 192, 256	Encrypt, Decrypt
A4653	AES [197]	CCM [38C]	Key Sizes: 128, 192, 256 Tag Len: 32, 48, 64, 80, 96, 112, 128	Authenticated Encrypt, Authenticated Decrypt
A4653	AES [197]	CFB128 [38A]	Key Sizes: 128, 192, 256	Encrypt, Decrypt
A4653	AES [197]	CTR [38A]	Key Sizes: 128, 192, 256	Encrypt, Decrypt
A4653	AES [197]	CMAC [38B]	Key Sizes: 128, 192, 256 MAC Len: 32-128 Increment 8	Message Authentication
A4653	AES [197]	ECB [38A]	Key Sizes: 128, 192, 256	Encrypt, Decrypt
A4653	AES [197]	OFB [38A]	Key Sizes: 128, 192, 256	Encrypt, Decrypt
A4653	AES [197]	XTS[1] [38E]	Key Sizes: 128, 256	Encrypt, Decrypt
A4654 A4655	AES [197]	GCM/GMAC [38D]	Key Sizes: 128, 192, 256 Tag Len: 32, 64, 96, 104, 112, 120, 128	Authenticated Encrypt, Decrypt
A4653	CTR_DRBG [90A]	CTR	Use_df (default), No_df AES-128, AES-192, AES-256	Deterministic Random Number Generation
A4653	DSA [186]	(L = 2048, N = 224) (L = 2048, N = 256) (L = 3072, N= 256)		KeyGen

[1] The XTS algorithm implementation includes a check to ensure Key_1 ≠ Key_2 per IG C.I.

CAVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
		(L = 2048, N = 224) SHA2 (224, 256, 384, 512) (L = 2048, N = 256) SHA2 (256, 384, 512) (L = 3072, N= 256) SHA2 (256, 384, 512)		PQG Gen
		(L = 2048, N = 224) SHA2 (224, 256, 384, 512) (L = 2048, N = 256) SHA2 (256, 384, 512) (L = 3072, N= 256) SHA2 (256, 384, 512) (L = 1024, N = 160) SHA-1, SHA2 (224, 256, 384, 512) – *Legacy Use*		PQG Ver
		(L = 2048, N = 224), SHA2 (224, 256, 384, 512) (L = 2048, N = 256), SHA2 (224, 256, 384, 512) (L = 3072, N= 256), SHA2 (224, 256, 384, 512)		SigGen
		(L = 2048, N = 224) SHA2 (224, 256, 384, 512) (L = 2048, N = 256) SHA2 (224, 256, 384, 512) (L = 3072, N = 256) SHA2 (224, 256, 384, 512) (L = 2048, N = 224) SHA-1 – *Legacy Use* (L = 2048, N = 256) SHA-1 – *Legacy Use* (L = 3072, N = 256) SHA-1 – *Legacy Use* (L = 1024, N = 160) SHA-1, SHA2(224, 256, 384, 512) – *Legacy Use*		SigVer
A4653	ECDSA [186]	P-224, P-256, P-384, P-521		KeyGen
		P-192*, P-224, P-256, P-384, P-521 – *Legacy Use		PKV
		P-224 SHA2 (224, 256, 384, 512) P-256 SHA2 (224, 256, 384, 512) P-384 SHA2 (224, 256, 384, 512) P-521 SHA2 (224, 256, 384, 512)		SigGen
		P-192* SHA-1, SHA2 (224, 256, 384, 512) – *Legacy Use P-224 SHA-1, SHA2 (224, 256, 384, 512) P-256 SHA-1, SHA2 (224, 256, 384, 512) P-384 SHA-1, SHA2 (224, 256, 384, 512) P-521 SHA-1, SHA2 (224, 256, 384, 512)		SigVer
A4653	EDDSA [186]	ED-25519 SHA-512 – 128-bits of security ED-448 SHAKE256 – 224-bits of security		Pure EdDSA (excluding the pre-hash or context variants - RFC8032) KeyGen, KeyVer, SigGen, SigVer
A4653	HMAC [198]	SHA-1	Key Sizes: Key Length = 112-65536 increment 8 MAC = *32–160 increment 8*	Message Authentication, KDF
A4653	HMAC [198]	SHA2-224	Key Sizes: Key Length = 112-65536 increment 8 MAC = *32–224 increment 8*	Message Authentication, KDF

CAVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A4653	HMAC [198]	SHA2-256	Key Sizes: Key Length = 112-65536 increment 8 MAC = *32–256 increment 8*	Message Authentication, KDF
A4653	HMAC [198]	SHA2-384	Key Sizes: Key Length = 112-65536 increment 8 MAC = *32–384 increment 8*	Message Authentication, KDF
A4653	HMAC [198]	SHA2-512	Key Sizes: Key Length = 112-65536 increment 8 MAC = *32–512 increment 8*	Message Authentication, KDF
A4653	HMAC [198]	SHA3-224	Key Sizes: Key Length = 112-65536 increment 8 MAC = *32–224 increment 8*	Message Authentication, KDF
A4653	HMAC [198]	SHA3-256	Key Sizes: Key Length = 112-65536 increment 8 MAC = *32–256 increment 8*	Message Authentication, KDF
A4653	HMAC [198]	SHA3-384	Key Sizes: Key Length = 112-65536 increment 8 MAC = *32–384 increment 8*	Message Authentication, KDF
A4653	HMAC [198]	SHA3-512	Key Sizes: Key Length = 112-65536 increment 8 MAC = *32–512 increment 8*	Message Authentication, KDF
A4653	KAS ECC [SP 800-56Arev3] per IG D.F Scenario 2 path (2)	Ephemeral Unified (Initiator, Responder), KPG, Full with twoStepKdf	P-224, P-256, P-384, P-521 HMAC SHA-1, SHA2-(224, 256, 384, 512), SHA3-(224, 256, 384, 512) Concatenation KDF mode: feedback, supports empty IV	Key Agreement Scheme provides between 112 and 256 bits of encryption strength

CAVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
A4653	KAS FFC [SP 800-56Arev3] per IG D.F Scenario 2 path (2)	Diffie-Hellman Ephemeral (Initiator, Responder), KPG, Full with twoStepKdf	FB, FC MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 HMAC SHA-1, SHA2-(224, 256, 384, 512), SHA3-(224, 256, 384, 512) Concatenation KDF mode: feedback, supports empty IV	Key Agreement Scheme provides between 112 and 200 bits of encryption strength
A4653	KDF SP800-108 [108]	Feedback	HMAC SHA-1, SHA2-(224, 256, 384, 512), SHA3-(224, 256, 384, 512) 8-bit counter after fixed input data	Key Based Key Derivation
A4653	RSA [186]	KeyGen	Modulus sizes: 2048, 3072, 4096	Asymmetric Key Generation
		PKCS1_v1.5	Modulus 2048 SHA2 (224, 256, 384, 512) Modulus 3072 SHA2 (224, 256, 384, 512) Modulus 4096 SHA2 (224, 256, 384, 512)	Digital Signature Generation
		PSS	Modulus 2048 SHA2 (224, 256, 384, 512) Modulus 3072 SHA2 (224, 256, 384, 512) Modulus 4096 SHA2 (224, 256, 384, 512)	Digital Signature Generation
		PKCS1_v1.5	Modulus 1024* SHA-1*, SHA2-(224, 256, 384, 512) Modulus 2048 SHA-1*, SHA2-(224, 256, 384, 512) Modulus 3072 SHA-1*, SHA2-(224, 256, 384, 512) Modulus 4096 SHA-1*, SHA2-(224, 256, 384, 512) * *Legacy Use*	Digital Signature Verification

CAVP Cert	Algorithm and Standard	Mode/Method	Description / Key Size(s) / Key Strength(s)	Use / Function
		PSS	Modulus 1024* SHA-1*, SHA2-(224, 256, 384, 512) Modulus 2048 SHA-1*, SHA2-(224, 256, 384, 512) Modulus 3072 SHA-1*, SHA2-(224, 256, 384, 512) Modulus 4096 SHA-1*, SHA2-(224, 256, 384, 512) * *Legacy Use*	Digital Signature Verification
A4653	SHS [180]	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512		Message Digest Generation
A4653	SHA-3 [202]	SHA3-224 SHA3-256 SHA3-384 SHA3-512 SHAKE128 SHAKE256		Hash Function

Table 5 – Vendor Affirmed Algorithms Allowed in the Approved Mode of Operation

Algorithm	Description / Key Size(s) / Key Strength(s)	Description
CKG	SP800-133rev3	SP800-133rev3 - Sections 4 and 5.1 Asymmetric signature key generation using unmodified DRBG output. CKG does not have certificate as IG D.H.states: "The module's validation certificate shall have a CKG entry only if the module is generating keys for the symmetric-key algorithms."
CKG	SP800-133rev3	SP800-133rev3 - Sections 4 and 5.2 Asymmetric key establishment key generation using unmodified DRBG output. CKG does not have certificate as IG D.H.states: "The module's validation certificate shall have a CKG entry only if the module is generating keys for the symmetric-key algorithms."

Note: The module does not have any non-approved but allowed algorithms.

Note: The module does not have any non-approved algorithms with no security claimed.

Table 6 – Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

Algorithm	Use/Function
AES EAX	Authentication/Encryption, non-approved algorithm
AES GCM 4K AES GCM 64K	256-bit encryption/decryption for 256k implementation
AES XCBC	Message Authentication, non-approved algorithm
DES	Encryption/Decryption, non-approved algorithm
DH	Key agreement; key establishment methodology provides less than 112 bits of encryption strength; non-compliant
DSA	SigGen using 2048/N=224 using SHA-1
ECC CDH	Key agreement; key establishment methodology provides less than 112 bits of encryption strength; non-compliant
EDDH	Curve 25519, Curve 448
HMAC	HMAC generation with key size less than 112 bits; non-compliant
HMAC-MD5	Non-approved algorithm
MD2, MD4, MD5	Message Digest, non-approved algorithm
RNG	FIPS 186-2 Random Number Generation
RSA	Key wrapping; key establishment methodology provides less than 112 bits of encryption strength; non-compliant
RSA	PKCS #1 v2.1 RSAES-OAEP encryption/decryption
RSA (key wrapping)	Key establishment methodology provides between 112 and 128 bit of encryption strength. Per IG D.G the module wraps data sent by the requesting application via an API call. Data being wrapped is unknown. PKCS non-approved padding
Triple-DES	Encryption/Decryption, non-approved algorithm.

Note: All the various AES mode (e.g., EAX, XCBC, XTS, etc.) use the same underlying AES implementation as the approved AES cert.

The Module implements the approved security functions listed in the table below.

Table 7 – Security Function Implementation (SFI)

Name	Type	Description	SF Properties	Algorithms/CAVP Cert
KAS-1	KAS	SP 800-56Arev3. KAS-ECC per IG D.F Scenario 2 path (2).	P-224, P-256, P-384, P-521 curves providing between 112 and 196 bits of encryption strength	KAS-ECC SP800-56Ar3/A4653
KAS-2	KAS	SP 800-56Arev3. KAS-FFC per IG D.F Scenario 2 path (2).	2024-bit, 3072-bit, 4096-bit, 6144-bit, 8192-bit keys providing between 112 and 200 bits of encryption strength	KAS-FFC SP800-56Ar3/A4653

