# Mocana Cryptographic Loadable Kernel Module

Version 7.0.0f

# Non-Proprietary FIPS 140-3 Security Policy

Document Version: 1.1

**Date: August 01, 2024**

*DigiCert, Inc.*

2801 North Thanksgiving Way

Suite 500

Lehi, UT 84043

+1 800-896-7973

# Table of Contents

1 General ..... 4

2 Cryptographic Module Specification ..... 5

2.1 Operational Environment ..... 5

2.2 Cryptographic Boundary ..... 6

2.3 Modes of Operation ..... 7

2.3.1 Configuration of the Approved Mode of Operation ..... 8

2.3.2 Configuration of the Non-Approved Modes of Operation ..... 8

2.4 Security Functions ..... 8

2.5 Overall Security Design ..... 10

2.6 Rules of Operation ..... 11

3 Cryptographic Module Interfaces ..... 13

4 Roles, Services and Authentication ..... 13

4.1 Assumption of Roles and Related Services ..... 13

4.2 Services ..... 14

5 Software Security ..... 17

6 Operational Environment ..... 19

7 Physical Security ..... 19

8 Non-Invasive Security ..... 19

9 Sensitive Security Parameter (SSP) Management ..... 19

9.1 DRBG Entropy Source ..... 21

10 Self-Tests ..... 21

10.1 Pre-Operational Self-Tests ..... 21

10.2 Conditional Self-Tests ..... 21

10.3 Error States and Indicators ..... 23

11 Life-Cycle Assurance ..... 23

11.1 (Cryptographic Officer Guidance) ..... 23

12 Mitigation of Other Attacks ..... 23

13 References and Definitions ..... 23

## List of Tables

Table 1 – Security Level of Security Requirements ........ 4
Table 2 – Tested Operational Environments - Software ........ 5
Table 3 – Vendor Affirmed Operational Environment ........ 5
Table 4 – Approved Algorithms ........ 8
Table 5 – Non-Approved Algorithms Not Allowed in the Approved Mode of Operation ........ 10
Table 6 – Ports and Interfaces ........ 13
Table 7 – Roles, Service Commands, Input and Output ........ 14
Table 8 – Approved Services ........ 15
Table 9– Non-Approved Services ........ 16
Table 10 – SSP Management Methods ........ 19
Table 11– SSPs ........ 20
Table 12 – Non-Deterministic Random Number Generation Specification ........ 21
Table 13 – Pre-Operational Self-Test ........ 21
Table 14 – Conditional Self-Tests ........ 22
Table 15 – Error States and Indicators ........ 23
Table 16 – References ........ 24
Table 17 – Acronyms and Definitions ........ 25

## List of Figures

Figure 1 – Cryptographic Module Interface Design ........ 6
Figure 2 – Logical Object ........ 7
Figure 3 - Code Example for Self-Test ........ 18

## 1 General

This document defines the non-proprietary Security Policy for the Mocana Cryptographic Loadable Kernel Module version 7.0.0f 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 only, 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 kernel object (KO), moc_crypto.ko and associated signature file.

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 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 (64-bit) | Xerox Explorer 6.5 | Intel Atom E3950 | without PAA | NA |
| 2 | Yocto Linux 3.1 (64-bit) | Xerox Explorer 8.0 | Intel Atom x6413E | without PAA | NA |
| 3 | Yocto Linux 3.1 (64-bit) | Xerox Alexandra Platform | ARM Cortex A53 | 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 |

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 kernel module, (moc_crypto.ko) and the integrity check signature file (moc_crypto.ko.sig). 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.

