Amazon Linux 2 OpenSSL Cryptographic Module

Module Version 1.0

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.1

Last update: 2019-10-09

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of Contents

List of Tables

List of Figures

Copyrights and Trademarks

Amazon is a registered trademark of Amazon Web Services, Inc. or its affiliates.

1 Introduction

This document is the non-proprietary FIPS 140-2 Security Policy for version 1.0 of the Amazon Linux 2 OpenSSL Cryptographic Module. It contains the security rules under which the module must be operated and describes how this module meets the requirements as specified in FIPS 140-2 (Federal Information Processing Standards Publication 140-2) for a Security Level 1 module.

1.1 Purpose of the Security Policy

There are three major reasons that a security policy is needed:

- It is required for FIPS 140 2 validation,
- It allows individuals and organizations to determine whether a cryptographic module, as implemented, satisfies the stated security policy, and
- It describes the capabilities, protection and access rights provided by the cryptographic module, allowing individuals and organizations to determine whether it will meet their security requirements.

1.2 Target Audience

This document is part of the package of documents that are submitted for FIPS 140 2 conformance validation of the module. It is intended for the following audience:

- Developers.
- FIPS 140-2 testing lab.
- The Cryptographic Module Validation Program (CMVP).
- Customers using or considering integration of Amazon Linux 2 OpenSSL Cryptographic Module.

2 Cryptographic Module Specification

2.1 Module Overview

The Amazon Linux 2 OpenSSL Cryptographic Module (hereafter referred to as the “module”) is a software module supporting FIPS 140-2 Approved cryptographic algorithms within Amazon Linux 2 OpenSSL Cryptographic Module. The module provides a C language application program interface (API) for use by other processes that require cryptographic functionality.

2.2 FIPS 140-2 Validation Scope

Table 1 shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard.

Table 1: FIPS 140-2 Security Requirements.

Security Requirements Section		Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles and Services and Authentication	1
4	Finite State Machine Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	1
Overall Level		1

2.3 Definition of the Cryptographic Module

The Amazon Linux 2 OpenSSL Cryptographic Module is defined as a Multi-chip Standalone module per the requirements within FIPS 140-2. The logical cryptographic boundary of the module consists of shared library files and their integrity test HMAC files, which are delivered through the Amazon Linux 2 yum core repository (ID amz2-core/2/x86_64) from the RPM file with version openssl-1.0.2k-16.amzn2.0.3.x86_64.

- /usr/lib64/.libcrypto.so.1.0.2k.hmac (64 bits)
- /usr/lib64/.libssl.so.1.0.2k.hmac (64 bits)
- /usr/lib64/libcrypto.so.1.0.2k (64 bits)
- /usr/lib64/libssl.so.1.0.2k (64 bits)

The module instantiation is provided by the dracut-fips RPM package with the file version of dracut-033-535.amzn2.x86_64.

The AES-NI configuration of the kernel is provided by the dracut-fips RPM package with the file version of dracut-fips-aesni-033-502.amzn2.1.x86_64

The Amazon Linux 2 OpenSSL Cryptographic Module package includes the binary files, integrity check HMAC files, Man Pages and the OpenSSL engines provided by the standard OpenSSL shared library. The OpenSSL engines and their shared object files are not part of the module, and therefore they must not be used when operating the module.

The module shall be instantiated by the dracut-fips package with the RPM file version specified above. The dracut-fips RPM package is only used for the installation and instantiation of the module. This code is not active when the module is operational and does not provide any services to users interacting with the module. Therefore, the dracut-fips RPM package is outside the module's logical boundary.

Figure 1 shows the logical block diagram of the module executing in memory on the host system. The logical cryptographic boundary is indicated with a dashed colored box.