2.5 Overall Security Design

1. The Module provides one operator role: Cryptographic Officer.
2. The Module does not provide any operator authentication.
3. An operator does not have access to any cryptographic services prior to assuming an authorized role.
4. The Module allows the operator to initiate power-up self-tests by power cycling power or reloading the Module into memory.
5. Pre-operational self-tests do not require any operator action.
6. Data outputs are inhibited during key generation, self-tests, zeroization, and error states. Because the logical interface is defined as the API of the Module and the API of the Module is single-threaded, key generation or zeroization must be complete before the API returns control to the calling application.
7. Status information does not contain SSPs or sensitive data that if misused could lead to a compromise of the Module.
8. There are no restrictions on which keys or SSPs are zeroized by the zeroization service.
9. The Module does not support concurrent operators.
10. The Module does not support a maintenance interface or role.
11. The Module does not support manual SSP establishment method.
12. The Module does not have any proprietary external input/output devices used for entry/output of data.
13. The Module does not enter or output plaintext SSPs, except to/from the calling application via API parameters. The module does not support the entry or output of encrypted SSPs.
14. The Module does not store any plaintext SSPs. SSPs provided to the Module by the calling processes are destroyed when released by the appropriate API function calls.
15. The Module does not output intermediate key values.
16. The Module does not provide bypass services or ports/interfaces.
17. AES GCM IV uniqueness: The AES GCM implementation meets Option 1 of IG C.H. The Module supports TLS 1.2 GCM Cipher Suites for TLS, as described in RFCs 5116, 5288 and 5289. The counter portion of the IV is set by the Module within its cryptographic boundary.

    When the nonce explicit (counter) part of the IV exhausts the maximum number of possible values for a given session key this condition triggers a handshake to establish a new encryption key per RFC 5246. During operational testing, the Module was tested against an independent version of TLS and found to behave correctly.

    AES GCM keys are zeroized when the Module is power-cycled and for each new TLS session, a new AES GCM key is established.
18. AES XTS is to be used only for storage purposes, per SP800-38E.

2.6 Rules of Operation

The Module shall be installed within the operating system confines and structures consistent with Mocana's operating environment specific documentation. For example: on most linux systems, this means that the libmss.so shared library will be installed in the file system in "/lib" or "/lib64", or it may be specified in the operating environment specific documentation to be installed in "/usr/local/lib" as appropriate for the target platform. The Module is automatically started when linked and loaded with an application using the cryptographic functions of the Module.

To update or replace the module, all SSPs shall first be zeroized and the calling application shall be closed. SSP zeroization is performed through the Key Destruction service that is described below. API calls will overwrite the memory occupied by the key information with zeros before that memory is de-allocated. If the calling application is terminated prior to zeroization, the Linux kernel overwrites the keys in physical memory before the physical memory is allocated to another process. The key zeroization process is performed in a sufficient time to prevent compromise of SSPs, taking only a few milliseconds.

The previous existing module files shall be removed prior to following the installation directions above, for the new module version.

(RSA)

The calling application of the Module must generate RSA key pairs of at least 2048 bits to operate in approved mode.

(ECC)

The calling application of the module must generate ECC keys using a P-Curve with a security strength of at least 112 bits to operate in the approved mode of operation.

(Random Number Generation)

The Module implements a CTR-based DRBG per SP800-90A for creation of symmetric and asymmetric keys.

The Module accepts input from entropy sources external to the cryptographic boundary for use as seed material for the Module's approved DRBGs. External entropy can be added via several APIs available to the cryptographic module client application.

The calling application of the Module shall use entropy sources that meet the security strength required for the random bit generation mechanism as shown in NIST SP 800-90A Table 3 (CTR_DRBG). A minimum of 384 bits of entropy must be provided by the calling application. The calling application shall provide full entropy for 256-bit keys. Due to the entropy being provided by an external source, the following caveat applies: There is no assurance of the minimum strength of generated SSPs (e.g., keys).

The Module performs DRBG health tests (Instantiate, Generate, Reseed) as defined in section 11.3 of SP800-90A.

(Key Management)

The application that uses the module is responsible for appropriate destruction and zeroization of the keys. The Module provides API calls for key allocation and destruction. These API calls overwrite the memory occupied by the key information with zeros before that memory is de-allocated. See Key Destruction Service below.

(Key/CSP Authorized Access and Use)

An authorized application has access to all key data generated during the operation of the Module.

(Key/CSP Storage)

Private and public keys are provided to the module by the calling process and are destroyed when released by the appropriate API function calls. The module does not perform persistent storage of keys.

(Key/CSP Zeroization)

The application is responsible for calling the appropriate destruction functions from the API. These functions overwrite the memory with zeros and de-allocate the memory. In case of abnormal termination, the Linux kernel overwrites the keys in physical memory before the physical memory is allocated to another process.

(Key Destruction Service)

A context structure is associated with every cryptographic algorithm available in the Module. Context structures hold sensitive information such as cryptographic keys. These context structures must be destroyed via respective API calls when the application software no longer needs to use a specific algorithm. This API call will zeroize all sensitive information before freeing the dynamically allocated memory. This will occur while the application process is still in memory, but no longer needs the specific algorithm, which protects the sensitive information from compromise. See the *Mocana Cryptographic API Reference* for additional information.

3 Cryptographic Module Interfaces

The Module's ports and associated defined logical interface categories are listed in Table 8. The module's logical interface (API) provides logical separation of the input and output interfaces.

Table 8 – Ports and Interfaces

Physical Port	Logical Interface	Data that passes over port/interface
General Purpose Computer	Data Input	Input parameters of API function calls
General Purpose Computer	Data Output	Output parameters of API function calls
General Purpose Computer	Control Input	API Function Calls
General Purpose Computer	Control Output	API Function Calls
General Purpose Computer	Status Output	For approved mode, function calls returning status information and return codes provided by API function calls
General Purpose Computer	Power	None

4 Roles, Services and Authentication

4.1 Assumption of Roles and Related Services

The module is Level 1 and does not implement any Authentication techniques.

The Module supports one operator role, Cryptographic Officer (CO). The cryptographic module does not support multiple concurrent users, bypass capability, or a maintenance role. The Cryptographic Officer role is implicitly identified by the service that is requested.