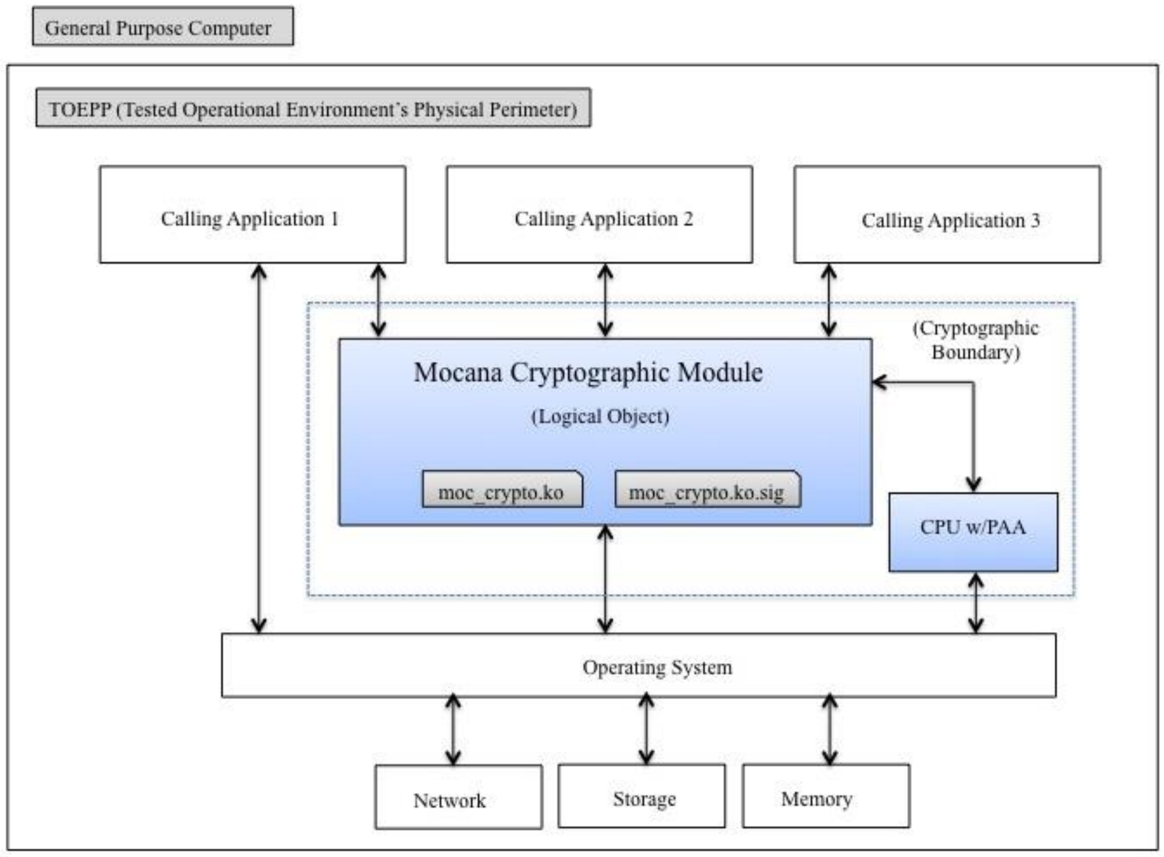


**Figure 1 – Cryptographic Module Interface Design**

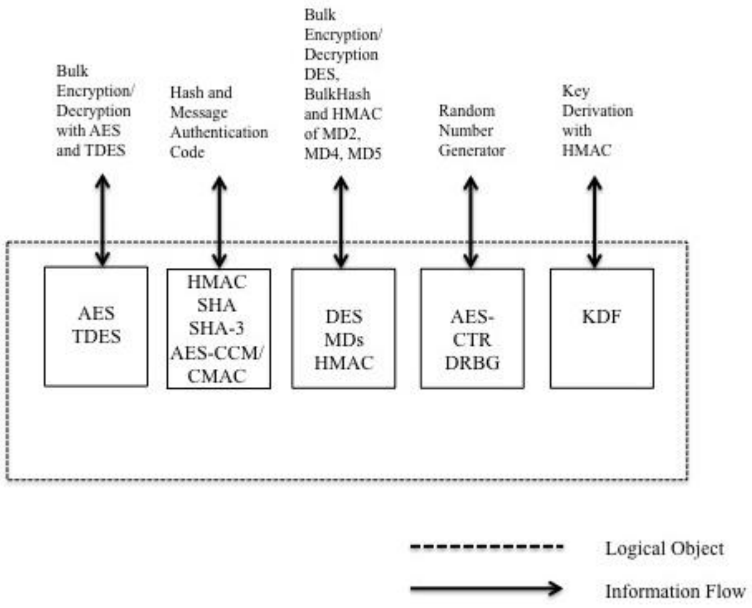

Bulk Encryption/ Decryption with AES and TDES
Hash and Message Authentication Code
Bulk Encryption/ Decryption DES, BulkHash and HMAC of MD2, MD4, MD5
Random Number Generator
Key Derivation with HMAC
AES TDES
HMAC SHA SHA-3 AES-CCM/ CMAC
DES MDs HMAC
AES-CTR DRBG
KDF
Logical Object
Information Flow