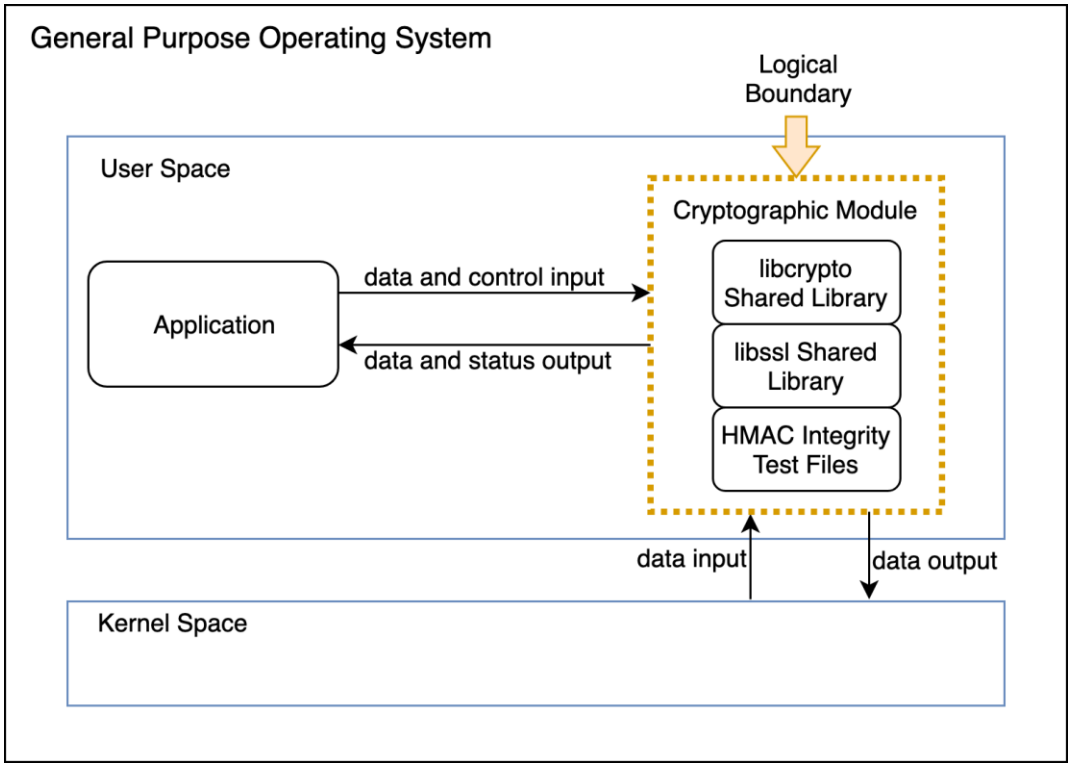

Figure 1: Logical cryptographic boundary.

2.4 Definition of the Physical Cryptographic Boundary

The physical cryptographic boundary of the module is defined as the hard enclosure of the host system on which the module runs. Figure 2 depicts the hardware block diagram. The physical hard enclosure is indicated by the dashed colored line. No components are excluded from the requirements of FIPS 140-2.

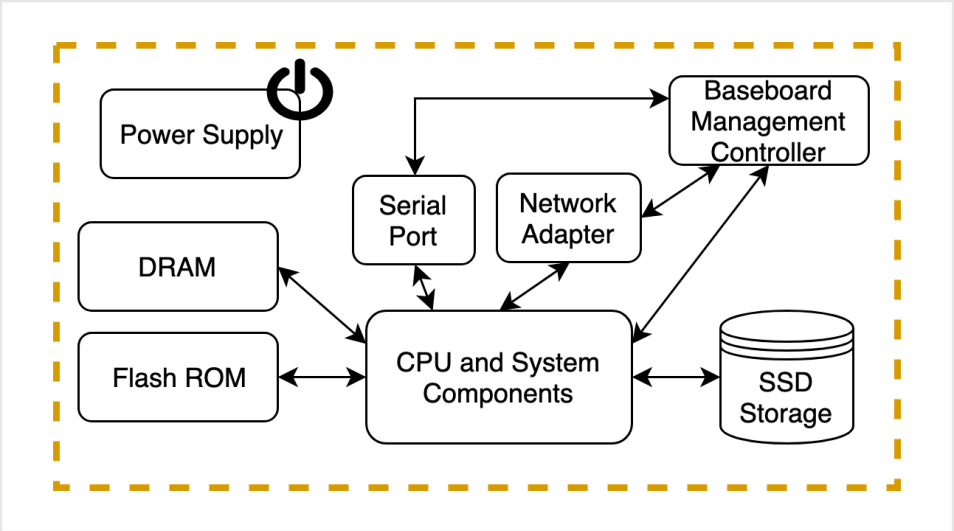

Figure 2: Hardware block diagram.

2.5 Tested Environments

The module was tested on the environments/platforms listed in Table 2. The tested operational environment is not a virtualized cloud service, and was controlled such that the laboratory had full and exclusive access to the environment and module during the testing procedures.

Table 2: Tested operational environments.

Operating System	Processor	Hardware
Amazon Linux 2	Intel Xeon E5-2686 (Broadwell) x86_64bit with PAA (i.e., AES-NI)	Amazon EC2 i3.metal 512 GiB system memory 13.6 TiB SSD storage + 8 GiB SSD boot disk 25 Gbps Elastic Network Adapter
Amazon Linux 2	Intel Xeon E5-2686 (Broadwell) x86_64bit without PAA (i.e., AES-NI)	Amazon EC2 i3.metal 512 GiB system memory 13.6 TiB SSD storage + 8 GiB SSD boot disk 25 Gbps Elastic Network Adapter

2.6 Modes of Operation

The module supports two modes of operation.

- In "**FIPS mode**" (the Approved mode of operation), only approved or allowed security functions with sufficient security strength are offered by the module.
- In "**non-FIPS mode**" (the non-Approved mode of operation), non-approved security functions are offered by the module.

The module enters the operational mode after Power-On Self-Tests (POST) succeed. Once the module is operational, the mode of operation is implicitly assumed depending on the security

function invoked and the security strength[1] of the cryptographic keys chosen for the function or service.

If the POST or the Conditional Tests fail (Section 9), the module goes into the error state. The status of the module can be determined by the availability of the module. If the module is available, then it had passed all self-tests. If the module is unavailable, it is because any self-test failed, and the module has transitioned to the error state.

Keys and Critical Security Parameters (CSPs) used or stored in FIPS mode shall not be used in non-FIPS mode, and vice versa.

[1] See Section 5.6.1 in [SP800-57] for a definition of “security strength”.

3 Module Ports and Interfaces

As a Software module, the module does not have physical ports. The operator can only interact with the module through the API provided by the module. Thus, the physical ports within the physical boundary are interpreted to be the physical ports of the hardware platform on which the module runs and are directed through the logical interfaces provided by the software.

