Tigera Cryptographic Module

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.1

June 27, 2022

Prepared for:

Tigera, Inc.
58 Maiden Ln
5th floor
San Francisco, CA 94108
tigera.io

Prepared by:

KeyPair Consulting Inc.
987 Osos Street
San Luis Obispo, CA 93401
+1 805.316.5024
keypair.us

Table of Contents

List of Tables

List of Figures

1 Introduction

The Tigera Cryptographic Module (hereafter referred to as the "module") is an open-source, general-purpose cryptographic library which provides FIPS 140-2 approved cryptographic algorithms to serve BoringSSL and other user-space applications. The validated version of the library is ae223d6138807a13006342edfeef32e813246b39. For the purposes of the FIPS 140-2 validation, its embodiment type is defined as multi-chip standalone.

The cryptographic module was tested on the following operational environments on the general-purpose computer (GPC) platforms detailed below:

Table 1 - Tested Operational Environments

#	Operating System	Platform	Compiler
1	Debian Linux 4.19.37 (Rodete)	HPE Z620 with Intel Xeon E5-2680 (with PAA)	Clang (7.0.1)
2	Debian Linux 4.19.37 (Rodete)	HPE Z620 with Intel Xeon E5-2680 (without PAA)	Clang (7.0.1)
3	Ubuntu Linux 18.04	Google Arcadia-Rome with AMD Rome (with PAA)	Clang (7.0.1)
4	Ubuntu Linux 18.04	Google Arcadia-Rome with AMD Rome (without PAA)	Clang (7.0.1)
5	Ubuntu Linux 18.04	Zaius P9 with POWER9 (with PAA)	Clang (7.0.1)
6	Ubuntu Linux 18.04	Zaius P9 with POWER9 (without PAA)	Clang (7.0.1)

The cryptographic module is also supported on the following operational environments for which operational testing and algorithm testing was not performed:

- Ubuntu 20.04 running on Intel Xeon (Skylake)
- Ubuntu 20.04 running on AMD EPYC Rome
- Red Hat Enterprise Linux 8.2 running on Intel Xeon (Skylake)
- Red Hat Enterprise Linux 8.2 running on AMD EPYC Rome
- Debian 10 running on Intel Xeon (Skylake)
- Debian 10 running on AMD EPYC Rome

As per FIPS 140-2 Implementation Guidance G.5, compliance is maintained for other versions of the respective operational environments where the module binary is unchanged. No claim can be made as to the correct operation of the module or the security strengths of the generated keys if any source code is changed and the module binary is reconstructed.

The GPC(s) used during testing met Federal Communications Commission (FCC) FCC Electromagnetic Interference (EMI) and Electromagnetic Compatibility (EMC) requirements for business use as defined by 47 Code of Federal Regulations, Part 15, Subpart B. FIPS 140-2 validation compliance is maintained when the module is operated on other versions of the GPOS running in single user mode, assuming that the requirements outlined in NIST IG G.5 are met.

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

2 FIPS 140-2 Security Levels

The following table lists the level of validation for each area in FIPS 140-2:

Table 2 - Validation Level by FIPS 140-2 Section

FIPS 140-2 Section Title	Validation Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
Electromagnetic Interference / Electromagnetic Compatibility	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A
Overall Level	1

3 Cryptographic Module Specification

3.1 Cryptographic Boundary

The module is a software library providing a C-language application program interface (API) for use by other processes that require cryptographic functionality. All operations of the module occur via calls from host applications and their respective internal daemons/processes. As such there are no untrusted services calling the services of the module.

The physical cryptographic boundary is the general-purpose computer on which the module is installed. The logical cryptographic boundary of the module is a single object file named “bcm.o” which is statically linked to BoringSSL. The module performs no communications other than with the calling application (the process that invokes the module services) and the host operating system.

Figure 1 shows the logical relationship of the cryptographic module to the other software and hardware components of the computer.

