Microsoft FIPS 140 Validation

Microsoft Azure Linux OpenSSL Cryptographic Library (version 2.0)

Non-Proprietary

Security Policy Document

Version Number	1.4
Updated On	May 16th, 2024

Version History

Version	Date	Summary of changes
1.0	September 17, 2021	Draft sent to NIST CMVP
1.1	March 14, 2023	Updates from CMVP feedback
1.2	April 5, 2023	Updates from CMVP feedback
1.3	April 18, 2023	Updates from CMVP feedback
1.4	May 16th,2024	Renaming updates

TABLE OF CONTENTS

1 Introduction

The Microsoft Azure Linux OpenSSL Cryptographic Library[1] (the "module") is a general-purpose, software-based cryptographic module that supports FIPS 140-2 approved cryptographic algorithms. The codebase of the module is a combination of standard OpenSSL shared libraries and custom development work by Microsoft. It provides an application programming interface (API) for use by programs that require cryptographic functionality. The module is implemented as a set of shared libraries and binary files.

1.1 List of Cryptographic Module Libraries and Binaries

The module includes the following libraries and binaries:

Library or Binary	Description
/usr/lib64/.libcrypto.so.1.1.1k.hmac	Integrity check HMAC value for libcrypto
/usr/lib64/.libssl.so.1.1.k.hmac	Integrity check HMAC value for libssl
/usr/lib64/libcrypto.so.1.1.1k	Shared library for cryptographic algorithms
/usr/lib64/libssl.so.1.1.1k	Shared library for TLS/DTLS network protocols

The following package is required for the module to operate:

Package Name	Description
openssl-libs-1.1.1k-13.cm1.x86_64.rpm	Red Hat Package Manager (RPM) package that delivers the module binaries.

1.2 Validated Platforms

The module has been validated on the following platforms:

Test Platform	Processor	Operating System	Configuration
Azure Compute C2030 Server	Intel® Xeon® Platinum 8272CL (Intel x64)	Azure Linux 2.0	With and without AES-NI (PAA)
Azure Host Hypervisor, running on Azure Compute C2030 Server	Intel® Xeon® Platinum 8272CL (Intel x64)	Azure Linux 2.0	With and without AES-NI (PAA)

1.3 Modes of Operation

The module supports two modes of operation:

1. **FIPS-approved mode:** This is the approved mode of operation. In this mode, only approved security functions with sufficient security strength can be used.

[1] The Microsoft Azure Linux OpenSSL Cryptographic Library in this validation is based on OpenSSL version 1.1.1k-13.

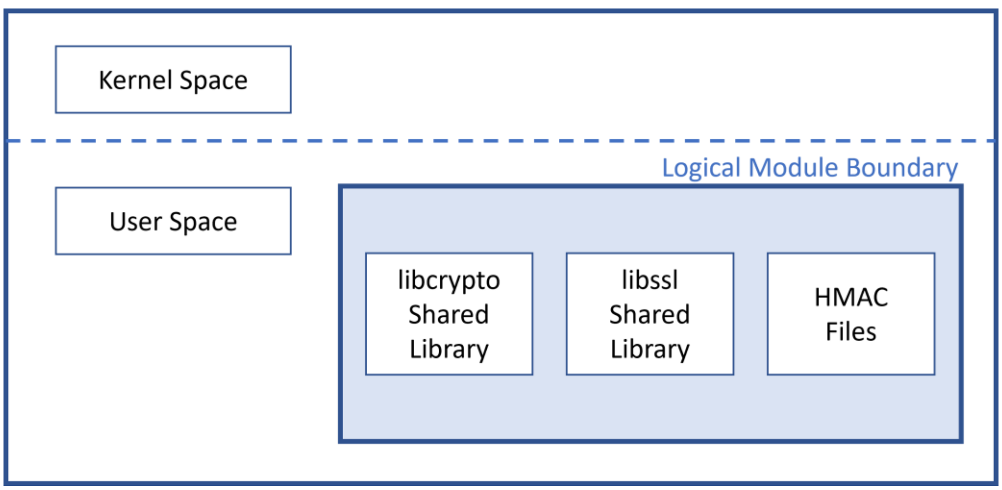
2. **Non-FIPS approved mode:** This is the non-approved mode of operation. In this mode, non-approved security functions can also be used.

The mode may be determined either by policy or by executing an approved security function. FIPS approved mode may be enabled by adding "fips=1" to the kernel command line or by calling the OpenSSL API, FIPS_mode_set.

The module verifies the integrity of the runtime executable by performing an integrity check that leverages an HMAC-SHA-256 digest computed at build time. If the digests match, the power-up self-test is performed. All data output is inhibited if the model is in a self-test state. If the self-test passes, the module is operational. If the self-test fails, the module goes into an error state and is not functional.

1.4 Cryptographic Boundary

The Microsoft Azure Linux OpenSSL Cryptographic Library is defined as a multi-chip standalone module. The logical cryptographic boundary of the module is the set of shared library files and their integrity check HMAC files, as described in the List of Cryptographic Module Libraries and Binaries. The following software block diagram depicts the logical boundary of the module.



1.5 FIPS 140-2 Approved Algorithms

The Microsoft Azure Linux OpenSSL Cryptographic Library implements the following FIPS-140-2 Approved algorithms:[2]

[2] This module may not use some of the capabilities described in each CAVP certificate. Per FIPS 140-2 IG G.5, the CMVP makes no statement as to the correct operation of the Module or the security strengths of the generated keys when those are ported and executed in an operational environment not listed on the validation certificate.