The logical interfaces are the API through which applications request services and receive output data through return values or modified data referenced by pointers. Table 3 summarizes the logical interfaces.

Table 3: Ports and interfaces.

Logical Interface	Description
Data Input	API input parameters for data.
Data Output	API output parameters for data.
Control Input	API function calls.
Status Output	API return codes, error message.
Power Input	Not applicable for the Software module.

4 Roles, Services and Authentication

4.1 Roles

The module supports the following roles:

- **User role**: performs all services (in both FIPS mode and non-FIPS mode of operation), except module installation and configuration. This role is assumed by the calling application accessing the module.
- **Crypto Officer role**: performs module installation and configuration.

The User and Crypto Officer roles are implicitly assumed depending on the service requested.

4.2 Services

The module provides services to calling applications that assume the user role, and human users assuming the Crypto Officer role. Table 4 and Table 5 depict all services, which are described with more detail in the user documentation.

The tables use the following convention when specifying the access permissions that the module has for each CSP or key.

- **Create (C):** the calling application can create a new CSP.
- **Read (R):** the calling application can read the CSP.
- **Update (U):** the calling application can write a new value to the CSP.
- **Zeroize (Z):** the calling application can zeroize the CSP.
- **N/A:** the calling application does not access any CSP or key during its operation.

For the “Role” column, U indicates the User role, and CO indicates the Crypto Officer role. An X marks which role has access to that service.

4.2.1 Services in the FIPS-Approved Mode of Operation

Table 4 provides a full description of FIPS Approved services and the non-Approved but Allowed services provided by the module in the FIPS-approved mode of operation and lists the roles allowed to invoke each service.

Table 4: Services in the FIPS-approved mode of operation.

Service	Service Description and Algorithms	Role		Keys and CSPs	Access Types
		U	CO		
Symmetric Encryption/Decryption	Encrypts or decrypts a block of data using AES.	X		AES or Triple-DES Key	R
Symmetric Encryption/Decryption	Encrypts or decrypts a block of data using 3-Key Triple-DES.	X		AES or Triple-DES Key	R
RSA Key Generation	Generate RSA asymmetric keys using DRBG.	X		RSA public/private keys	C, R, U

Service	Service Description and Algorithms	Role		Keys and CSPs	Access Types
		U	CO		
DSA Key Generation	Generate DSA asymmetric keys using DRBG.	X		DSA public/private keys	C, R, U
ECDSA Key Generation	Generate ECDSA asymmetric keys using DRBG.	X		ECDSA public/private keys	C, R, U
RSA Digital Signature Generation and Verification	Sign and verify signature operations for RSA PCKS#1v1.5 and X9.31.	X		RSA public/private keys	R
DSA Digital Signature Generation and Verification	Sign and verify signature operations for DSA.	X		DSA public/private	R
ECDSA Digital Signature Generation and Verification	Sign and verify signature for ECDSA.	X		ECDSA public/private keys	R
TLS Network Protocol v1.0, v1.1, v1.2	Provide data encryption and authentication over TLS network protocol with AES, Triple-DES, HMAC.	X		AES key Triple-DES key HMAC key Pre-master secret Master secret Derived key Diffie-Hellman public/private key pair EC Diffie-Hellman public/private key pair RSA, DSA, ECDSA public/private keys	C, R, U
TLS Key Derivation	Establish a TLS secure channel. KDF (PRF) in TLS v1.0/1.1, TLS v1.2.	X		Shared secret (pre-master secret), master secret, derived key	C, R, U
Diffie-Hellman Key Agreement	Establish a shared secret. KAS FFC	X		Diffie-Hellman public/private keys	C, R, U
EC Diffie-Hellman Key Agreement	Establish a shared secret. KAS ECC	X		EC Diffie-Hellman public/private keys	C, R, U
Key Wrapping with RSA	Encapsulates and decapsulates a key using RSA encrypt/decrypt primitives	X		RSA public/private keys	R

Service	Service Description and Algorithms	Role		Keys and CSPs	Access Types
		U	CO		
Key Wrapping with Symmetric Algorithms	Wrap and unwrap keys with AES-KW, AES-KWP, AES-GCM, AES-CCM	X		AES keys	R
Certificate Management	Management of key properties within certificates.	X		RSA, DSA, and ECDSA public/private keys associated to an X.509 certificate	R, U
Message Authentication Code (MAC)	Authenticate and verify authentication of data using HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	X		HMAC Key	R
	Authenticate and verify authentication of data using CMAC with AES.	X		AES Key	R
	Authenticate and verify authentication of data using CMAC with Triple-DES.	X		Triple-DES Key	R
Message Digest	Hash a block of data with SHS. SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	X		None	N/A
Random Number Generation	Generate random numbers based on the SP 800-90A DRBG.	X		Entropy input string and internal state	C, R, U
Other FIPS-related Services					
Show Status	Show status of the module state	X		None	N/A
Self-Test	Initiate power-on self-tests	X		None	N/A
Zeroize	Zeroize all critical security parameters	X		All keys and CSPs	Z
Module Installation	Installation of the module		X	None	N/A
Module Configuration	Configuration of the module		X	None	N/A

4.2.2 Services in the Non-FIPS-Approved Mode of Operation

Table 5 presents the services only available in non-FIPS-approved mode of operation.