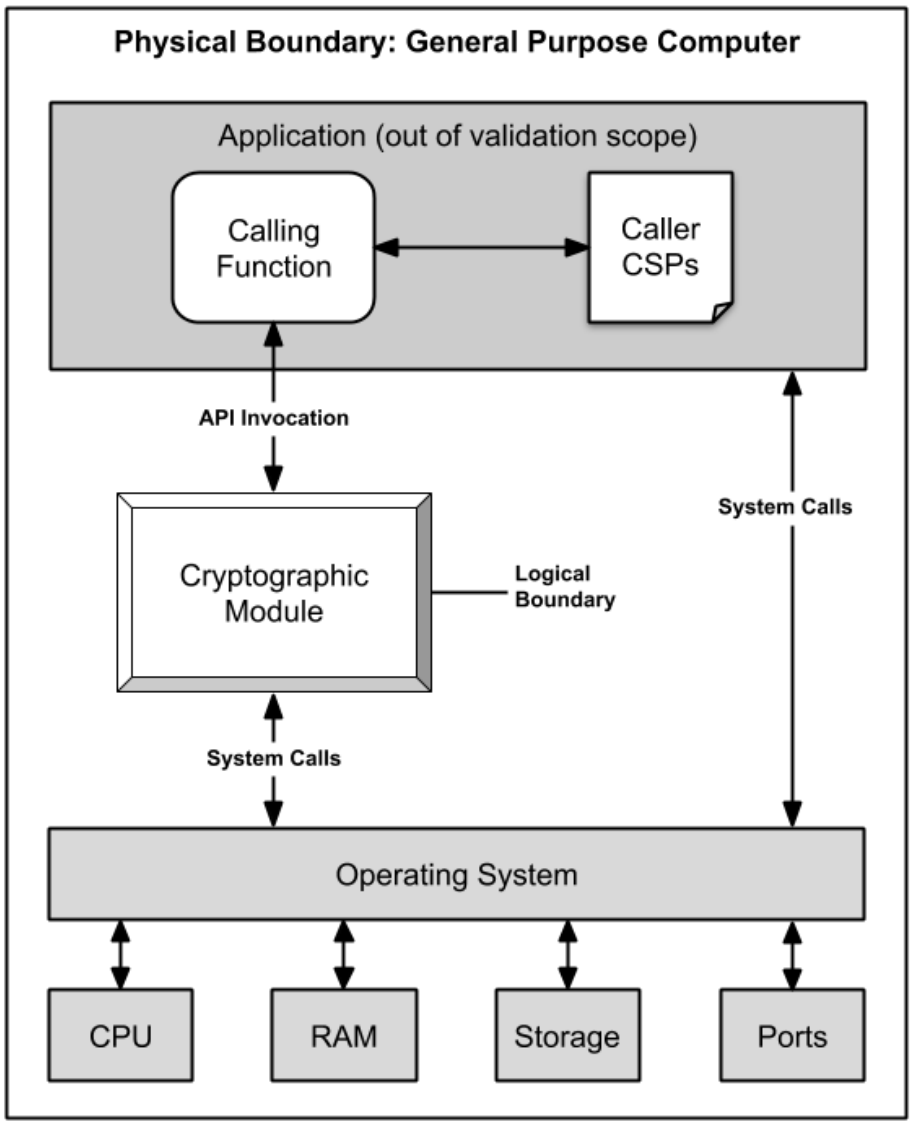

Figure 1 - Logical Boundary

4 Modes of Operation

The module supports two modes of operation: Approved and Non-approved. The module will be in FIPS-approved mode when all power up self-tests have completed successfully, and only Approved algorithms are invoked. See Table 7 below for a list of the supported Approved algorithms and Table 8 for allowed algorithms. The non-Approved mode is entered when a non-Approved algorithm is invoked. See Table 9 for a list of non-Approved algorithms.

5 Cryptographic Module Ports and Interfaces

The Data Input interface consists of the input parameters of the API functions. The Data Output interface consists of the output parameters of the API functions. The Control Input interface consists of the actual API input parameters. The Status Output interface includes the return values of the API functions.

Table 3 - Ports and Interfaces

FIPS Interface	Physical Ports	Logical Interfaces
Data input	Physical ports of the tested platforms	API input parameters
Data output	Physical ports of the tested platforms	API output parameters and return values
Control input	Physical ports of the tested platforms	API input parameters
Status output	Physical ports of the tested platforms	API return values
Power input	Physical ports of the tested platforms	N/A

As a software module, control of the physical ports is outside module scope. However, when the module is performing self-tests, or is in an error state, all output on the module's logical data output interfaces is inhibited.

6 Roles, Authentication and Services

The cryptographic module implements both User and Crypto Officer (CO) roles. The module does not support user authentication. The User and CO roles are implicitly assumed by the entity accessing services implemented by the module. A user is considered the owner of the thread that instantiates the module and, therefore, only one concurrent user is allowed.

The Approved services supported by the module and access rights within services accessible over the module's public interface are listed in the table below.