Algorithm	Purpose	Standards, Modes, and Methods	Keys and CSPs	CAVP Certificate
AES	Encryption and Decryption	FIPS 197 (AES) NIST SP 800-38A (CBC, CFB1, CFB8, CFB128, CTR, ECB) NIST SP 800-38C (CCM) NIST SP 800-38D (GCM) NIST SP 800-38E (XTS[3]) NIST SP 800-38F (KW, KWP)	AES keys 128 bits, 192 bits (except XTS-AES) and 256 bits	# A2665
	MAC Generation and Verification	NIST SP 800-38B (CMAC)	Mac length 128 bits	# A2665
Triple-DES[4]	Encryption and Decryption	NIST SP 800-67 Rev2 NIST SP 800-38A (ECB) NIST SP 800-38A (CBC, OFB, CFB1, CFB8, CFB64)	Triple-DES keys 168 bits	# A2665
	Mac Generation and Verification	NIST SP 800-67 Rev2 NIST SP 800-38B (CMAC)		# A2665
DSA	Domain Parameters Generation and Verification, Key Generation, Signature Generation	FIPS 186-4	DSA keys: • L=2048, N=224 • L=2048, N=256 • L=3072, N=256	# A2665
	Signature Verification		DSA keys: • L=1024, N=160 • L=2048, N=224 • L=2048, N=256 • L=3072, N=256 1024-bit DSA signature verification for legacy use.	# A2665
RSA	Key Generation	FIPS 186-4 Appendix B.3.3	RSA keys: • 2048 bits • 3072 bits • 4096 bits	# A2665

[3] AES XTS must be used only to protect data at rest and the caller needs to ensure that the length of data encrypted does not exceed 2^20 AES blocks.
[4] After December 31, 2023, Triple-DES is Approved for only decryption.

Algorithm	Purpose	Standards, Modes, and Methods	Keys and CSPs	CAVP Certificate
	Signature Generation	FIPS 186-4 PKCS#1 v1.5 and PSS SHA-224, SHA-256, SHA-384, SHA-512	RSA keys: • 2048 bits • 3072 bits • 4096 bits	# A2665
	Signature Verification	FIPS 186-4 PKCS#1 v1.5 and PSS SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	RSA keys: • 1024 bits • 2048 bits • 3072 bits • 4096 bits 1024-bit RSA signature verification for legacy use	# A2665
	Signature Generation	ANSI X9.31 SHA-256, SHA-384, SHA-512	RSA keys: • 2048 bits • 3072 bits • 4096 bits	# A2665
	Signature Verification	ANSI X9.31 SHA-1, SHA-256, SHA-384, SHA-512	RSA keys: • 1024 bits • 2048 bits • 3072 bits • 4096 bits 1024-bit RSA signature verification for legacy use	# A2665
ECDSA	Key Pair Generation and Public Key Verification	FIPS 186-4	ECDSA keys based on P-256, P-384 or P-521 curve	# A2665
	Signature Generation	FIPS 186-4 SHA-224, SHA-256, SHA-384, SHA-512	ECDSA keys based on P-224, P-256, P-384 or P-521 curve	# A2665
	Signature Verification			# A2665
DRBG	Random Number Generation	NIST SP 800-90A Rev1 (CTR_DRBG) AES-128, AES-192, AES-256	Entropy input string, seed, C, V, and Key	# A2665

Algorithm	Purpose	Standards, Modes, and Methods	Keys and CSPs	CAVP Certificate
		NIST SP 800-90A Rev1 (Hash_DRBG, HMAC_DRBG) SHA-1, SHA-224, SHA-256, SHA-384, SHA-512		# A2665
SHS	Hashing	FIPS 180-4 SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	N/A	# A2665
SHA-3	Hashing	FIPS 202 SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	N/A	# A2665
HMAC	Message Integrity	FIPS 198-1 HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, SHA-512/224, SHA-512/256	At least 112 bits HMAC key	# A2665
		HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512		# A2665
KAS-ECC-SSC	EC Diffie-Hellman Key Agreement	NIST SP 800-56A Rev3 Scheme: ephemeralUnified KAS Role: initiator, responder	P-224, P-256, P-384, P-521	# A2665
KAS-FFC-SSC	Diffie-Hellman Key Agreement	NIST SP 800-56A Rev3 Scheme: dhEphemeral KAS Role: initiator, responder	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192	# A2665
Key Agreement Scheme (ECC/FFC)	Key Agreement Scheme per SP 800-56A Rev3 with key	NIST SP 800-56A Rev3 NIST SP 800-135 Rev1 KAS (ECC):	KAS (ECC): P-224, P-256, P-384, P-521. KAS (FFC):	# A2665

Algorithm	Purpose	Standards, Modes, and Methods	Keys and CSPs	CAVP Certificate
KAS (KAS-SSC CAVP # A2665, CVL # A2665);	derivation function (SP 800-135 Rev1)	Scheme: ephemeralUnified KAS Role: initiator, responder KAS (FFC): Scheme: dhEphemeral KAS Role: initiator, responder	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192	
Key Agreement Scheme (ECC/FFC) **KAS (KAS-SSC CAVP # A2665, KDA # A2665);**	Key Agreement Scheme per SP 800-56A Rev3 with key derivation algorithm (SP 800-56C Rev2)	NIST SP 800-56A Rev3 NIST SP 800-56C Rev2 KAS (ECC): Scheme: ephemeralUnified KAS Role: initiator, responder KAS (FFC): Scheme: dhEphemeral KAS Role: initiator, responder	KAS (ECC): P-224, P-256, P-384, P-521. KAS (FFC): MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192	# A2665
Safe Primes	Diffie-Hellman Key Agreement	NIST SP 800-56A Rev3 Generation and Verification	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192	# A2665
(CVL) TLS-KDF[5]	Key Derivation in TLS	NIST SP 800-135 Rev1	Derived Key, TLS Pre-Master Secret, Master Secret	# A2665
PBKDF	Password-Based Key Derivation	NIST SP 800-132 SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	PBKDF password PBKDF Derived Key	# A2665
(CVL) SSH KDF	Secure Shell Key Derivation	NIST SP 800-135 Rev1	SSH KDF Derived Key	# A2665

[5] Per IG D.11, the TLS and SSH protocols have not been reviewed or tested by the CAVP and CMVP.