Table 5: Services in the non-FIPS approved mode of operation.

Service	Service Description and Algorithms	Role		Keys and CSPs	Access Types
		U	CO		
RSA key wrapping	Encrypts or decrypts using non-Approved RSA key size	X		RSA key pair	C, R, U
Symmetric Encryption/Decryption	Encrypts or decrypts using non-Approved algorithms	X		Camellia, CAST, DES, IDEA, RC2, RC4, RC5 keys	R
Digital Signature Generation and Verification	Sign or verify operations with non-Approved RSA or DSA key lengths	X		RSA key < 2048 DSA keys not listed in Table 6 Signature Generation with SHA-1	C, R, U
TLS Key Exchange	Negotiate a TLS key agreement secure channel with non-Approved key sizes or curves	X		RSA/ Diffie-Hellman key < 2048 EC Diffie-Hellman with P-192	C, R, U
Diffie-Hellman Key Agreement	Establish a shared secret. KAS FFC	X		Diffie-Hellman key < 2048	C, R, U
EC Diffie-Hellman Key Agreement	Establish a shared secret. KAS ECC	X		EC Diffie-Hellman with P-192	C, R, U
Asymmetric Key Generation	Generation of non-Approved RSA and DSA keys	X		RSA key < 2048 DSA keys not listed in Table 6	C, R, U
Random Number Generation	Generation of random numbers using the ANSI X9.31 PRNG	X		seed, seed key	C, R, U
Message Digest	Hashing using non-Approved hash functions that include MD2, MD4, MD5, MDC2, RIPEMD, Whirlpool	X		None	N/A
J-PAKE Key Agreement	Password authenticated key agreement using J-PAKE	X		J-PAKE key pair	C, R, U

4.3 Algorithms

The module implements cryptographic algorithms that are used by the services provided by the module. The cryptographic algorithms that are approved to be used in the FIPS mode of operation are tested and validated by the CAVP. No parts of the Transport Layer Security (TLS) protocol, other than the key derivation function, have been tested by the CAVP or by the CMVP.

The module provides assembler implementations for AES and SHS; support for the AVX2, AVX, AESNI and SSSE3 instruction sets from the CPU (per the tested operational environment in Table 2) for AES and SHS; and C-language generic implementations for the rest of the algorithms.

Table 6, Table 7 and Table 8 present the cryptographic algorithms in specific modes of operation. These tables include the CAVP certificates for different implementations, the algorithm name, respective standards, the available modes and key sizes wherein applicable, and usage. Information from certain columns may be applicable to more than one row.

4.3.1 FIPS-Approved

Table 6 lists the cryptographic algorithms that are approved to be used in the FIPS mode of operation.

Table 6: FIPS-approved cryptographic algorithms.

Algorithm	Standard	Mode/Method	Key size	Use	CAVP Cert#
AES	[FIPS197] [SP800-38A]	CBC, CFB1, CFB8, CFB128, CTR, ECB, OFB	128, 192 and 256 bits	Data Encryption and Decryption	#C523 (AESNI) #C524 (AES assembler) #C525 (AES Vector Perm.)
	[FIPS197] [SP800-38B]	CMAC	128, 192 and 256 bits	MAC Generation and Verification	
	[FIPS197] [SP800-38C]	CCM	128, 192 and 256 bits	Data Encryption and Decryption	
	[FIPS197] [SP800-38D]	GCM	128, 192 and 256 bits	Data Encryption and Decryption	
	[FIPS197] [SP800-38E]	XTS	128 and 256 bits	Data Encryption and Decryption	
	[FIPS197] [SP800-38F]	KW, KWP	128, 192 and 256 bits	Key Wrapping and Unwrapping	
DSA	[FIPS 186-4]	SHA-224, SHA-256	L=2048, N=224; L=2048, N=256; L=3072, N=256	Key Pair Generation. Domain Parameter Generation	#C523 (C Implementation)
		SHA-224, SHA-256, SHA-384, SHA-512	L=2048, N=224	Signature Generation	

Algorithm	Standard	Mode/Method	Key size	Use	CAVP Cert#
		SHA-256, SHA-384, SHA-512	L=2048, N=256; L=3072, N=256	Signature Generation	
		SHA-224, SHA-256	L=2048, N=224; L=2048, N=256; L=3072, N=256	Domain Parameter Verification	
		SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	L=1024, N=160; L=2048, N=224; L=2048, N=256; L=3072, N=256	Signature Verification	
DRBG	[SP800-90A]	Hash_DRBG SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) with/without PR	n/a	Random Number Generation	#C523 (SHA AVX2) #C524 (SHA assembler) #C525 (SHA SSSE3) #C526 (SHA AVX)
		HMAC_DRBG HMAC with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 with/without PR	n/a	Random Number Generation	
		CTR_DRBG AES128, AES192, AES256 with/without DF, with/without PR	n/a	Random Number Generation	#C523 (AESNI) #C524 (AES assembler) #C525 (AES Vector Perm.)
ECDSA	[FIPS186-4]		P-256, P-384, P-521	Key Pair Generation	#C523 (C Implementation)
		SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Signature Generation	
			P-256, P-384, P-521	Public Key Verification	