Table 9 – Roles, Service Commands, Input and Output

Roles	Service	Input	Output
CO	Asymmetric Encryption	Perform encryption of messages with input of RSA public key and message	Return encrypted message and status of OK or error condition
CO	Asymmetric Decryption	Perform decryption of messages with input of RSA private key and message	Return decrypted message and status of OK or error condition
CO	Asymmetric Key Generation	Generate key command with input of random number and key size for RSA, DSA, ECDSA, or EdDSA algorithm	Return of the Key (public and private) and status of OK or error condition

Roles	Service	Input	Output
CO	Asymmetric Key Verification	Perform key verification with input of (DSA, ECDSA, or EdDSA) Public/Private key pairs	Return of status of OK or error condition
CO	Digital Signature	Signature command with input of (RSA, DSA, ECDSA, or EdDSA) private key and message using RSA, DSA, ECDSA, or EdDSA algorithm	Return of the Digital Signature and status of OK or error condition
		Verify command with input of (RSA, DSA, ECDSA, or EdDSA) public key and signature to be verified by RSA, DSA, ECDSA, or EdDSA algorithm	Return of the Valid Signature Indicator (True or False) and status of OK or error condition
CO	Event Logging	Install callback function	Return status of algorithm is Approved or non-Approved
CO	Integrity Status	Integrity Command to perform Integrity Check with no input parameter	Return OK or error condition
CO	Key Agreement	Generate command with input of random number for DH or ECDH algorithm	Return secret shared symmetric key and status of OK or error condition
CO	Key Derivation (HMAC)	Derive key material using input psuedorandom key material to expand into additional key material	Return key material and status of OK or error condition
CO	Keyed Message Digest (CMAC)	Generate a keyed-hash message authentication code using input of AES key and message	Return keyed-hash with status of OK or error condition
CO	Keyed Message Digest (GMAC)	Generate a keyed-hash message authentication code using input of AES key and message	Return keyed-hash with status of OK or error condition
CO	Keyed Message Digest (HMAC)	Generate a keyed-hash message authentication code using input of HMAC key and message	Return keyed-hash with status of OK or error condition
CO	Message Authentication	Authenticate command with input of a message, None, and AAD using CMAC, GMAC, and HMAC algorithm	Return MAC and status of OK or error condition
CO	Message Digest	Hash command with input data of a message using a SHA-2 or SHA-3 algorithm	Return hash with status of OK or error condition
CO	Random Number Generation	Generate command with input of entropy	Return random number with OK or error condition

Roles	Service	Input	Output
CO	Self-tests	Execute command with input of list of CAST tests to be performed	Return OK or error condition
CO	Show module's versioning information	Version command with no input parameter	SW Version: xxx and Git repository name
CO	Show status	Status command with no input parameter	Overall status of tests run - either OK or error condition
CO	Symmetric Encryption	Encrypt command with AES, input data, input size, key, key size, mode of operation	Return the ciphertext and status of OK or error condition
CO	Symmetric Decryption	Decrypt command with AES, input data, input size, key, key size, mode of operation	Return the plaintext and status of OK or error condition
CO	Zeroize	Zeroize command, no input parameter	Return OK or an error condition

4.2 Services

All services implemented by the Module are listed in Table 10 and Table 11 below.

The SSPs modes of access shown in Table 10 are defined as:

- **G** = Generate: The Module generates or derives the SSP.
- **R** = Read: The SSP is read from the Module (e.g., the SSP is output).
- **W** = Write: The SSP is updated, imported, or written to the Module.
- **E** = Execute: The Module uses the SSP in performing a cryptographic operation.
- **Z** = Zeroize: The Module zeroizes the SSP

Table 10 – Approved Services

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Asymmetric Encryption and Decryption	Perform encryption and decryption of messages using public and private keys	RSA Encryption	RSA Public Key	CO	W, E	Approved
		RSA Decryption	RSA Private Key	CO	W, E	
Asymmetric Key Generation and Verification	Generate and verify Public / Private Asymmetric key pairs.	ECDSA Key Generation	ECDSA Private Key	CO	G, R	Approved
			ECDSA Public Key	CO	G, R	
		ECDSA Key Verification	ECDSA Private Key	CO	W, E	Approved
			ECDSA Public Key	CO	W, E	
Asymmetric Key Generation and Verification	Generate and verify Public / Private Asymmetric key pairs.	EdDSA Key Generation	EdDSA Private Key	CO	G, R	Approved
			EdDSA Public Key	CO	G, R	
		EdDSA Key Verification	EdDSA Private Key	CO	W, E	Approved

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
			EdDSA Public Key	CO	W, E	
Asymmetric Key Generation and Verification	Generate and verify Public / Private Asymmetric key pairs.	RSA Key Generation	RSA Private Key	CO	W, E	Approved
			RSA Public Key	CO	G, R	
Asymmetric Key Generation and Verification	Generate and verify Public / Private Asymmetric key pairs.	DSA Key Generation	DSA Private Key	CO	G, R	Approved
			DSA Public Key	CO	G, R	
		DSA Key Verification	DSA Private Key	CO	W, E	Approved
			DSA Public Key	CO	W, E	
Digital Signature	Perform digital signature generation and digital signature verification functions.	DSA Signature Generation	DSA Private Key	CO	W, E	Approved
		DSA Signature Verification	DSA Public Key	CO	W, E	
Digital Signature	Perform digital signature generation and digital signature verification functions.	ECDSA Signature Generation	ECDSA Private Key	CO	W, E	Approved
		ECDSA Signature Verification	ECDSA Public Key	CO	W, E	
Digital Signature	Perform digital signature generation and digital signature verification functions.	EdDSA Signature Generation	EdDSA Private Key	CO	W, E	Approved
		EdDSA Signature Verification	EdDSA Public Key	CO	W, E	