Algorithm	Purpose	Standards, Modes, and Methods	Keys and CSPs	CAVP Certificate
KDA OneStep	Single-Step Key Derivation	NIST SP 800-56C Rev2 SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Derived Key	# A2665
KTS	Key Establishment	NIST SP 800-38F AES KW, KWP AES CCM AES GCM	AES keys 128, 192, 256 bits	# A2665
		NIST SP 800-38F AES CBC and HMAC	AES keys 128, 192, 256 bits	# A2665
		NIST SP 800-38F Triple-DES CBC and HMAC[6]	Triple DES keys 168 bits Encryption strength of 112 bits.	# A2665
ENT	CPU time jitter entropy source	NIST SP 800-90B	N/A	N/A

1.6 Non-Approved Algorithms

The following tables present the non-FIPS 140-2 approved algorithms implemented by the module. One non-approved algorithm is allowed in FIPS-approved mode:

Algorithm	Usage	Keys/CSPs
MD5	Message digest for TLS. (This is the only usage allowed in Approved mode; all other MD5 usage causes the module to operate in non-FIPS mode.) Note: No security is claimed for MD5.	N/A

The remainder of the non-approved algorithms may not be used in FIPS-approved mode. Any use of the following non-approved algorithms will cause the Module to operate in the non-FIPS mode:

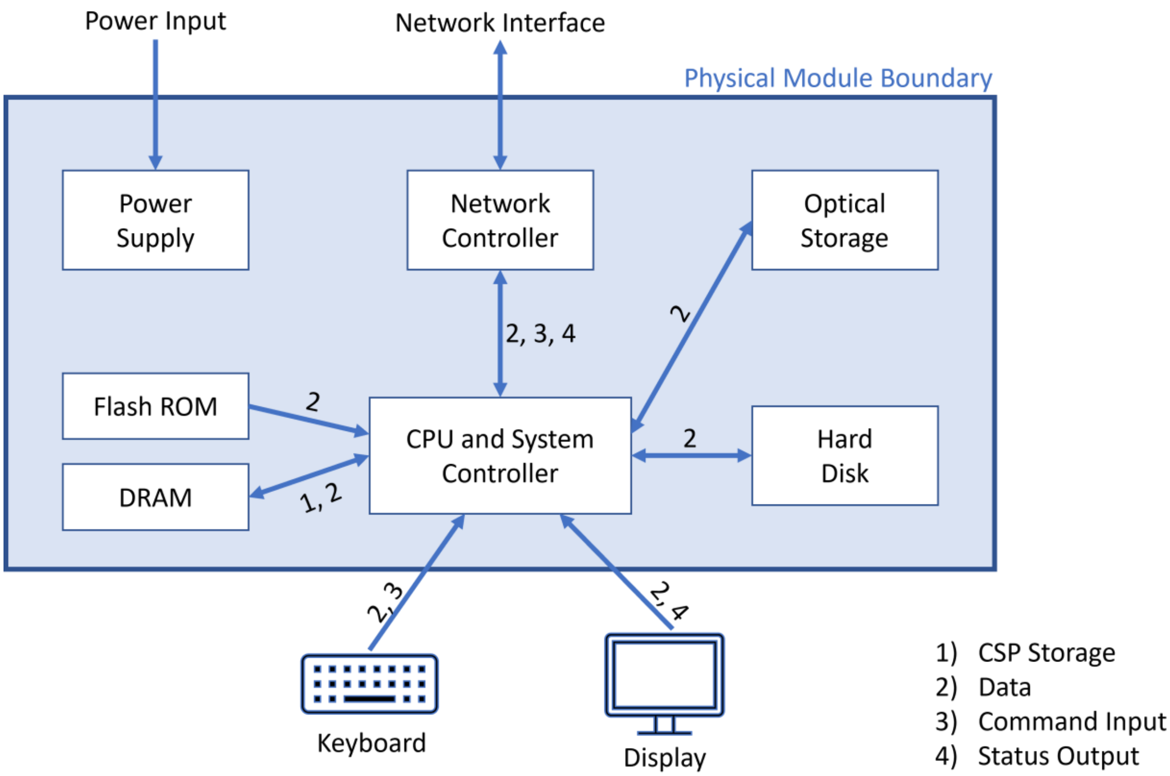
Algorithm	Usage	Keys/CSPs

[6] After December 31, 2023, Triple-DES is Approved for only decryption.

RSA (encrypt, decrypt)	Key wrapping, Key establishment	RSA keys
RSA with key sizes not listed in the approved-algorithms table	Sign, verify and key generation	RSA keys
Digital Signature Generation (DSA, ECDSA, and RSA) using SHA-1	Sign	DSA, ECDSA, and RSA keys
DSA with key sizes not listed in the approved-algorithms table	Sign, verify, and key generation	DSA keys
KAS-FFC-SSC using any key size not listed in the Approved-algorithms table	Key agreement	Diffie-Hellman keys
KAS-ECC-SSC with P-192 curve, K-curves, B-curves, and non-NIST curves	Key agreement	EC Diffie-Hellman private key
ECDSA with curves not listed in the approved-algorithms table	Key Pair Generation, Public Key Verification, Signature Generation, Signature Verification	ECDSA keys
AES-OCB	Authenticated encryption and decryption	Symmetric key
AES-OFB	Encryption and Decryption	AES keys
Camellia	Encryption and decryption	Symmetric key
CAST	Encryption and decryption	Symmetric key
DES	Encryption and decryption	Symmetric key
2-Key 3DES	Encryption and decryption	Symmetric key
IDEA	Encryption and decryption	Symmetric key
KBKDF	Key derivation	Derived key
RFC-3961 KDF	Key derivation	Derived key
RFC 7914 KDF	Key derivation	Derived key
HKDF	Key derivation	Derived key
MD2	Hash function	N/A
MD4	Hash function	N/A
MD5	Hash function	N/A
RC2	Encryption and decryption	Symmetric key
RC4	Encryption and decryption	Symmetric key
RC5	Encryption and decryption	Symmetric key
RIPEMD	Hash function	N/A
Whirlpool	Hash function	N/A

1.7 Hardware Block Diagram

The Microsoft Azure Linux OpenSSL Cryptographic Library is a multi-chip standalone software module. The physical boundary of the module is the physical boundary of the computer that contains the module. The following hardware block diagram depicts the hardware components used by the module and the physical module boundary.