Algorithm	Standard	Mode/Method	Key size	Use	CAVP Cert#
		SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Signature Verification	
KAS FFC Component	[SP800-56A]	FFC dhEphem scheme	p=2048, q=224 (FB); p=2048, q=256 (FC)	Diffie-Hellman Key Agreement	#C523 (C Implementation)
KAS ECC Component	[SP800-56A]	ECC Ephemeral Unified scheme	P-256, P-384, P-521	EC Diffie-Hellman Key Agreement	#C523 (C Implementation)
HMAC	[FIPS198-1]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 bits or greater	Message Authentication Code	#C523 (SHA AVX2) #C524 (SHA assembler) #C525 (SHA SSSE3) #C526 (SHA AVX)
KDF(PRF) in TLS v1.0/1.1 TLS v1.2	[SP800-135]	SHA-256, SHA-384		Key Derivation	#C523 (C Implementation)
RSA	[FIPS186-4]	B.3.3 Random Probably Primes	2048 and 3072 bits	Key Pair Generation	#C523 (C Implementation)
		X9.31 with SHA-256, SHA-384, SHA-512	2048 and 3072 bits	Digital Signature Generation	
		PKCS#1v1.5 and PSS with SHA-224, SHA-256, SHA-384, SHA-512	2048 and 3072 bits		
		X9.31 with SHA-1, SHA-256, SHA-384, SHA-512	1024, 2048, and 3072 bits	Signature Verification	

Algorithm	Standard	Mode/Method	Key size	Use	CAVP Cert#
		PKCS#1v1.5 and PSS with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024, 2048, and 3072 bits		
	[FIPS186-2]	X9.31 with SHA-256, SHA-384, SHA-512	4096 bits	Digital Signature Generation	
		PKCS#1v1.5 and PSS with SHA-224, SHA-256, SHA-384, SHA-512	4096 bits		
SHS	[FIPS180-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512		Message Digest	#C523 (SHA AVX2) #C524 (SHA assembler) #C525 (SHA SSSE3) #C526 (SHA AVX)
Triple-DES	[SP800-67] [SP800-38A]	ECB, CBC, CTR, CFB1, CFB8, CFB64, OFB	192 bits	Data Encryption and Decryption	#C523 (C Implementation)
	[SP800-67] [SP800-38B]	CMAC	192 bits	MAC Generation and Verification	

4.3.2 Non-Approved-but-Allowed

Table 7 lists the non-Approved-but-Allowed cryptographic algorithms provided by the module that are allowed to be used in the FIPS mode of operation.

Table 7: Non-Approved-but-allowed cryptographic algorithms.

Algorithm	Usage
RSA Key Wrapping with key size between 2048 bits and 15360 bits or more	Key wrapping, key establishment methodology provides between 112 and 256 bits of encryption strength.

Algorithm	Usage
Diffie-Hellman with key size between 2048 bits and 10000 bits	Key agreement, key establishment methodology provides between 112 and 220 bits of encryption strength.
EC Diffie-Hellman with P-256, P-384, P-521 curves	Key agreement, key establishment methodology provides between 128 and 256 bits of encryption strength.
NDRNG	Used for seeding NIST SP 800-90A DRBG.
MD5	Message digest used in TLS only

4.3.3 Non-Approved

Table 8 lists the cryptographic algorithms that are not allowed to be used in the FIPS mode of operation. Use of any of these algorithms (and corresponding services in Table 5) will implicitly switch the module to the non-Approved mode.

Table 8: Non-FIPS approved cryptographic algorithms.

Algorithm	Usage
ANSI X9.31 RNG	Random Number Generation
Camellia	Encryption/Decryption
CAST	Encryption/Decryption
DES	Encryption/Decryption
Diffie-Hellman	Key agreement using keys of length less than 2048 bits
DSA	Parameter/Key generation/Signature generation with keys not listed in Table 6
EC Diffie-Hellman	Key agreement using curves not listed in Table 6 or Table 7
ECDSA	Key generation/Signature generation with curves not listed in Table 6
IDEA	Encryption/Decryption
J-PAKE	Password Authenticated Key Exchange
MD2	Hash Function
MD4	Hash Function
MDC2	Hash Function
RC2	Encryption/Decryption
RC4	Encryption/Decryption
RC5	Encryption/Decryption
RIPEMD	Hash Function

Algorithm	Usage
RSA	Key generation/Signature generation with keys of length less than 2048 bits
SHA-1	Signature generation
Whirlpool	Hash Function

4.4 Operator Authentication

The module does not support operator authentication mechanisms. The role of the operator is implicitly assumed based on the service requested.

5 Physical Security

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

6 Operational Environment

6.1 Applicability

The module operates in a modifiable operational environment per FIPS 140-2 Security Level 1 specifications. The module runs on the Amazon Linux 2 operating system executing on the hardware specified in Section 2.5.

6.2 Policy

The operating system is restricted to a single operator mode of operation (i.e., concurrent operators are explicitly excluded by the operating system).

The application that makes calls to the modules is the single user of the modules, even when the application is serving multiple clients.

7 Cryptographic Key Management

Table 9 summarizes the keys and other CSPs that are used by the cryptographic services implemented in the module.

Table 9: Lifecycle of keys and other Critical Security Parameters (CSPs).