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Digital Signature	Perform digital signature generation and digital signature verification functions.	RSA Signature Generation	RSA Private Key	CO	W, E	Approved
		RSA Signature Verification	RSA Public Key	CO	W, E	
Integrity Status	Perform integrity test and return the status	N/A	N/A	CO	E	N/A
Key Agreement	Generate a secret key to be used between two or more parties based on the key-agreement protocol	DH Key Generation	FCC Private Key	CO	R, G, E	Approved
			FFC Public Key	CO	R, G, E	
		DH Key Exchange	FCC Private Key	CO	W, E	
			FFC Public Key	CO	W, E	
		ECC CDH Key Generation	ECC CDH Private Key	CO	R, G, E	
			ECC CDH Public Key	CO	R, G, E	
		ECC CDH Key Exchange	ECC CDH Private Key	CO	W, E	
			ECC CDH Public Key	CO	W, E	
Key Agreement	Generate a secret key to be used between two or more parties based on the key-agreement protocol	DH Shared Secret	DH Shared Secret	CO	R, G	Approved

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Key Agreement	Generate a secret key to be used between two or more parties based on the key-agreement protocol	ECC CDH Shared Secret	ECC CDH Shared Secret	CO	R, G	Approved
Key Derivation (HMAC)	Extract input key material and expand into additional keys	HMAC-SHA-1, HMAC-SHA2-224, -256, -384, -512, HMAC-SHA3-224, -256, -384, -512, Shake128, Shake256	HMAC-KDF psuedorandom Key	CO	W, E	Approved
Keyed Message Digest (CMAC)	Generate a keyed-hash message authentication code	AES-CMAC	AES Key	CO	W, E	Approved
Keyed Message Digest (GMAC)	Generate a keyed-hash message authentication code	AES-GMAC	AES Key	CO	W, E	Approved
Keyed Message Digest (HMAC)	Generate a keyed-hash message authentication code	HMAC-SHA-1, HMAC-SHA2-224, -256, -384, -512, HMAC-SHA3-224, -256, -384, -512, Shake128, Shake256	HMAC Key	CO	W, E	Approved
Message Hash	Generate a SHA-1, SHA-2, or SHA-3 message digest	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	N/A	CO	N/A	Approved

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
Random Number Generation	Generate and Re-seed random numbers.	AES-CTR DRBG Generation	Seed, Nonce and DRBG Values	CO	R	Approved
		AES-CTR DRBG Re-seed	DRBG Entropy Input	CO	W	Approved
Self-tests	Initiate self-tests (Software Integrity Check, DRBG KAT, SHA-256 KAT, HMAC-SHA-256 KAT)	N/A	N/A	CO	R, E	Approved
Show Status	Return status of the module state, exit codes, kernel log (dmesg).	N/A	N/A	CO	E	N/A
Show Version	Return module version information	N/A	N/A	CO	E	N/A
Symmetric Encryption/ Decryption	Perform encryption and decryption on a block of data using the shared key	AES-CBC, AES-CTR, AES-ECB, AES-CFB, AES-OFB, AES-XTS Encryption	AES Keys	CO	W, E	Approved
		AES-CBC, AES-CTR, AES-ECB, AES-CFB, AES-OFB, AES-XTS Decryption	AES Keys	CO	W, E	
Symmetric Encryption/ Decryption with Message Digest (CCM)	Perform encryption and decryption on a block of data using shared key and message authentication code	AES-CCM	AES Keys	CO	W, E	Approved
Symmetric Encryption/ Decryption with Message Digest (GCM)	Perform encryption and decryption on a block of data using shared key and message authentication code	AES-GCM	AES Keys	CO	W, E	Approved
Zeroize	Destroys all SSPs	N/A	All SSPs	CO	Z	Approved

Table 11 – Non-Approved Services

Service	Description	Algorithm Accessed	Role	Indicator
Digital Signature	SigGen using 2048/N=224 using SHA-1	DSA	CO	Non-Approved
Key Agreement	Generate a secret key to be used between 2 or more parties based on the key-agreement protocol	EDDH Key Generation/Key Exchange	CO	Non-Approved
Key Agreement	Key agreement; key establishment methodology provides less than 112 bit of encryption strength; non-compliant	DH, ECC CDH	CO	Non-Approved
Keyed Message Digest	AES GCM encryption and decryption for 256 implementations	AES-GCM 4k / GMAC 4k AES-GCM 64k / GMAC 64k	CO	Non-Approved
Keyed Message Digest	HMAC generation with key size less than 112 bit; non-compliant	HMAC-MD5	CO	Non-Approved
Key Wrapping	Key wrapping; key establishment methodology non-compliant	RSA Encryption/Decryption	CO	Non-Approved
Message Digest	Generate an MD2, MD4, or MD5 message digest	MD2, MD4, MD5	CO	Non-Approved
Random Number Generation	FIPS 186-2 Random Number Generation	RNG	CO	Non-Approved
Symmetric Encryption/ Decryption	Non Approved Algorithm	AES-EAX, AES-XCBC, DES	CO	Non-Approved
Symmetric Encryption/ Decryption	Non Approved Algorithm	Triple-DES	CO	Non-Approved

5 Software Security

The Module is composed of the following single software component packaged as a shared object (also known as a shared library).

- libmss.so – This is the shared library that contains the cryptographic module code, data, and constants.
- libmss.so.sig – This is the Integrity check signature file that contains an HMAC-SHA-256 of the crypto module.
- mssp.bin – This is the POST status file. It contains persistent results of the first-run of POST. See POST-integrity-check algorithm self-test.

Integrity Check:
The shared library is protected with the authentication technique HMAC-SHA-256 described in
.

The HMAC for the shared library is calculated during the manufacturing (build) process of the shared library. This HMAC value is stored either within the resulting "libmss.so" shared object or as a separate "libmss.so.sig" file dependent upon the development tools and target operating systems constraints.

During the load of the shared object, the integrity check of the library code and constants occurs in the module startup function. It verifies the integrity of the shared library by executing the HMAC-SHA-256 fingerprint algorithm on the libmss.so file and comparing the result with the signature. This integrity check is performed as part of the function FIPS_powerupSelfTest(). This function is called automatically by the host O/S upon loading the shared object into memory as shown below.

```
#ifdef __ENABLE_MOCANA_FIPS_LIB_CONSTRUCTOR__
static void FIPS_constructor() __attribute__((constructor));
void FIPS_constructor()
{
        FIPS_powerupSelfTest();
}
#endif
```

The operator can also explicitly initiate the integrity test on demand by calling the API function: FIPS_StartupSelftestIntegrity(void).