2 Cryptographic Module Ports and Interfaces

The Microsoft Azure Linux OpenSSL Cryptographic Library is a software module and has no physical ports of its own. The physical ports of the module are interpreted as those on the underlying hardware platform. The logical interfaces are the application program interface (API) through which applications request services. The table below describes the logical interfaces and the physical ports they leverage:

Logical Interface	Physical Port	Description
Data Input	Ethernet ports	API input parameters
Data Output	Ethernet ports	API output parameters
Control Input	Keyboard, serial port, Ethernet port, network	API function calls
Status Output	Serial port, Ethernet port, network	API return codes and error messages
Power Input	PC power supply port	N/A

3 Roles, Services and Authentication

3.1 Roles

The module supports two roles: User and Crypto Officer. The User and Crypto Officer roles are implicitly assumed by the party accessing services implemented by the Module.

- **User role**: Performs all services except module installation. This is the role assumed by calling applications.
- **Crypto Officer role:** Performs module installation and configuration.

3.2 Services

The following tables provide a mapping of the available services, algorithms, Critical Security Parameters, and access types that the module provides. The FIPS 140-2 Approved services available in FIPS mode include the following:

Service	Keys, CSPs	Role	Access
Symmetric encryption/decryption	AES and Triple-DES keys	User	Read, Execute
Asymmetric key generation	RSA, DSA, and ECDSA private keys	User	Read, Write, Execute
Digital signature generation and verification	RSA, DSA, and ECDSA private keys	User	Read, Execute
TLS network protocol	AES or Triple-DES keys and HMAC keys	User	Read, Execute
TLS key agreement	AES or Triple-DES key, RSA, DSA or ECDSA private key, HMAC Key, Pre-Master Secret, Master Secret, Diffie-Hellman Private and EC Diffie-Hellman Private	User	Read, Write, Execute
Shared secret computation	Diffie-Hellman and EC Diffie-Hellman public and private keys, shared secret	User	Read, Write
Certificate Management	RSA, DSA, or ECDSA private keys	User	Read, Write, Execute
Keyed hash	HMAC or CMAC keys	User	Read, Execute
Message digest	Hashed values	User	N/A
Random number generation (SP 800-90A DBRG)	Entropy input string, seed, C, V and Key	User	Read, Write, Execute
Key derivation	PBKDF, SSH-KDF, TLS-KDF, KDA	User	Read, Write, Execute
Show status	None	User	N/A
Module initialization	None	User	N/A
Self-test	None	User	N/A
Zeroization	All CSPs above	User	Read, Write, Execute
On-demand self-test	None	User	Read, Write, Execute

Service	Keys, CSPs	Role	Access
Module installation	None	Crypto Officer	N/A

The non-Approved services available in non-FIPS mode include the following:

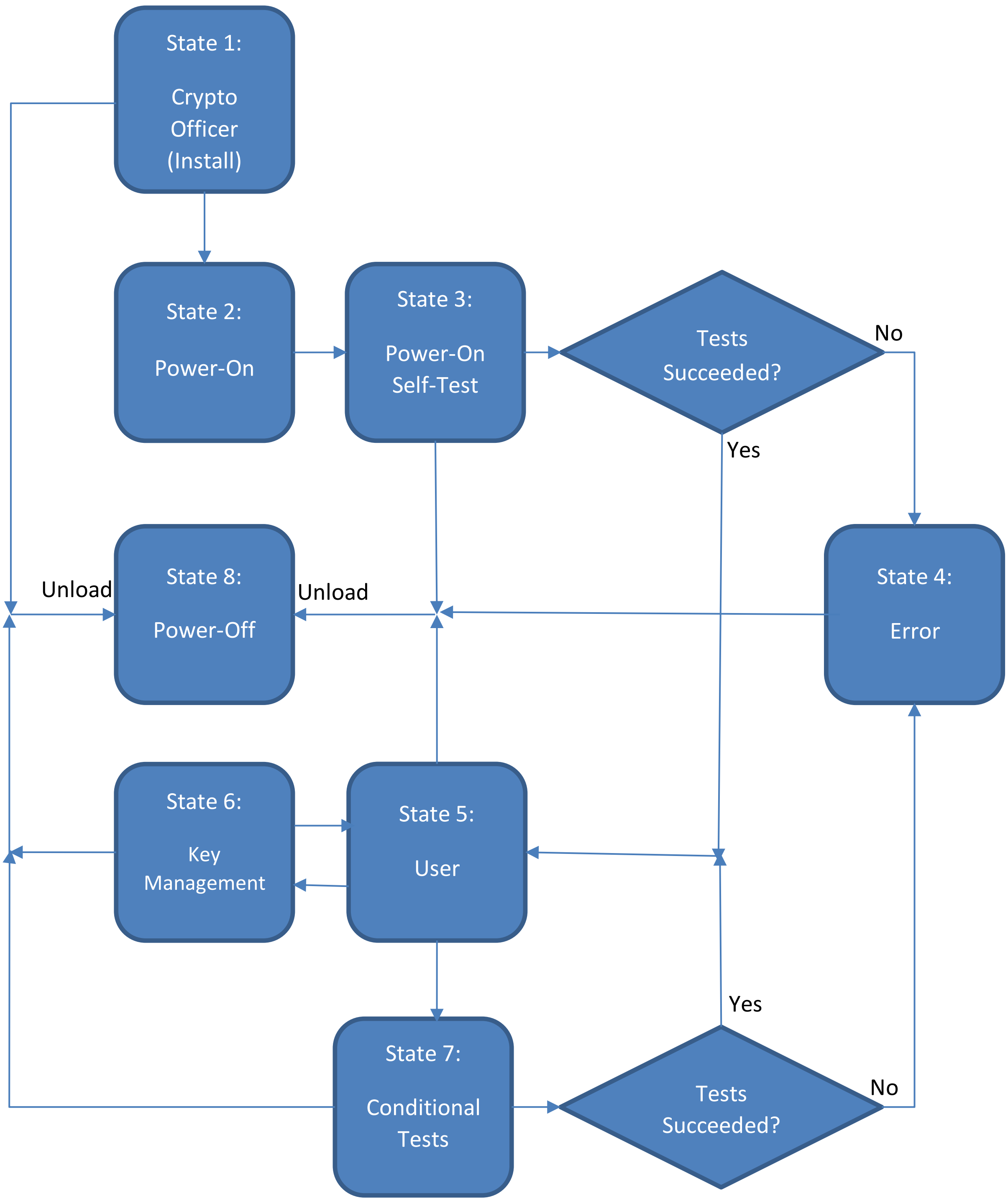
Service	Role	Access
Asymmetric encryption/decryption using non-approved RSA key size	User	Read, Execute
Symmetric encryption/decryption using non-approved algorithms	User	Read, Execute
Hash operation using non-approved algorithms	User	Read, Execute
Digital signature generation and verification using non-approved RSA, DSA, and ECDSA private key sizes	User	Read, Execute
Digital signature generation using SHA-1	User	Read, Execute
RSA key wrapping	User	Read
TLS connection using keys established by Diffie-Hellman with non-approved key sizes	User	Read, Write, Execute
TLS connection using keys established by RSA with key size less than 2048 bits	User	Read, Write, Execute
Asymmetric key generation using non-Approved RSA and DSA key sizes	User	Read, Write, Execute