Name	Use	Generation	Entry and Output
AES Key	Encryption, decryption. MAC generation and verification for CMAC. Key wrapping.	Provided by the calling application, or derived during TLS handshake using SP800-135 KDF.	Entered via API input parameter. No output.
Triple-DES Keys	Encryption, decryption. MAC generation and verification for CMAC.	Provided by the calling application, or derived during TLS handshake using SP800-135 KDF.	Entered via API input parameter. No output.
HMAC Key	MAC generation and verification	Provided by the calling application, or derived during TLS handshake SP800-135 KDF.	Entered via API input parameter. No output.
RSA public and private key	RSA signature generation and verification. Key wrapping.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG.	Entered via API input parameter or generated by module. Output via API output parameters in plaintext.
DSA key pair	DSA signature generation and verification.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG.	Entered via API input parameter or generated by module. Output via API output parameters in plaintext.
ECDSA key pair	ECDSA signature generation and verification.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG.	Entered via API input parameter or generated by module. Output via API output parameters in plaintext.
Diffie-Hellman key pair	Key agreement.	Keys are generated using FIPS 186-4 and the random value used in the key generation is obtained from SP800-90A DRBG.	N/A
EC Diffie-Hellman key pair	Key agreement.		N/A
Shared secret (pre-master secret)	Establishment of encrypted session.	Generated during the key agreement when using Diffie-Hellman or	Entry: if received by module as TLS server, wrapped with server’s

Name	Use	Generation	Entry and Output
		EC Diffie-Hellman key exchange. Generated by TLS client as output from DRBG when using RSA key exchange.	public RSA key; otherwise no entry. Output: if generated by module as TLS client, wrapped with server's public RSA key; otherwise, no output.
Master secret	Establishment of encrypted session.	Derived from pre-master secret.	N/A
Entropy input string (seed)	Entropy input strings used as seed to the DRBG.	Obtained from NDRNG.	N/A
DRBG Internal state (V, C, Key)	V and key are used internally by HMAC and CTR DRBGs. V and C are used internally by HASH DRBG. Used to generate random bits.	During DRBG initialization.	N/A
RSA, ECDSA, DSA private key associated to an X.509 certificate, and X.509 certificates	Client and server authentication during TLS exchange.	N/A. Provided by the calling application.	Entered via API parameters. The certificate can exit the module via TLS protocol.

7.1 Random Number Generation

The module provides a DRBG compliant with [SP800-90A] for the creation of key components of asymmetric keys, and random number generation. The DRBG implements a Hash_DRBG, CTR_DRBG, and HMAC_DRBG mechanisms. The DRBG is initialized during module initialization and seeded from the NDRNG from /dev/urandom. The NDRNG is provided by the operational environment (i.e., Linux RNG), which is within the module's physical boundary but outside of the module's logical boundary. The NDRNG provides at least 256 bits of entropy to the DRBG.

The module performs continuous random number generator tests (CRNGT) on the output of SP800-90A DRBG to ensure that consecutive random numbers do not repeat. The operational environment, the Linux RNG, performs the continuous test on the NDRNG.

7.2 Key Generation

For generating RSA, DSA, ECDSA, Diffie-Hellman and EC Diffie-Hellman keys, the module implements asymmetric key generation services compliant with [FIPS186-4] and using a DRBG compliant with [SP800-90A]. The random value used in asymmetric key generation is obtained from the DRBG. In accordance with FIPS 140-2 IG D.12, the cryptographic module performs Cryptographic Key Generation (CKG) for asymmetric keys as per SP800-133 (vendor affirmed).

Symmetric keys are derived from the shared secret established by Diffie-Hellman and EC Diffie-Hellman in a manner that is compliant to NIST SP 800-135 for TLS KDF.

The module does not support manual key entry or intermediate key generation output. In addition, the module does not produce key output outside its physical boundary. The keys can be entered or output from the module in plaintext form via API parameters, to and from the calling application only.

7.3 Key/CSP Storage

Public and private 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.

7.4 Key/CSP Zeroization

The application is responsible for calling the appropriate destruction functions from the OpenSSL API. The destruction functions then overwrite the memory occupied by keys with zeros and deallocates the memory with the free() call. In case of abnormal termination, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

7.5 Key Establishment

The module provides Diffie-Hellman and EC Diffie-Hellman key agreement schemes from [SP800-56A]. These key agreement schemes are used as part of the TLS protocol key exchange. The module provides AES key wrapping per [SP800-38F] and RSA key wrapping (encapsulation) using public key encryption and private key decryption primitives as allowed by [FIPS140-2_IG] D.9. RSA key wrapping may be used as part of the TLS protocol key exchange.

The module provides approved key transport methods according to IG D.9. The key transport methods are provided either by:

- Using an approved key wrapping algorithm (e.g., AES-KW).
- An approved authenticated encryption mode (e.g., AES-GCM, AES-CCM).
- A combination method that occurs exclusively within the context of a TLS protocol connection, using TLS ciphersuites. The combination method consists of using an approved symmetric encryption mode (e.g., AES-128-CBC, AES-256-CBC, or Triple-DES CBC) together with an approved authentication method (HMAC), again within the context of the TLS protocol ciphersuites.