Pre-Integrity Check Power up Self Tests:
To fulfill the Implementation Guidance 10.3.A, the algorithms used to perform the integrity check are executed before the module integrity check. These are lists in more detail in Table 14. The following algorithm KAT tests are performed in the FIPS_powerupSelfTest implementation before the integrity check:

- SHA-256
- HMAC-SHA-256

Note: Although not used by the integrity check, the DRBG KAT test is also run during this initial FIPS_powerupSelfTest().

Post-Integrity Check Algorithm Self Tests:
As a startup optimization, the self-tests for all other approved algorithms are implemented as Conditional self-tests (see Table 16). These CASTs are performed on-demand. The first time a FIPS algorithm is used, the CAST test for that algorithm is tested before the FIPS algorithm processing is done.

After the CAST test for each algorithm is run, the status of each algorithm CAST test is stored as an array within the FIPS module so that subsequent usage of each algorithm can detect the CAST test for that module has completed successfully and no longer needs to run. If the CAST test fails for an algorithm, then a global CAST status is set to a failure code and subsequent calls to FIPS cryptographic functions will fail.

The FIPS module provides an API to allow the Cryptographic Officer to initiate the FIPS powerup self-tests on demand. This includes all CAST tests. Additionally, an API is provided to enable the Cryptographic Officer to persist the state of these self-tests. The state of the self-test is stored in the mssp.bin file. During Module startup, the presence of the previously saved state of self-tests is detected. If the saved-state is more recent than the O/S boot time, then the saved state information for the self-tests is loaded as the starting state. This will enable a cryptographic officer to execute a single run of all conditional self-tests after each boot of the operating system, and subsequent applications using the same Module will not have to re-run the conditional self-tests. Note that the Pre-Integrity Check Power up Self Tests, and Integrity Check described above are always run within each instance of the Module.

6 Operational Environment

The Module has a modifiable operational environment under the FIPS 140-3 definitions. The tested operational environments - Software are listed in Table 2 above. In addition, Mocana claims that the Module can be ported on the Operational Environment(s) listed in Table 3; no statement is made regarding the correct operation of the Module on the Vendor Affirmed Operational Environments.

7 Physical Security

The FIPS 140-3 Physical Security requirements are not applicable because the Mocana Cryptographic Suite B Module is software only.

8 Non-Invasive Security

The Module does not implement any mitigation method against non-invasive attacks.

9 Sensitive Security Parameter (SSP) Management

The SSPs access methods are described in Table 12 below:

Table 12 – SSP Management Methods

Method	Description
G1	Generated external (logical) to the Module and input from calling application
G2	Derived from the DRBG input per SP800-90Ar1
G3	Derived from the DRBG output
G4	FIPS 186-4 compliant DSA key generation, using the internal CAVP validated DRBG
G5	FIPS 186-4 compliant RSA key generation, using the internal CAVP validated DRBG
G6	FIPS 186-4 compliant ECDSA key generation, using the internal CAVP validated DRBG.
G7	FIPS 186-5 compliant EdDSA key generation, using the internal CAVP validated DRBG.
G8	Diffie-Hellman shared secret generation using the internal CAVP validated 56Arev3 protocol

Method	Description
G9	EC Diffie-Hellman shared secret generation using the internal CAVP validated 56Arev3 protocol
S1	Only stored in volatile memory (RAM).
E1	Input in plaintext (client key)
E2	Output in plaintext
E3	Output in plaintext public key
Z1	Zeroized by the Key destruction service by overwriting with a fixed pattern of zeros.

All SSPs used by the Module are described in this section. All usage of these SSPs by the Module is described in the services detailed in Section 4.2.

Table 13 – SSPs

Key/SSP Name/Type	Strength	Security Function and Cert. Number	Generation	Import /Export	Establishment	Storage	Zeroization	Use & Related keys
FFC Private Key	112 to 200 bits	KAS	G3	N/A	N/A	S1	Z1	Used to derive the secret key during DH key agreement protocol
DH Shared Secret	112 to 200 bits	KAS	N/A	E2	G8	S1	Z1	Shared secret computation established as part of DH key agreement scheme
FFC Public Key	112 to 200 bits	KAS	G3	E1, E2	N/A	S1	Z1	Used to derive the secret key during DH key agreement protocol
ECC CDH Private Key	112 to 256 bits	KAS	G3	E1	N/A	S1	Z1	Used to derive the secret session key during ECC CDH key agreement protocol
ECC CDH Shared Secret	112 to 256 bits	KAS	N/A	E1	G9	S1	Z1	Shared secret computation
ECC CDH Public Key	112 to 256 bits	KAS	G3	E1, E2	N/A	S1	Z1	Used to derive the secret session key during ECC CDH key agreement protocol
DRBG Entropy Input	384 bits for 384 entropy bits	CTR DRBG	N/A	E1	N/A	S1	Z1	Used to seed the DRBG for key generation