Table 4 - Approved Services, Roles and Access Rights

Service	Approved Security Functions	Keys and/or CSPs	Roles	Access Rights to Keys and/or CSPs
Module Initialization	N/A	N/A	CO	N/A
Symmetric Encryption/ Decryption	AES, Triple-DES	AES, Triple-DES symmetric keys	User, CO	Execute
Keyed Hashing	HMAC-SHA	HMAC key	User, CO	Execute
Hashing	SHS	None	User, CO	N/A
Random Bit Generation	CTR_DRBG	DRBG seed, internal state V and Key values	User, CO	Write/Execute
Signature Generation/ Verification	CTR_DRBG, RSA, ECDSA	RSA, ECDSA private key	User, CO	Write/Execute
Key Transport	RSA	RSA private key	User, CO	Write/Execute
Key Agreement	KAS ECC	EC DH private key	User, CO	Write/Execute
Key Generation	CTR_DRBG, RSA, ECDSA	RSA, ECDSA private key	User, CO	Write/Execute
On-demand Self-test	None	None	User, CO	Execute
Zeroization	None	All keys	User, CO	Write/Execute
Show Status	None	None	User, CO	N/A

The module provides the following non-Approved services which utilize algorithms listed in Table 9:

Table 5 - Non-Approved or Non-Security Relevant Services

Service	Non-Approved Functions	Roles	Keys and/or CSPs
Symmetric Encryption/ Decryption	AES (non-compliant), DES, Triple-DES (non-compliant)	User, CO	N/A
Hashing	MD4, MD5, POLYVAL	User, CO	N/A
Signature Generation/ Verification	RSA (non-compliant), ECDSA (non-compliant)	User, CO	N/A
Key Transport	RSA (non-compliant)	User, CO	N/A
Key Generation	RSA (non-compliant), ECDSA (non-compliant)	User, CO	N/A

The module also provides the following non-Approved or non-security relevant services over a non-public interface:

Table 6 - Non-Security Relevant Services

Service	Approved Security Functions	Roles	Access Rights to Keys and/or CSPs
Large integer operations	None	User, CO	N/A
Disable automatic generation of CTR_DRBG "additional_input" parameter	CTR_DRBG	User, CO	N/A
Wegman-Carter hashing with POLYVAL	None	User, CO	N/A

7 Physical Security

The cryptographic module is comprised of software only and thus does not claim any physical security.

8 Operational Environment

The cryptographic module operates under Ubuntu Linux 18.04 and Debian Linux 4.19.37. The module runs on a GPC running one of the operating systems specified in Table 1. Each approved operating system manages processes and threads in a logically separated manner. The module's user is considered the owner of the calling application that instantiates the module.

9 Cryptographic Algorithms & Key Management

9.1 Approved Cryptographic Algorithms

The module implements the following FIPS 140-2 Approved algorithms:

Table 7 - Approved Algorithms and CAVP Certificates

CAVP Cert. #	Algorithm	Standard	Mode/Method/Size	Use
C1063	AES	SP 800-38A FIPS 197 SP 800-38F	128, 192, 256 CBC, ECB, CTR	Encryption, Decryption, Authentication
C1063	AES	SP 800-38D	128 and 256 GCM/GMAC	Authenticated Encryption, Authenticated Decryption
C1063	KTS	SP 800-38F	128, 192, 256 KW, KWP	Key Wrapping, Key Unwrapping
C1063	CVL	SP 800-135rev1	TLS 1.0/1.1 and 1.2 KDF[1]	Key Derivation
Vendor Affirmed	CKG	SP 800-133	Cryptographic Key Generation	Key Generation
C1063	DRBG	SP 800-90Arev1	AES-256 CTR_DRBG	Random Bit Generation
C1063	ECDSA	FIPS 186-4	Signature Generation Component, Key Pair Generation, Signature Generation, Signature Verification, Public Key Validation P-224, P-256, P-384, P-521	Digital Signature Services
C1063	HMAC	FIPS 198-1	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	Generation, Authentication
C1063	RSA	FIPS 186-4	Key Generation, Signature Generation, Signature Verification (1024, 2048, 3072) Note: Key size 1024 is only used for Signature Verification	Digital Signature Services
C1063	SHA	FIPS 180-4	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications
C1063	Triple-DES	SP 800-67 SP 800-38A	TCBC, TECB	Encryption, Decryption
Vendor Affirmed	KAS-SSC	SP 800-56rev3	EC Diffie-Hellman P-224, P-256, P-384 and P-521 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Key Agreement Scheme – Key Agreement Scheme Shared Secret Computation (KAS-SSC) per SP 800-56Arev3, Key Derivation per SP 800-135 (CVL Cert. #C1063 TLS KDF)

[1] The module supports FIPS 140-2 approved/allowed cryptographic algorithms for TLS 1.0, 1.1 and 1.2.