Table 6 and Table 7 specify the key sizes allowed in the FIPS mode of operation. According to “Table 2: Comparable strengths” in [SP800-57], the key sizes of key wrapping and transport, RSA, Diffie-Hellman and EC Diffie-Hellman provide the following security strengths:

- AES key wrapping provides between 128 and 256 bits of encryption strength.
- RSA key wrapping provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman key establishment methodology provides between 112 and 220 bits of encryption strength.
- EC Diffie-Hellman key establishment methodology provides between 128 and 256 bits of encryption strength.
- Approved authenticated encryption mode key establishment methodology (AES-GCM, AES-CCM) provides between 128 and 256 bits of encryption strength.
- Combination of approved AES-128 and AES-256 encryption and HMAC authentication key establishment methodology provides 128 or 256 bits of encryption strength.

- Combination of approved Triple-DES encryption and HMAC authentication key establishment methodology provides 112 bits of encryption strength.

8 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test platforms listed in Table 2 have been tested and found to conform to the EMI/EMC requirements specified by 47 Code of Federal Regulations, FCC PART 15, Subpart B, Unintentional Radiators, Digital Devices, Class A (i.e., Business use). These devices are designed to provide reasonable protection against harmful interference when the devices are operated in a commercial environment.

9 Self-Tests

9.1 Power-Up Self-Tests

The module performs power-up self-tests (POSTs) automatically during loading of the module by making use of default entry point (DEP). These POSTs ensure that the module is not corrupted and that the cryptographic algorithms work as expected. No operator intervention is necessary to run the POSTs.

While the module is executing the POSTs, services are not available, and input and output are inhibited. The module is not available for use until successful completion of the POSTs.

The integrity of the module binary is verified using an HMAC-SHA-256. The HMAC value is computed at build time and stored in the .hmac file. The value is recalculated at runtime and compared against the stored value in the file. If the comparison succeeds, then the remaining POSTs (consisting of the algorithm-specific Known Answer Tests) are performed. On successful completion of the all the power-up tests, the module becomes operational and crypto services are then available. If any of the tests fails, the module transitions to the error state and subsequent calls to the module will fail. Thus, in the error state, no further cryptographic operations will be possible.

Table 10 details the self-tests that are performed on the FIPS-approved cryptographic algorithms supported in the FIPS-approved mode of operation, using the Known-Answer Tests (KATs) and Pairwise Consistency Tests (PCTs).

Table 10: Self-tests.

Algorithm	Test
AES	• KAT AES(ECB) with 128-bit key, encryption • KAT AES(ECB) with 128-bit key, decryption • KAT AES(CCM) with 192-bit key, encryption • KAT AES(CCM) with 192-bit key, decryption • KAT AES(GCM) with 256-bit key, encryption • KAT AES(GCM) with 256-bit key, decryption • KAT AES(CMAC) with 128-bit, 192-bit and 256-bit key • KAT AES(XTS) with 128-bit and 256-bit keys, encryption • KAT AES(XTS) with 128-bit and 256-bit keys, decryption
Triple-DES	• KAT Triple-DES (ECB) with 192-bit key, encryption • KAT Triple-DES (ECB) with 192-bit key, decryption • KAT Triple-DES with 192-bit key (CMAC)
DSA	• PCT DSA with L=2048, N=224 and SHA-256
RSA	• KAT RSA PKCS#1v1.5 signature generation and verification with 2048-bit key and using SHA-224, SHA-256, SHA-384, and SHA-512 • KAT RSA PSS signature generation and verification with 2048-bit key and SHA-224, SHA-256, SHA-384, and SHA-512 • KAT RSA with 2048-bit key, public-key encryption • KAT RSA with 2048-bit key, private-key decryption

Algorithm	Test
ECDSA	• PCT ECDSA with P-256 and SHA-256
KAS FFC (Diffie-Hellman)	• Primitive "Z" Computation KAT with 2048-bit key
KAS ECC (EC Diffie-Hellman)	• Primitive "Z" Computation KAT with P-256 curve
DRBG	• KAT CTR_DRBG using AES-256 with and without DF, with and without PR • KAT HASH_DRBG using SHA-256 with and without PR • KAT HMAC_DRBG using SHA-256 with and without PR
HMAC	• KAT HMAC-SHA-1 • KAT HMAC-SHA-224 • KAT HMAC-SHA-256 • KAT HMAC-SHA-384 • KAT HMAC-SHA-512
SHS	• KAT SHA-1 • KAT SHA-256 • KAT SHA-512
Module Integrity	• HMAC-SHA-256

9.2 Conditional Self-Tests

Conditional tests are performed during operational state of the module when the respective crypto functions are used. If any of the conditional tests fails, module transitions to error state.

Table 11 lists the conditional self-tests performed by the functions.

Table 11: Conditional self-tests.

Algorithm	Test
DSA Key generation	PCT using SHA-256, signature generation and verification
ECDSA Key generation	PCT using SHA-256, signature generation and verification
RSA Key generation	PCT using SHA-256, signature generation and verification, and for encryption and decryption
DRBG	Continuous Random Number Generator Test

9.3 On-Demand self-tests

The module provides the Self-Test service to perform self-tests on demand. On demand self-tests can be invoked by powering-off and reloading the module. This service performs the same

cryptographic algorithm tests executed during power-up. During the execution of the on-demand self-tests, cryptographic services are not available and no data output or input is possible.