Key/SSP Name/Type	Strength	Security Function and Cert. Number	Gener-ation	Import /Export	Establish-ment	Storage	Zeroiza -tion	Use & Related keys
Seed, Nonce and DRBG values	384 bits for 384 entropy bits	CTR DRBG	G2	N/A	N/A	S1	Z1	Used by the DRBG to generate random bits
RSA Private Key	112 to 256 bits	RSA	G1, G5	E1, E2	N/A	S1	Z1	Used to create RSA digital signatures
RSA Public Key	112 to 256 bits	RSA	G5	E1, E3	N/A	S1	Z1	Used to verify RSA signatures
DSA Private Key	112 to 128 bits	DSA	G1, G4	E1, E2	N/A	S1	Z1	Used to create DSA digital signatures
DSA Public Key	112 to 128 bits	DSA	G4	E1, E3	N/A	S1	Z1	Used to verify DSA signatures
ECDSA Private Key	112 to 256 bits	ECDSA	G1, G6	E1, E2	N/A	S1	Z1	Used to create ECDSA digital signatures
ECDSA Public Key	112 to 256 bits	ECDSA	G6	E1, E3	N/A	S1	Z1	Used to verify ECDSA signatures
EdDSA Private Key	128 to 224 bits	EdDSA	G1, G7	E1, E2	N/A	S1	Z1	Used to create EdDSA digital signatures
EdDSA Public Key	128 to 224 bits	EdDSA	G7	E1, E3	N/A	S1	Z1	Used to verify EdDSA signatures
AES Keys	128 to 256 bits	AES	G1	E1	N/A	S1	Z1	Used during AES encryption, decryption, CMAC and GMAC operations
HMAC Key	112 to 256 bits	HMAC	G1	E1	N/A	S1	Z1	Used during HMAC-SHA-1, HMAC-SHA-224, 256, 384, 512, HMAC-SHA3-224, 256, 384, 512 operations
HMAC-KDF Psuedorandom Key	112 to 256 bits	HMAC-KDF	G1	E1	N/A	S1	Z1	Used in deriving other keys per SP 800-108 with HMAC-SHA2-224, 256, 384, 512, HMAC-SHA3-224, 256, 384, 512 operations

9.1 DRBG Entropy Source

Table 14 – Non-Deterministic Random Number Generation Specification

Entropy Source	Minimum number of bits of entropy	Details
External Entropy Source (API)	384	The calling application of the Module shall use entropy sources that meet the security strength required for the random bit generation mechanism. A minimum of 384 bits of entropy must be provided by the calling application. The module does not exercise control over the amount or quality of the entropy that is provided to it.

10 Self-Tests

The Module performs self-tests to ensure the proper operation of the Module. Per FIPS 140-3 these are categorized as either pre-operational self-tests or conditional self-tests.

10.1 Pre-Operational Self-Tests

Security Level 1 - Pre-operational self-tests are available on demand by power cycling or reloading the Module into memory. The Module is available to perform services only after successfully completing the pre-operational self-tests.

The Module performs the following pre-operational self-tests:

Table 15 – Pre-Operational Self-Test

Security Function	Method	Description	Error state
DRBG AES-256 CTR-DRBG	KAT	Critical Function Test (Instantiation, Generation, Reseed): This CAST is performed before the Software Integrity Check. This is the first KAT performed	ES2
HMAC-SHA-256 (Cert. A4653)	KAT	Software Integrity on shared lib .so file, result is compared against the hash value in the signature .sig file.	ES1

Note: The module does not perform pre-operational bypass tests. Bypass is not implemented.

10.2 Conditional Self-Tests

The Module performs the following conditional self-tests:

Table 16 – Conditional Self-Tests

Security Function	Method	Description	Error state
AES-ECB, CBC, OFB, CFB, CTR	KAT	AES encryption and decryption KAT – Inclusive to AES, CBC, CFB, CTR, ECB, OFB with 256-bit key	ES2
AES-XTS	KAT	AES encryption and decryption KAT with 128-bit key	ES2
AES-CCM, CMAC	KAT	AES encryption and decryption with CBC-MAC authentication with 128-bit key	ES2
AES-GCM, GMAC	KAT	AES encryption and decryption with GMAC authentication with 128-bit key – for both 4k and 64k versions	ES2
DH	KAT	Diffie Hellman key exchange-agreement protocol tested by comparing the shared secret key followed by testing the key derivation function used in deriving the keying material.	ES2
DRBG: AES-256 CTR-DRBG	KAT	Critical Function Test (Instantiation, Generation, Reseed): This CAST is performed before the Software Integrity Check. This is the first KAT performed	ES2
DSA	KAT	Digital Signature Algorithm tested using a public-private key pair to perform a Signature and verify the resultant Signature against a known value.	ES2
DSA	PCT	Digital Signature Algorithm tested using a public-private key pair to perform a Signature and verify the resultant Signature	ES3
ECC CDH	KAT	Elliptic Curve Cryptography Co-factor Diffie Hellman key exchange-agreement protocol tested by comparing the shared secret key by testing the key derivation function used in deriving the keying material.	ES2
ECDSA	KAT	Elliptic Curve Digital Signature Algorithm tested using a public-private key pair to perform a Signature and verify the resultant Signature against a known value.	ES2
ECDSA	PCT	ECDSA P-224 Pairwise Consistency Test	ES3
EdDSA	KAT	Edwards Curve Digital Signature Algorithm tested using a public-private key pair to perform a Signature and verify the resultant Signature against a known value.	ES2
EdDSA	PCT	EdDSA Pairwise Consistency Test	ES3
HMAC-SHA2	KAT	Calculate a cryptographic hash-based authentication on data for SHA-1, SHA-224, SHA-256*, SHA-384, SHA-512	ES2

		* HMAC-SHA-256 instantiation: This CAST is performed before the Software Integrity Check. This is the third KAT performed.	
HMAC-SHA3	KAT	Calculate a cryptographic hash-based authentication on data for SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	ES2
HMAC-KDF-SHA	KAT	Extract and expand the input key into additional keys using SHA-1, SHA-224, -256, -384, -512	ES2
HMAC-KDF-SHA3	KAT	Extract and expand the input key into additional keys using SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	ES2
SHA	KAT	Calculate a cryptographic hash function on the data for SHA-1, SHA-224, SHA-256*, SHA-384, SHA-512 * SHA-256 instantiation: This CAST is performed before the Software Integrity Check. This is the second KAT performed.	ES2
SHA3	KAT	Calculate a cryptographic hash function on the data for SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	ES2
Triple-DES CBC	KAT	3-key Triple-DES encryption and decryption tested using a 192-bit key	ES2
RSA	KAT	2048-bit RSA PKCSv1.5 with SHA-224 - no hash - Signature Generation and Signature Verification	ES2
RSA	PCT	RSA Pairwise Consistency Test Signature Generation and Signature Verification	ES3
RSA	KAT	3072-bit RSA Encryption and Decryption per IG D.G.	ES2