9.2 Allowed Cryptographic Algorithms

The module supports the following non-FIPS 140-2 Approved but allowed algorithms that may be used in the Approved mode of operation.

Table 8 - Allowed Algorithms

Algorithm	Use
RSA Key Transport	Key establishment methodology provides between 112 and 256 bits of encryption strength
MD5	When used with the TLS protocol version 1.0 and 1.1
NDRNG	Used only to seed the Approved DRBG

9.3 Non-Approved Cryptographic Algorithms

The module employs the methods listed in Table 9, which are not allowed for use in a FIPS-Approved mode. Their use will result in the module operating in a non-Approved mode.

Table 9 - Non-Approved Algorithms

Algorithm	Algorithm
MD5, MD4	DES
AES-GCM (non-compliant)	AES (non-compliant)
ECDSA (non-compliant)	RSA (non-compliant)
POLYVAL	Triple-DES (non-compliant)

9.4 Cryptographic Key Management

The table below provides a complete list of Private Keys and CSPs used by the module:

Table 10 - Keys and CSPs Supported

Key/CSP Name	Key Description	Generated/Input	Output
AES Key	AES (128/192/256) encrypt/decrypt key	Input via API in plaintext	Output via API in plaintext
AES-GCM/GMAC Key	AES (128/256) encrypt/decrypt/ generate/verify key	Input via API in plaintext	Output via API in plaintext
AES Wrapping Key	AES (128/192/256) key wrapping key	Input via API in plaintext	Output via API in plaintext
Triple-DES Key	Triple-DES (3-Key) encrypt/decrypt key	Input via API in plaintext	Output via API in plaintext
ECDSA Signing Key	ECDSA (P-224/P-256/P-384/P-521) signature generation key	Internally generated or input via API in plaintext	Output via API in plaintext
EC DH Private Key	EC DH (P-224/P-256/P-384/P-521) private key	Internally generated or input via API in plaintext	Output via API in plaintext
HMAC Key	Keyed hash key (160/224/256/384/512)	Input via API in plaintext	Output via API in plaintext
RSA Key (Key Transport)	RSA (2048 to 16384 bits) key decryption (private key transport) key	Internally generated or input via API in plaintext	Output via API in plaintext
RSA Signature Generation Key	RSA (2048 to 16384 bits) signature generation key	Internally generated or input via API in plaintext	Output via API in plaintext
TLS Pre-Master Secret	Shared Secret; 48 bytes of pseudorandom data	Internally Generated	Output via API in plaintext
TLS Master Secret	Shared Secret; 48 bytes of pseudorandom data	Internally derived via key derivation function defined in SP 800-135 KDF (TLS)	Output via API in plaintext
CTR_DRBG V (Seed)	128 bits	Internally generated	Does not exit the module
CTR_DRBG Key	256 bits	Internally generated	Does not exit the module
CTR_DRBG Entropy Input	384 bits	Input via API in plaintext	Does not exit the module

9.5 Public Keys

The table below provides a complete list of the Public keys used by the module:

Table 11 - Public Keys Supported

Public Key Name	Key Description
ECDSA Verification Key	ECDSA (P-224/P-256/P-384/P-521) signature verification key
EC DH Public Key	EC DH (P-224/P-256/P-384/P-521) public key
RSA Key (Key Transport)	RSA (2048 to 16384 bits) key encryption (public key transport) key
RSA Signature Verification Key	RSA (1024 to 16384 bits) signature verification public key

9.6 Key Generation

The module supports generation of ECDSA, EC Diffie-Hellman and RSA key pairs as specified in Section 5 of NIST SP 800-133. The module employs a NIST SP 800-90A random bit generator for creation of the seed for asymmetric key generation. The module requests a minimum number of 128 bits of entropy from its Operational Environment per each call.

The output data path is provided by the data interfaces and is logically disconnected from processes performing key generation or zeroization. No key information will be output through the data output interface when the module zeroizes keys.

9.7 Key Storage

The cryptographic module does not perform persistent storage of keys. Keys and CSPs are passed to the module by the calling application. The keys and CSPs are stored in memory in plaintext. Keys and CSPs residing in internally allocated data structures (during the lifetime of an API call) can only be accessed using the module defined API. The operating system protects memory and process space from unauthorized access.