**Figure 2 – Logical Object**

## 2.3 Modes of Operation

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

1. Approved mode (the Approved mode of operation): 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 Mode 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 |
|---|---|---|---|---|
| A845 | AES [197] | CBC [38A] | Key Sizes: 128, 192, 256 | Encrypt, Decrypt |
| A845 | AES [197] | CCM [38C] | Key Sizes: 128, 192, 256<br>Tag Len: 32, 48, 64, 80, 96, 112, 128 | Authenticated Encrypt, Authenticated Decrypt |
| A845 | AES [197] | CFB128 [38A] | Key Sizes: 128, 192, 256 | Encrypt, Decrypt |
| A845 | AES [197] | CMAC [38B] | Key Sizes: 128, 192, 256<br>MAC Len: 32-128 Increment 8; Message Len: 0-65536 Increment 8 | Message Authentication |
| A845 | AES [197] | CTR [38A] | Key Sizes: 128, 192, 256 | Encrypt, Decrypt |
| A845 | AES [197] | ECB [38A] | Key Sizes: 128, 192, 256 | Encrypt, Decrypt |
| A845 | AES [197] | OFB [38A] | Key Sizes: 128, 192, 256 | Encrypt, Decrypt |
| A845 | AES [197] | XTS[1] [38E] | Key Sizes: 128, 256 | Encrypt, Decrypt |
| A846<br>A847 | AES [197] | GCM/GMAC [38D] | Key Sizes: 128, 192, 256<br>Tag Len: 32, 64, 96, 104, 112, 120, 128 | Authenticated Encrypt, Authenticated Decrypt |
| A845 | CTR_DRBG [90A] | CTR | Use_df (default), No_df<br>AES-128, AES-192, AES-256 | Deterministic Random Number Generation |
| A845 | HMAC [198] | SHA-1 | Key Sizes:<br>Key Length = 112-65536 increment 8<br>MAC = *32–160 increment 8* | Message Authentication, KDF |

[1] The XTS algorithm implementation includes a check to ensure Key_1 ≠ Key_2

| CAVP Cert | Algorithm and Standard | Mode/Method | Description / Key Size(s) / Key Strength(s) | Use / Function |
|---|---|---|---|---|
| A845 | HMAC [198] | SHA-224 | Key Sizes:<br>Key Length = 112-65536 increment 8<br>MAC = *32–224 increment 8* | Message Authentication, KDF |
| A845 | HMAC [198] | SHA-256 | Key Sizes:<br>Key Length = 112-65536 increment 8<br>MAC = *32–256 increment 8* | Message Authentication, KDF |
| A845 | HMAC [198] | SHA-384 | Key Sizes:<br>Key Length = 112-65536 increment 8<br>MAC = *32–384 increment 8* | Message Authentication, KDF |
| A845 | HMAC [198] | SHA-512 | Key Sizes:<br>Key Length = 112-65536 increment 8<br>MAC = *32–512 increment 8* | Message Authentication, KDF |
| A845 | HMAC [198] | SHA3-224 | Key Sizes:<br>Key Length = 112-65536 increment 8<br>MAC = *32–224 increment 8* | Message Authentication, KDF |
| A845 | HMAC [198] | SHA3-256 | Key Sizes:<br>Key Length = 112-65536 increment 8<br>MAC = *32–256 increment 8* | Message Authentication, KDF |
| A845 | HMAC [198] | SHA3-384 | Key Sizes:<br>Key Length = 112-65536 increment 8<br>MAC = *32–384 increment 8* | Message Authentication, KDF |
| A845 | HMAC [198] | SHA3-512 | Key Sizes:<br>Key Length = 112-65536 increment 8<br>MAC = *32–512 increment 8* | Message Authentication, KDF |
| A845 | KBKDF [108] | Feedback | HMAC-SHA-1, SHA2-(224, 256, 384, 512, SHA3-(224, 256, 384, 512)<br>8-bit counter after fixed input data | Key Based Key Derivation |
| A845 | SHS [180] | SHA-1, SHA2-224, SHA-256, SHA-384, SHA-512 | | Message Digest Generation |