3.3 Authentication

The module does not provide authentication of operators. Roles are implicitly assumed based on the services that are executed.

4 Finite State Model

The diagram on the following page presents the module's operational and error states.


State 1:
Crypto Officer (Install)
State 2:
Power-On
State 3:
Power-On Self-Test
Tests Succeeded?
No
Yes
Unload
State 8:
Power-Off
Unload
State 4:
Error
State 6:
Key Management
State 5:
User
Yes
State 7:
Conditional Tests
Tests Succeeded?
No

4.1 State Descriptions

The module has eight distinct states, as shown in the diagram above and described in the list below.

1) Crypto Officer State

In this state, the Crypto Officer is installing the cryptographic module.

2) Power-On State

The module transitions to the Power-On state when the module (shared library) is loaded into memory by a user-mode process created by the host operating system.

3) Power-On Self-Test (POST) State

After being loaded, the module enters the POST state when either (a) the process calls the FIPS_mode_set() API or (b) "fips=1" is set on the Linux kernel command line. The POST state will execute the integrity tests as well as the self-tests. Depending on the test results, the module will either enter the Error or User states.

Below is a list of errors that may occur during POST:

- FIPS_R_MODE_ALREADY_SET – "fips mode already set"
- FIPS_R_ENTROPY_INIT_FAILED – "entropy init failed"
- FIPS_R_FINGERPRINT_DOES_NOT_MATCH – "fingerprint does not match"
- FIPS_R_SELFTEST_FAILED – "fips selftest failed"
- FIPS_R_TEST_FAILURE – "test failure"

4) Error State

The POST failed or a conditional test failed. The module will terminate upon further use.

5) User State

The POST passed. The cryptographic algorithms can now be used.

6) Key Management State

The module creates Keys/CSPs used by crypto operations, on behalf of the User application.

7) Conditional Test State

The User application is performing a cryptographic operation, and the module performs conditional tests as appropriate. Conditional tests include DRBG self-tests and pair-wise consistency tests. Pair-wise consistency tests will be run during certain key management operations. If the conditional test results in a fatal error, the module will enter the error state.

8) Power-Off State

The operating system has terminated the User process and released its memory.

5 Operational Environment

The modifiable operational environment for the module is the Azure Linux operating system, running on one of the supported hardware platforms specified in the Validated Platforms section. Azure Linux is installed on a bare metal server blade or runs as a virtual machine using a hypervisor on the Azure Host computer.

5.1 Single Operator

The underlying operating system is restricted to a single operator mode of operation. The application that uses the cryptographic services is the single user of the module, even when the module is serving multiple clients.

6 Cryptographic Key Management

6.1 Random Number and Key Generation

The module provides an SP 800-90A Rev1 compliant Deterministic Random Bit Generator (DRBG) for the generation of keys and random numbers. A SP 800-90B compliant CPU time jitter RNG is used as entropy source for seeding the DRBG. The jitter RNG is implemented within module's logical boundary. An application can provide a nonce, personalization string, and/or additional input string, that are also used during seeding and reseeding the DRBG as specified by SP 800-90A Rev1. The personalization string or additional input string specified by the application is concatenated with additional personalization data obtained from the getrandom() system call. The jitter RBG provides at least 128 bits of entropy to the DRBG during initialization and reseeding, which means the reader should be aware that "the module generates cryptographic keys whose strengths are modified by available entropy."

6.2 Key and CSP Management Summary

The following table outlines the Critical Security Parameters (CSPs) used by the cryptographic services implemented in the module. All keys / CSPs are stored in RAM.