9.8 Key Zeroization

The module is passed keys as part of a function call from a calling application and does not store keys persistently. The calling application is responsible for parameters passed in and out of the module. The Operating System and the calling application are responsible to clean up temporary or ephemeral keys.

10 Self-Tests

FIPS 140-2 requires the module to perform self-tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. Some functions require conditional tests during normal operation of the module. The supported tests are listed and described in this section.

10.1 Power-On Self-Tests

Power-on self-tests are run upon the initialization of the module and do not require operator intervention to run. If any of the tests fail, the module will not initialize. The module will enter an error state and no services can be accessed.

The module implements the following power-on self-tests:

Table 12 - Power-On Self-Tests

Type	Test
Integrity Test	HMAC-SHA-512
Known Answer Test (KAT)	AES KAT (encryption and decryption. Key size: 128-bits)
	AES-GCM/GMAC KAT (encryption and decryption. Key size: 128-bits)
	Triple-DES KAT (encryption and decryption. Key size: 168-bits)
	ECDSA KAT (signature generation/signature verification. Curve: P-256)
	HMAC KAT (HMAC-SHA-1, HMAC-SHA-512)
	SP 800-90A CTR_DRBG KAT (Key size: 256-bits)
	RSA KAT (signature generation/signature verification and encryption/decryption. Key size: 2048-bit)
	SHA KAT (SHA-1, SHA-256, SHA-512)

Each module performs all power-on self-tests automatically when the module is initialized. All power-on self-tests must be passed before a User/Crypto Officer can perform services. The Power-on self-tests can be run on demand by power-cycling the host platform.

10.2 Conditional Self-Tests

Conditional self-tests are run during operation of the module. If any of these tests fail, the module will enter an error state, where no services can be accessed by the operators. The module can be reinitialized to clear the error and resume FIPS mode of operation. Each module performs the following conditional self-tests:

Table 13 - Conditional Self-Tests

Type	Test
Pair-wise Consistency Test	ECDSA Key Pair generation RSA Key Pair generation
CRNGT	Performed on NDRNG per IG 9.8
DRBG Health Tests	Performed on DRBG, per SP 800-90A Section 11.3. Required per IG C.1.

Pairwise consistency tests are performed for both possible modes of use, e.g. Sign/Verify and Encrypt/Decrypt.

11 Mitigation of other Attacks

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

12 Guidance and Secure Operation

12.1 Installation Instructions

The following steps shall be performed to build, compile and statically link the Module to BoringSSL on the tested Operational Environments.

The below tools are required in order to build and compile the module:

- Clang compiler version 7.0.1 (http://releases.llvm.org/download.html)
- Go programming language version 1.12.7 (https://golang.org/dl/)
- Ninja build system version 1.90 (https://github.com/ninja-build/ninja/releases)

Once the above tools have been obtained, issue the following command to create a CMake toolchain file to specify the use of Clang:

- printf "set(CMAKE_C_COMPILER \"clang\")\nset(CMAKE_CXX_COMPILER \"clang++\")\n" > ${HOME}/toolchain

The FIPS 140-2 validated release of the module can be obtained by downloading the tarball containing the source code at the following location:

https://commondatastorage.googleapis.com/chromium-boringssl-fips/boringsslae223d6138807a13006342edfeef32e813246b39.tar.xz

or by issuing the following command:

wget https://commondatastorage.googleapis.com/chromium-boringssl-fips/boringsslae223d6138807a13006342edfeef32e813246b39.tar.xz

The set of files specified in the archive constitutes the complete set of source files of the validated module. There shall be no additions, deletions, or alterations of this set as used during module build.

The downloaded tarball file can be verified using the below SHA-256 digest value:

3b5fdf23274d4179c2077b5e8fa625d9debd7a390aac1d165b7e47234f648bb8

By issuing the following command:

- sha256sum boringssl-ae223d6138807a13006342edfeef32e813246b39.tar.xz

After the tarball has been extracted, the following commands will compile the module:

1. cd boringssl
2. mkdir build && cd build && cmake -GNinja -DCMAKE_TOOLCHAIN_FILE=${HOME}/toolchain -DFIPS=1 -DCMAKE_BUILD_TYPE=Release ..
3. ninja
4. ninja run_tests

Upon completion of the build process. The module's status can be verified by issuing:

- ./tool/bssl isfips

The module will print "1" if it is in a FIPS 140-2 validated mode of operation.