| CAVP Cert | Algorithm and Standard | Mode/Method | Description / Key Size(s) / Key Strength(s) | Use / Function |
|---|---|---|---|---|
| A845 | SHA-3 [202] | SHA3-224<br>SHA3-256<br>SHA3-384<br>SHA3-512<br>SHAKE 128<br>SHAKE 256 | | Hash Function |

Note: Other algorithms were tested but not implemented by this module.

Note: The module does not have any vendor affirmed algorithms allowed in the approved mode of operation.

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 5 – Non-Approved Algorithms Not Allowed in the Approved Mode of Operation**

| Algorithm | Use/Function |
|---|---|
| AES EAX | Authentication/Encryption, non-approved algorithm |
| AES GCM/GMAC 4K<br>AES GCM/GMAC 64K | 256-bit decryption for 256k implementation |
| AES XCBC | Message Authentication, non-approved algorithm |
| DES | Encryption/Decryption, non-approved algorithm |
| 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 |
| Triple-DES 2-key | Encryption/Decryption, non-approved algorithm |
| Triple-DES 3-key | Encryption/Decryption, Disallowed after 2023 |

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

### 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 Digicert's operating environment specific documentation. For example: on most linux systems, this means that the moc_crypto.ko kernel module 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.

**(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: When operated in approved mode. 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 FIPS defined logical interface categories are listed in Table 6 – Ports and Interfaces. The module's logical interface (API) provides logical separation of the input and output interfaces.

**Table 6 – 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 7 – Roles, Service Commands, Input and Output**

| Roles | Service | Input | Output |
|---|---|---|---|
| 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 | 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 |
| CO | Self-tests | Execute command with input of list 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 8 and Table 9 below.

The SSPs modes of access shown in Table 8 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 8 – Approved Services**

| Service | Description | Approved Security Functions | Keys and/or SSPs | Roles | Access rights to Keys and/or SSPs | Indicator |
|---|---|---|---|---|---|---|
| Integrity Status | Perform integrity test and return the status | N/A | N/A | CO | E | N/A |
| Key Derivation (HMAC) | Extract input key material and expand into additional keys | HMAC-SHA-1<br>HMAC-SHA2-224, -256, -384, -512<br>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 Keys | CO | W, E | Approved |
| Keyed Message Digest (GMAC) | Generate a keyed-hash message authentication code | AES-GMAC | AES Keys | CO | W, E | Approved |
| Keyed Message Digest (HMAC) | Generate a keyed-hash message authentication code | HMAC-SHA-1,<br>HMAC-SHA2-224, -256, -384, -512<br>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 |
| 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 |

| Service | Description | Approved Security Functions | Keys and/or SSPs | Roles | Access rights to Keys and/or SSPs | Indicator |
|---|---|---|---|---|---|---|
| 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 9– Non-Approved Services**

| Service | Description | Algorithm Accessed | Roles | Indicator |
|---|---|---|---|---|
| Key Derivation (HMAC) | Extract input key material and expand into additional keys | HMAC-KDF-SHA1 | CO | Non-Approved |

| Service | Description | Algorithm Accessed | Roles | Indicator |
|---|---|---|---|---|
| Keyed Message Digest | HMAC generation with key size less than 112 bit; non-compliant | HMAC-SHA1, HMAC-MD5 | CO | Non-Approved |
| Message Digest | Generate an MD2, MD4, or MD5 message digest | MD2, MD4, MD5 | CO | Non-Approved |
| Message Digest | Generate a SHA-1 message digest | SHA-1 | 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 | AES GCM encryption/decryption for 256 implementation | AES-GCM 64k/GMAC 64k | 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 kernel object.

- moc_crypto.ko - This is the shared library that contains the cryptographic module code, data, and constants.
- moc_crypto.ko.sig – This is the Integrity check signature file that contains an HMAC-SHA-256 of the crypto module.