Key or CSP	Generation	Zeroization
AES Symmetric Key	Established via TLS handshake or not generated but passed in as API input parameter.	EVP_Cipher_CTX_free(), EVP_Cipher_CTX_reset()
Triple-DES Symmetric Key		
HMAC Key		HMAC_CTX_free()
CMAC Key	Not generated, passed in as API input parameter	CMAC_CTX_free()
RSA Public and Private Keys	Generated using the FIPS 186-4 standard, with a random value from DRBG.	RSA_free()
DSA Public and Private Keys		DSA_free()
ECDSA Public and Private Keys		EC_KEY_Free()
Diffie-Hellman Public and Private Keys	Generated as specified in SP 800-56A Rev3, with a random value from DRBG.	DH_Free()
EC Diffie-Hellman Public and Private Keys		EC_KEY_Free()

Shared Secret	Generated during the Diffie-Hellman or EC Diffie-Hellman shared secret computation.	DH_free(), EC_KEY_Free()
SP 800-90B DRBG entropy input	CPU time jitter RNG	FIPS_drbg_free(), RAND_DRBG_free()
TLS 1.2 Derived Key, Pre-Master Secret and Master Secret	Established during the TLS handshake, derived per SP 800-135.	SSL_free() SSL_clear()
PBKDF Password	Entered by an operator	KDF_pbkdf2_free()
PBKDF Derived Key	Derived per SP 800-132.	KDF_pbkdf2_free()
SSH-KDF Derived Keys	Derived per SP 800-135 Rev1.	KDF_sshkdf_free()
KDA OneStep Derived Key	SP 800-56C Rev2 KDF mechanism	sskdf_free()
NIST SP 800-90A Rev1 HASH_DRBG: V and C	Not generated; secret value maintained internal to the module	FIPS_drbg_free(), RAND_DRBG_free()
NIST SP 800-90A Rev1 HMAC_DRBG: V and Key	Not generated; secret value maintained internal to the module	FIPS_drbg_free(), RAND_DRBG_free()
NIST SP 800-90A Rev1 CTR_DRBG: V and Key	Not generated; secret value maintained internal to the module	FIPS_drbg_free(), RAND_DRBG_free()

6.3 Key and CSP Access

When an authorized application is the module user (the User role), it has access to all key data generated during the operation of the module. The module does not support the output of intermediate key generation values during the key generation process.

CSPs defined in an Approved mode of operation are not to be accessed or shared while in a non-Approved mode of operation. CSPs shall not be generated while in a non-approved mode.

6.4 Key and CSP Storage

Symmetric and asymmetric keys are provided to the module by the appropriate API input parameters and are destroyed when released by the appropriate API function calls. The module does not perform persistent storage of keys. The keys and CSPs are stored as plaintext in RAM.

6.5 Key and CSP Zeroization

The application that uses the module is responsible for key destruction and zeroization. The module provides API functions for key generation and destruction.

6.6 Key Establishment

The module provides Diffie-Hellman and EC Diffie-Hellman key agreement with the following security strengths:

- Diffie-Hellman provides between 112 and 200 bits of encryption strength.

- EC Diffie-Hellman provides between 112 and 256 bits of encryption strength.

The module provides AES and Triple-DES key wrapping with the following security strengths:

- AES (SP 800-38F) - using GCM, CCM, AES-KW, AES-KWP and approved block chaining modes with HMAC for authentication:
  - Between 128 and 256 bits of encryption strength.
- Triple-DES (SP 800-38F) – using approved block chaining modes with HMAC for authentication.
  - 112 bits of encryption strength.

6.7 Key and CSP Entry and Output

The module will import from, or export to, all keys or CSPs to the authorized application. When importing a key or CSP into the module, it is entered as plaintext form. When exporting a key or CSP from the module it will be in plaintext form if required by the application which requested the export.

7 Self-Tests

The module performs self-tests to ensure integrity and correct functionality. Some functions require continuous verification, such as the random number generator. The module will not perform cryptographic functions while in its self-test or error states. If the self-test fails, the module enters the error state and future cryptographic function calls fail. If the self-test passes, the module is loaded and cryptographic functions are available for use.

During module initialization, the module performs power-on tests, which start with the HMAC integrity test. No operator inputs or actions are required to run these tests. To execute the power-on tests on demand, an application has to unload the module from memory, then re-load and re-initialize it.

See section 8.2.6 for additional details on detecting possible self-test errors and recovery procedures.

7.1 Power-On Self-Tests

Algorithm	Use	Self-Test Type
AES (ECB, CCM, GCM, XTS modes)	Encryption and Decryption	Cryptographic Algorithm Self-Test, Known Answer Test (KAT) for each mode. Encryption and decryption are tested separately.
AES (CMAC)	Mac Generation and Verification	KAT
Triple-DES (ECB, CBC)	Encryption and Decryption	KAT. Encryption and decryption are tested separately.
Triple-DES (CMAC)	Mac Generation and Verification	KAT
DSA	Domain Parameters (PQG) Generation and Verification	Pair-wise Consistency Test (PCT).
DSA	Signature Generation and Verification	PCT
DSA	Key Generation	PCT

RSA	Key Generation	PCT
RSA	Signature Generation and Verification	KAT, with signature generation and verification tested separately.
ECDSA	Key Pair Generation and Verification	PCT
ECDSA	Signature Generation and Verification	PCT
SP 800-90A DRBG Rev1 (CTR, Hash, HMAC)	DRBG	KAT (Health tests specified in section 11.3 of SP 800-90A Rev1)
SHA-1, SHA-2, SHA-3	Hashing	KAT
HMAC-SHA-256	Module Integrity	KAT
HMAC (HMAC-SHA-1, -224, -256, -384, -512, -512/224, -512/256)	Message Integrity	KAT
KAS-ECC-SSC	EC Diffie-Hellman Key Agreement	Primitive “Z” computation KAT.
KAS-FFC-SSC, Safe Primes	Diffie-Hellman Key Agreement	Primitive “Z” computation KAT.
(CVL) TLS-KDF, (CVL) SSH KDF, PBKDF, KDA OneStep	Key Derivation	KAT

7.2 Conditional Tests

Algorithm	Generation
DSA	PCT, signature generation and verification.
RSA	PCT, signature generation and verification; encryption and decryption.
ECDSA	PCT, signature generation and verification.
KAS-FFC-SSC Key Agreement	Owner assurance of public key validity is implemented as specified by SP 800-56A Rev3 section 5.6.2.1.3. For public key validation, if Q is provided as part of the domain parameters, a full validation according to SP 800-56A Rev3 section 5.6.2.3.1 is performed. If Q is not provided, a partial validation according to SP 800-56A Rev3 section 5.6.2.3.2 is performed.
KAS-ECC-SSC Key Agreement	The module performs partial verification for ephemeral keys, per SP 800-56A Rev3 section 5.6.2.3.4, and full validation for other keys, per SP 800-56A rev3 section 5.6.2.3.3.
ENT	SP 800-90B Repetition Count Test and Adaptive Proportion Test as defined in sections 4.4.1 and 4.4.2 respectively.