12.2 Secure Operation

12.2.1 Initialization

The cryptographic module is initialized by loading the module before any cryptographic functionality is available. In User Space the operating system is responsible for the initialization process and loading of the library. The module is designed with a default entry point (DEP) which ensures that the power-up tests are initiated automatically when the module is loaded.

12.2.2 Usage of AES OFB, CFB and CFB8

In approved mode, users of the module must not utilize AES OFB, CFB and CFB8.

12.2.3 Usage of AES-GCM

In the case of AES-GCM, the IV generation method is user selectable and the value can be computed in more than one manner. AES GCM encryption and decryption are used in the context of the TLS protocol version 1.2 (compliant to Scenario 1 in FIPS 140-2 A.5). The module is compliant with NIST SP 800-52 and the mechanism for IV generation is compliant with RFC 5288. The module ensures that it's strictly increasing and thus cannot repeat. When the IV exhausts the maximum number of possible values for a given session key, the first party, client or server, to encounter this condition may either trigger a handshake to establish a new encryption key in accordance with RFC 5246, or fail. In either case, the module prevents and IV duplication and thus enforces the security property.

The module supports internal IV generation by the module's Approved DRBG. The DRBG seed is generated inside the module's physical boundary. The IV is 96-bits in length per NIST SP 800-38D, Section 8.2.2 and FIPS 140-2 IG A.5 scenario 2.

The selection of the IV construction method is the responsibility of the user of this cryptographic module. In approved mode, users of the module must not utilize GCM with an externally generated IV.

Per IG A.5, in the event module power is lost and restored the consuming application must ensure that any of its AES-GCM keys used for encryption or decryption are re-distributed.

12.2.4 Usage of Triple-DES

In accordance with CMVP IG A.13, when operating in a FIPS approved mode of operation, the same Triple-DES key shall not be used to encrypt more than 2^{20} or 2^{16} 64-bit data blocks.

The TLS protocol governs the generation of the respective Triple-DES keys. Please refer to IETF RFC 5246 (TLS) for details relevant to the generation of the individual Triple-DES encryption keys. The user is responsible for ensuring that the module limits the number of encrypted blocks with the same key to no more than 2^{20} when utilized as part of a recognized IETF protocol.

For all other uses of Triple-DES the user is responsible for ensuring that the module limits the number of encrypted blocks with the same key to no more than 2^{16}.

12.2.5 RSA and ECDSA Keys

The module allows the use of 1024 bits RSA keys for legacy purposes including signature generation, which is disallowed to be used in FIPS Approved mode as per NIST SP 800-131A. Therefore, the cryptographic operations with the non-approved key sizes will result in the module operating in non-Approved mode implicitly.

Approved algorithms shall not use the keys generated by the module's non-Approved key generation methods.

13 References and Standards

The following Standards are referred to in this Security Policy.

Table 14 - References and Standards

Abbreviation	Full Specification Name
FIPS 140-2	Security Requirements for Cryptographic modules, May 25 2001
FIPS 180-4	Secure Hash Standard (SHS)
FIPS 186-4	Digital Signature Standard (DSS)
FIPS 197	Advanced Encryption Standard
FIPS 198-1	The Keyed-Hash Message Authentication Code (HMAC)
IG	Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program
SP 800-38A	Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode
SP 800-38D	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
SP 800-38F	Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping
SP 800-56A	Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography
SP 800-67	Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher
SP 800-90A	Recommendation for Random Number Generation Using Deterministic Random Bit Generators
SP 800-52	Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations
SP 800-133	Recommendation for Cryptographic Key Generation
SP 800-135	Recommendation for Existing Application-Specific Key Derivation Functions

14 Acronyms and Definitions

Table 15 - Acronyms and Definitions

Acronym	Definition
AES	Advanced Encryption Standard
API	Application Programming Interface
CBC	Cipher-Block Chaining
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CO	Cryptographic Officer
CPU	Central Processing Unit
CSP	Critical Security Parameter
CTR	Counter-mode
CVL	Component Validation List
DES	Data Encryption Standard
DRAM	Dynamic Random Access Memory
DRBG	Deterministic Random Number Generator
EC	Elliptic Curve
ECB	Electronic Code Book

Acronym	Definition
EC DH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GPC	General Purpose Computer
HMAC	Keyed-Hash Message Authentication Code
IG	Implementation Guidance
IV	Initialization Vector
KAT	Known Answer Test
MAC	Message Authentication Code
MD5	Message Digest algorithm MD5
N/A	Not-Applicable
NDRNG	Non-Deterministic Random Number Generator
OS	Operating System
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
Triple-DES	Triple Data Encryption Standard