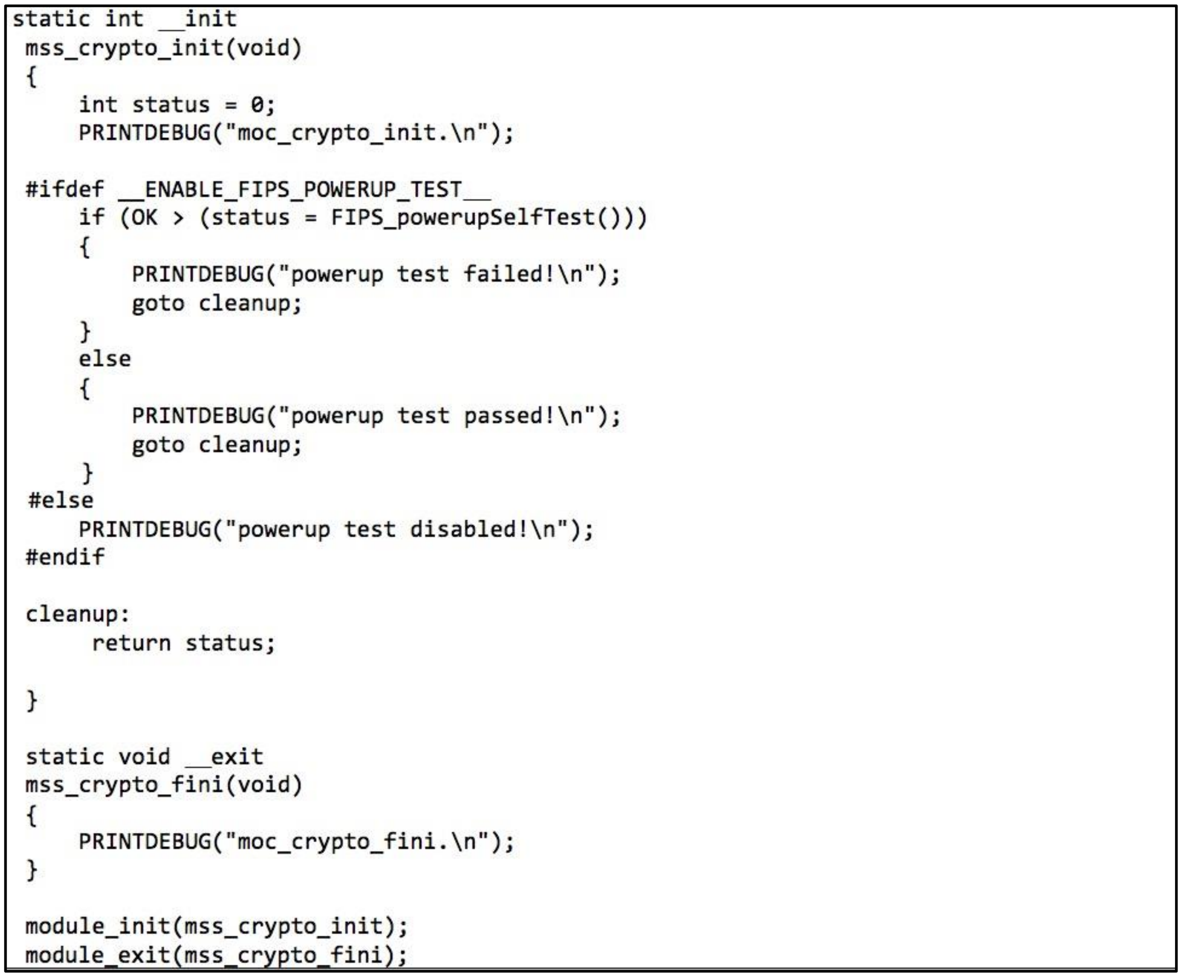
**Integrity Check**

The kernel module is protected with the authentication technique HMAC-SHA-256 described in

Table 13.

The HMAC for the kernel module is calculated during the manufacturing (build) process of the kernel module. This HMAC value is stored either within the resulting “moc_crypto.ko” kernel object or as a separate “moc_crypto.ko.sig” file dependent upon the development tools and target operating systems constraints.