8 Guidance

8.1 Crypto Officer Guidance

8.1.1 Module Installation

Crypto Officers use the Installation instructions to install the Module in their environment.

The version of the RPM containing the FIPS validated module is stated in section 1. The integrity of the RPM is automatically verified during the installation and the Crypto Officer shall not install the RPM file if the RPM tool indicates an integrity error.

8.1.2 Operating Environment Configuration

To configure the operating environment to support FIPS, perform the following steps.

- Install the dracut-fips package:

  ```
  dnf install dracut-fips
  ```

- Regenerate the initramfs

  ```
  mkinitrd
  ```

- Modify the mariner.cfg file:

  Append `fips=1` to variable `mariner_cmdline` in /boot/mariner.cfg.

- Reboot the system.
- Check that the file /proc/sys/crypto/fips_enabled contains 1.

8.2 User Guidance

8.2.1 TLS and Diffie-Hellman

As required by SP 800-131A Rev2, Diffie-Hellman with keys smaller than 2048 bits must not be used. However, the TLS protocol cannot enforce the support of FIPS approved Diffie-Hellman key sizes.

To enforce FIPS 140-2 compliance, the crypto officer must:

- If the module is used as a TLS server, the Diffie-Hellman parameters of "SSL_CTX_set_tmp_dh" must be 2048 bits or larger
- If the module is used as a TLS client, the TLS server must be configured to only offer keys of 2048 bits or larger

8.2.2 TLS and RSA Key Transport

As required by IG D.9, RSA key encapsulation must not be used. To enforce FIPS 140-2 compliance, the crypto officer must ensure TLS does not use the following ciphers:

- AES256-GCM-SHA384
- AES128-GCM-SHA256
- AES256-SHA256
- AES128-SHA256
- AES256-SHA
- AES128-SHA

8.2.3 AES-GCM-IV

- In case the module's power is lost and then restored, the key used for the AES GCM encryption or decryption shall be redistributed.
- The nonce_explicit part of the IV does not exhaust the maximum number of possible values for a given session key. The design of the TLS protocol in this module implicitly ensures that the nonce_explicit, or counter portion of the IV will not exhaust all of its possible values.
- The AES GCM IV generation shall only be used for the TLS protocol version 1.2, and complies with [RFC5288]. Therefore, AES GCM IV generation is compliant with Scenario 1a from [FIPS140-2_IG] IG A.5.
- When a GCM IV is used for decryption, the responsibility for the IV generation lies with the party that performs the AES GCM encryption and therefore there is no restriction on the IV generation.
- The module supports the TLS_*_GCM_* ciphersuites from SP 800-52 Rev2, section 3.3.1.

8.2.4 Triple-DES keys

According to IG A.13, the same Triple-DES key shall not be used to encrypt more than 2^{16} 64- bit blocks of data. It is the user’s responsibility to make sure that the module complies with this requirement and that the module does not exceed this limit.

8.2.5 RSA and DSA keys

The Module allows the use of 1024 bit RSA and DSA keys for legacy purposes, including signature generation. RSA must be used with either 2048, 3072 or 4096-bit keys because larger key sizes have not been CAVP tested. DSA must be used with either 2048 or 3072-bit keys because larger key sizes have not been CAVP tested. To comply with the requirements of FIPS 140-2, a user must therefore only use keys with 2048 bits or 3072 bits in FIPS Approved mode. Application can enforce the key generation bit length restriction for RSA and DSA keys by setting the environment variable OPENSSL_ENFORCE_MODULUS_BITS. This environment variable ensures that 1024-bit keys cannot be generated.

8.2.6 Handling Self-Test Errors

If a self-test fails, the module enters the error state. These errors are reported through ERR interface. The user can query information about the error using module interfaces such as ERR_get_error(), see the OpenSSL manual for additional information. While in error state, any calls to a cryptographic service of the module returns an error with the error message: 'FATAL FIPS SELFTEST FAILURE' printed to stderr and the module is terminated with the abort() call.

The only way to recover from the error state is to restart the module. For a hard error such as a failed POST, it may not be recoverable. If the restart of the module does not clear the error, the module should be reinstalled.

8.2.7 Key derivation using SP 800-132 PBKDF

The module supports option 1(a) of section 5.4 from SP 800-132: master key (or a segment of it) is used as the data protection key. The following requirements must be met:

- Derived keys shall only be used in storage applications. The Master Key (MK) shall not be used for other purposes. The length of the MK or DPK shall be of 112 bits or more.
- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the SP 800-90A DRBG.
- The iteration count shall be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users. The minimum value shall be 1000.
- Passwords or passphrases, used as an input for the PBKDF, shall not be used as cryptographic keys.
- The length of the password or passphrase shall be of at least 20 characters, and shall consist of lower-case, upper-case and numeric characters.

9 Mitigation of Other Attacks

RSA is vulnerable to timing attacks. In a setup where attackers can measure the time of RSA decryption or signature operations, blinding must be used to protect the RSA operation from that attack. The API function RSA_blinding_on() turns blinding on for the RSA key and generates a random blinding factor. The random number generator must be seeded prior to calling RSA_blinding_on().

For Weak Triple-DES keys, there is no weak key detection by default. The caller can explicitly set the DES_check_key to 1 or call DES_check_key_parity() and/or DES_is_weak_key() functions on its own.