10.3 Error States and Indicators

The self-tests error states and status indicator are described in table below:

Table 17 – Error States and Indicators

Error state	Description	Indicator
ES1	The Module fails the software integrity pre-operational self-test.	The Module enters the disable Crypto Module "Error State" and outputs status of ERR_FIPS_INTEGRITY_FAIL, otherwise it indicates successful completion by outputting the OK status.
ES2	The Module fails the software CAST test with a specified error number.	The Module enters the disable Crypto Module "Error State" and outputs a specific error status; otherwise, it indicates successful completion by enable Crypto Module with OK status.
ES3	Module fails all other self-tests not listed above.	The Module enters the disable Crypto Module "Error State" and outputs a specific error status; otherwise, it indicates successful completion by enable Crypto Module with OK status.

11 Life-Cycle Assurance

Installation is performed by placing the module in the target file system during the OEM or ISV's manufacturing process. The module initialization is performed automatically by the operating system's loader when a calling application is loaded into memory. Operation of the module is controlled by the calling application's use of the module's API functions.

There is no specific guidance for Administrator or non-Administrators. The module is provided with supporting documentation which includes an API Reference document and Operating Environment document.

11.1 (Cryptographic Officer Guidance)

The Cryptographic Officer will install the Module and associated signature of the Module into the proper location within the computer system. For example, the shared memory library and signature file may be installed in the /usr/local/lib directory, which is protected by Linux access control mechanisms. The Module is protected from modification by the integrity self-test performed during start-up. The Module is initialized by the operating system upon loading the Module into memory for use by calling applications.

The Module must be operated in the approved mode to ensure that FIPS 140-3 validated cryptographic algorithms and security functions are used.

12 Mitigation of Other Attacks

The Module does not implement any mitigation method against other attacks beyond the requirements for FIPS 140-3 Level 1 cryptographic modules.

13 References and Definitions

The following standards are referred to in this Security Policy.

Table 18 – References

Abbreviation	Full Specification Name
[FIPS140-3]	*Security Requirements for Cryptographic Modules*, March 22, 2019
[ISO19790]	*International Standard, ISO/IEC 19790, Information technology — Security techniques — Test requirements for cryptographic modules, Third edition, March 2017*
[ISO24759]	*International Standard, ISO/IEC 24759, Information technology — Security techniques — Test requirements for cryptographic modules, Second and Corrected version, 15 December 2015*
[IG]	*Implementation Guidance for FIPS PUB 140-3 and the Cryptographic Module Validation Program, 10/07/22*
[108]	*NIST Special Publication 800-108, Recommendation for Key Derivation Using Pseudorandom Functions (Revised), October 2009*
[131A]	*Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths, Revision 2, March 2019*

Abbreviation	Full Specification Name
[132]	*NIST Special Publication 800-132, Recommendation for Password-Based Key Derivation, Part 1: Storage Applications, December 2010*
[133]	*NIST Special Publication 800-133, Recommendation for Cryptographic Key Generation, Revision 2, June 2020*
[135]	*National Institute of Standards and Technology, Recommendation for Existing Application-Specific Key Derivation Functions, Special Publication 800-135rev1, December 2011.*
[186]	*National Institute of Standards and Technology, Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-4, July 2013.*
[197]	*National Institute of Standards and Technology, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, November 26, 2001*
[198]	*National Institute of Standards and Technology, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication 198-1, July, 2008*
[180]	*National Institute of Standards and Technology, Secure Hash Standard, Federal Information Processing Standards Publication 180-4, August, 2015*
[202]	*FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, FIPS PUB 202, August 2015*
[38A]	*National Institute of Standards and Technology, Recommendation for Block Cipher Modes of Operation, Methods and Techniques, Special Publication 800-38A, December 2001*
[38B]	*National Institute of Standards and Technology, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, Special Publication 800-38B, May 2005*
[38C]	*National Institute of Standards and Technology, Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, Special Publication 800-38C, May 2004*
[38D]	*National Institute of Standards and Technology, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, Special Publication 800-38D, November 2007*
[38E]	*National Institute of Standards and Technology, Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices, Special Publication 800-38E, January 2010*
[38F]	*National Institute of Standards and Technology, Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, Special Publication 800-38F, December 2012*
[56Ar3]	*NIST Special Publication 800-56A Revision 3, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, April 2018*
[56Br2]	*NIST Special Publication 800-56B Revision 2, Recommendation for Pair-Wise Key Establishment Schemes Using Finite Field Cryptography, March 2019*

Abbreviation	Full Specification Name
[56Cr2]	*NIST Special Publication 800-56C Revision 2, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, August 2020*
[67]	*National Institute of Standards and Technology, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, Special Publication 800-67, May 2004*
[90A]	*National Institute of Standards and Technology, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, Special Publication 800-90A, Revision 1, June 2015.*
[90B]	*National Institute of Standards and Technology, Recommendation for the Entropy Sources Used for Random Bit Generation, Special Publication 800-90B, January 2018.*

Table 19 – Acronyms and Definitions

Acronym	Definition
AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard New Instructions
API	Application Program Interface
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DH	Diffie-Hellman
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECC CDH	Elliptic Curve Cryptography Cofactor Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EdDSA	Edwards-curve Digital Signature Algorithm
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FIPS	Federal Information Processing Standard
GCM	Galois Counter Mode
HMAC	Hash Message Authentication Code
IG	Implementation Guidance

Acronym	Definition
KAT	Known Answer Test
KDF	Key Derivation Function
KVM	Kernel-based Virtual Machine
PAA	Processor Algorithm Acceleration
PCT	Pair-wise Consistency Test
RNG	Random Number Generator
RSA	Rivest, Shamir and Adleman Algorithm
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SO	Shared Object
TDES	Triple-DES
XTS	XEX-based Tweaked-codebook mode with ciphertext Stealing