During the load of the kernel object, the integrity check of the library code and constants occurs in the module startup function. It verifies the integrity of the kernel module by executing the HMAC-SHA-256 fingerprint algorithm on the moc_crypto.ko 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 kernel module into memory as shown below.



```
static int __init
 mss_crypto_init(void)
 {
     int status = 0;
     PRINTDEBUG("moc_crypto_init.\n");

 #ifdef __ENABLE_FIPS_POWERUP_TEST__
     if (OK > (status = FIPS_powerupSelfTest()))
     {
         PRINTDEBUG("powerup test failed!\n");
         goto cleanup;
     }
     else
     {
         PRINTDEBUG("powerup test passed!\n");
         goto cleanup;
     }
 #else
     PRINTDEBUG("powerup test disabled!\n");
 #endif

 cleanup:
      return status;

 }

 static void __exit
 mss_crypto_fini(void)
 {
     PRINTDEBUG("moc_crypto_fini.\n");
 }

 module_init(mss_crypto_init);
 module_exit(mss_crypto_fini);
```

**Figure 3 - Code Example for Self-Test**

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 18 Pre-Operational Self-Test. 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 Table 16). These KAT Self-Tests are performed on-demand. This means that the first time an approved algorithm is used, the CAST test for that algorithm is tested before the approved algorithm processing is done.

## 6 Operational Environment

The Module has a modifiable operational environment under the FIPS 140-3 definitions. The tested operational environments are listed in Table 2 above. In addition, Digicert 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 Cryptographic Module is software only.

## 8 Non-Invasive Security

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

## 9 Sensitive Security Parameter (SSP) Management

The SSPs access methods are described in Table 10 below:

**Table 10 – 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 |
| S1 | Only stored in volatile memory (RAM). |
| E1 | Input in plaintext (client key) |
| E2 | Output in plaintext |
| 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 11– SSPs**

| CSP | Strength (in bits) | Security Function / Cert. | Generation | Import /Export | Establish-ment | Storage | Zeroiza-tion | Use / Related SSPs |
|---|---|---|---|---|---|---|---|---|
| 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 |
| 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 |
| AES Keys | 128 to 256 bits | AES | N/A | E1 | N/A | S1 | Z1 | Used during AES encryption, decryption, CMAC and GMAC operations |
| HMAC Key | 112 to 256 bits | HMAC | N/A | 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 | N/A | E1 | N/A | S1 | Z1 | Used in deriving other keys per SP 800-108 with HMAC-SHA2-224, 256, 384, 512, HMAC- |

| CSP | Strength (in bits) | Security Function / Cert. | Generation | Import /Export | Establish -ment | Storage | Zeroiza- tion | Use / Related SSPs |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | SHA3-224, 256, 384, 512 operations |

Note: The module does not have PSPs.

## 9.1 DRBG Entropy Source

Table 12 – 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 13 – 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. A845) | 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 14 – 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 4k version | ES2 |
| AES-GCM, GMAC | KAT | AES encryption and decryption with GMAC authentication with 128-bit key – for 64k version | 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. This test also runs when Self-Tests are called or conditionally when DRBG is called. | ES2 |
| HMAC-SHA2 | KAT | Calculate a cryptographic hash-based authentication on data for SHA-1, SHA-224, SHA-256*, SHA-384, SHA-512<br><br>* HMAC-SHA-256 instantiation: This CAST is performed before the Software Integrity Check. This is the third KAT performed. | ES2 |
| 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<br><br>* 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 |

## 10.3 Error States and Indicators

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

**Table 15 – 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. |

## 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.

### 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 kernel module 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 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 16 – 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* |
| [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* |

| Abbreviation | Full Specification Name |
|---|---|
| [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* |
| [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 17 – 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 |

| Acronym | Definition |
|---|---|
| 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 |
| EMC | Electromagnetic Compatibility |
| EMI | Electromagnetic Interference |
| FIPS | Federal Information Processing Standard |
| GCM | Galois Counter Mode |
| HMAC | Hash Message Authentication Code |
| IG | Implementation Guidance |
| KAT | Known Answer Test |
| KDF | Key Derivation Function |
| KO | Kernel Object |
| 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 |
| TDES | Triple-DES |
| XTS | XEX-based Tweaked-codebook mode with ciphertext Stealing |