Weak Triple-DES keys may be detected by the following code:

```
/*-
* Weak and semi weak keys as taken from
* %A D.W. Davies
* %A W.L. Price
* %T Security for Computer Networks
* %I John Wiley & Sons
* %D 1984
*/
#define NUM_WEAK_KEY    16
static const DES_cblock weak_keys[NUM_WEAK_KEY] = {
    /* weak keys */
    {0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01},
    {0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE},
    {0x1F, 0x1F, 0x1F, 0x1F, 0x0E, 0x0E, 0x0E, 0x0E},
    {0xE0, 0xE0, 0xE0, 0xE0, 0xF1, 0xF1, 0xF1, 0xF1},
    /* semi-weak keys */
    {0x01, 0xFE, 0x01, 0xFE, 0x01, 0xFE, 0x01, 0xFE},
    {0xFE, 0x01, 0xFE, 0x01, 0xFE, 0x01, 0xFE, 0x01},
    {0x1F, 0xE0, 0x1F, 0xE0, 0x0E, 0xF1, 0x0E, 0xF1},
```

```
    {0xE0, 0x1F, 0xE0, 0x1F, 0xF1, 0x0E, 0xF1, 0x0E},
    {0x01, 0xE0, 0x01, 0xE0, 0x01, 0xF1, 0x01, 0xF1},
    {0xE0, 0x01, 0xE0, 0x01, 0xF1, 0x01, 0xF1, 0x01},
    {0x1F, 0xFE, 0x1F, 0xFE, 0x0E, 0xFE, 0x0E, 0xFE},
    {0xFE, 0x1F, 0xFE, 0x1F, 0xFE, 0x0E, 0xFE, 0x0E},
    {0x01, 0x1F, 0x01, 0x1F, 0x01, 0x0E, 0x01, 0x0E},
    {0x1F, 0x01, 0x1F, 0x01, 0x0E, 0x01, 0x0E, 0x01},
    {0xE0, 0xFE, 0xE0, 0xFE, 0xF1, 0xFE, 0xF1, 0xFE},
    {0xFE, 0xE0, 0xFE, 0xE0, 0xFE, 0xF1, 0xFE, 0xF1}
};
```

10 Security Levels

The security level for each FIPS 140-2 security requirement is given in the following table.

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

11 Additional Details

For more information about FIPS 140 validations of Microsoft products, please see:

https://docs.microsoft.com/en-us/windows/security/threat-protection/fips-140-validation

12 Glossary and Abbreviations

- **AES:** Advanced Encryption Standard
- **CAVP:** Cryptographic Algorithm Validation Program
- **CSP:** Critical Security Parameter
- **DES:** Data Encryption Standard
- **DRBG:** Deterministic Random Bit Generator
- **DSA:** Digital Signature Algorithm
- **ECB:** Electronic Codebook
- **HMAC:** Hash Message Authentication Code
- **OS:** Operating System
- **RNG:** Random Number Generator

- **RSA:** Rivest, Shamir, Adleman
- **SHA:** Secure Hash Algorithm
- **SHS:** Secure Hash Standard

13 References

- **FIPS 140-2, Security Requirements for Cryptographic Modules**, https://csrc.nist.gov/publications/detail/fips/140/2/final
- **FIPS 180-4, Secure Hash Standard (SHS)**, https://csrc.nist.gov/publications/detail/fips/180/4/final
- **FIPS 186-4, Digital Signature Standard (DSS)**, https://csrc.nist.gov/publications/detail/fips/186/4/final
- **FIPS 197, Advanced Encryption Standard (AES)**, https://csrc.nist.gov/publications/detail/fips/197/final
- **FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC)**, https://csrc.nist.gov/publications/detail/fips/198/1/final
- **FIPS 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions**, https://csrc.nist.gov/publications/detail/fips/202/final
- **SP 800-38A, Recommendation for Block Cipher Modes of Operation: Methods and Techniques**, https://csrc.nist.gov/publications/detail/sp/800-38a/final
- **SP 800-38B, Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication**, https://csrc.nist.gov/publications/detail/sp/800-38b/final
- **SP 800-38C, Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality**, https://csrc.nist.gov/publications/detail/sp/800-38c/final
- **NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC**, https://csrc.nist.gov/publications/detail/sp/800-38d/final
- **NIST SP 800-38E, Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices**, https://csrc.nist.gov/publications/detail/sp/800-38e/final
- **NIST SP 800-38F, Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping**, https://csrc.nist.gov/publications/detail/sp/800-38f/final
- **NIST SP 800-52 Rev.2, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations**, https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/final
- **NIST SP 800-56A Rev. 3, Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography**, https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final
- **NIST SP 800-56C Rev. 2, Recommendation for Key-Derivation Methods in Key-Establishment Schemes**, https://csrc.nist.gov/publications/detail/sp/800-56c/rev-2/final
- **NIST SP 800-67 Rev. 2, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher**, https://csrc.nist.gov/publications/detail/sp/800-67/rev-2/final

- **NIST SP 800-90A Rev. 1, Recommendation for Random Number Generation Using Deterministic Random Bit Generators**, https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final
- **NIST SP 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation**, https://csrc.nist.gov/publications/detail/sp/800-90b/final
- **NIST SP 800-131A Rev. 2, Transitioning the Use of Cryptographic Algorithms and Key Lengths**, https://csrc.nist.gov/publications/detail/sp/800-131a/rev-2/final
- **NIST SP 800-132, Recommendation for Password-Based Key Derivation: Part 1: Storage Applications**, https://csrc.nist.gov/publications/detail/sp/800-132/final
- **NIST SP 800-135 Rev. 1, Recommendation for Existing Application-Specific Key Derivation Functions**, https://csrc.nist.gov/publications/detail/sp/800-135/rev-1/final
- **ANSI X9.31, Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)**, https://standards.globalspec.com/std/1955293/ANSI%20X9.31