10 Guidance

This section provides guidance for the Crypto Officer and the User to maintain proper use of the module per FIPS 140-2 requirements.

10.1 Crypto-Officer Guidance

The RPM files containing the FIPS validated module referenced in Section 2.3 must be installed according to this guidance.

For proper operation of the in-module integrity verification, the prelink shall be disabled. This can be done by setting PRELINKING=no in the /etc/sysconfig/prelink configuration file. If the libraries were already prelinked, the prelink shall be undone on all the system files using the 'prelink –u -a' command.

To configure the operating environment to support FIPS perform the following steps:

1. Install the dracut-fips package.

   ```
   # yum install dracut-fips
   ```

2. Recreate the INITRAMFS image.

   ```
   # dracut -f
   ```

After regenerating the initramfs, the Crypto Officer shall append the following string to the kernel command line by changing the setting in the boot loader.

```
fips=1
```

If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<partition of /boot or /boot/efi> must be supplied. The partition can be identified with the command "df /boot" or "df /boot/efi".

For example:

```
$ df /boot
Filesystem  1K-blocks   Used  Available   Use%  Mounted on
/dev/sda1   233191      30454       190296      14%   /boot
```

The partition of /boot is located on /dev/sda1 in the example above. Therefore, the following string needs to be appended to the kernel command line.

```
"boot=/dev/sda1"
```

Reboot to apply these settings. After the reboot, the file /proc/sys/crypto/fips_enabled will contain 1. If the file does not exist or does not contain “1”, the operational environment is not configured to support FIPS and the module will not operate as a FIPS validated module.

After performing the configuration described above, the Crypto Officer shall proceed for module installation with the version of the RPM package listed in Section 2.3. The integrity of the RPM is automatically verified during the installation of the modules and the Crypto Officer shall not install the RPM file if the RPM tool indicates an integrity error

10.2 User Guidance

As specified in Section 2.6, the mode of operation of this module is implicitly selected depending upon which security functions or services and key sizes or curves are being used. To run the module in FIPS mode, the user shall only use the FIPS approved or allowed services listed in Table

4, or the validated or allowed cryptographic algorithms and security functions listed in Table 6 and Table 7.

The function calls FIPS_mode_set(0), ENGINE_register_* and ENGINE_set_default_* are prohibited while running the module.

10.2.1 TLS and Diffie-Hellman

The TLS protocol implementation provides both the server and the client sides. As required by SP800-131A, Diffie-Hellman with keys smaller than 2048 bits must not be used any longer. The TLS protocol lacks the support to negotiate the used Diffie-Hellman key sizes. To ensure full support for all TLS protocol versions, the TLS client implementation of the cryptographic module accepts Diffie-Hellman key sizes smaller than 2048 bits offered by the TLS server.

The TLS server implementation of the cryptographic module allows the application to set the Diffie-Hellman key size. The server side must always set the DH parameters with the API call of SSL_CTX_set_tmp_dh(ctx, dh)

For complying with the requirement to not allow Diffie-Hellman key sizes smaller than 2048 bits, the Crypto Officer must ensure that:

- In case the module is used as TLS server, the Diffie-Hellman parameters (dh argument) of the aforementioned API call must be 2048 bits or larger.
- In case the module is used as TLS client, the TLS server must be configured to only offer Diffie-Hellman keys of 2048 bits or larger.

10.2.2 Random Number Generator

The OpenSSL API call of RAND_cleanup must not be used. This call will clean up the internal DRBG state. This call also replaces the DRBG instance with the non-FIPS Approved SSLeay Deterministic Random Number Generator when using the RAND_* API calls.

10.2.3 AES GCM IV

AES GCM encryption and decryption are used in the context of the TLS protocol version 1.2. The module is compliant with [SP 800-52] and the mechanism for IV generation is compliant with [RFC5288]. The operations of one of the two parties involved in the TLS key establishment scheme are performed entirely within the cryptographic boundary of the module, including the setting of the counter portion of the IV.

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.

In case the module’s power is lost and then restored, the key used for AES GCM encryption or decryption shall be re-distributed.

10.2.4 Triple-DES Data Encryption

Data encryption using the same three-key Triple-DES key shall not exceed 2^{16} Triple-DES (64-bit) blocks, in accordance to [SP800-67] and IG A.13 in [FIPS140-2-IG]. The user of the module is responsible for ensuring the module’s compliance with this requirement.

10.2.5 AES-XTS

The AES algorithm in XTS mode can be only used for the cryptographic protection of data on storage devices, as specified in [SP800-38E]. In addition, the length of a single data unit encrypted with the XTS-AES shall not exceed 2^{20} AES blocks, that is, 16 MiB of data.

In addition, to meet the requirement in [FIPS140-2_IG] A.9, the module implements a check to ensure that the two AES keys used in XTS-AES algorithm are not identical.

10.2.6 Key Usage and Management

In general, a single key shall be used for only one purpose (e.g., encryption, integrity, authentication, key wrapping, random bit generation, or digital signatures) and be disjoint between the modes of operations of the module. Thus, if the module is switched between its FIPS mode and non-FIPS mode or vice versa, the following procedures shall be observed:

- The DRBG engine shall be reseeded.
- CSPs and keys shall not be shared between security functions of the two different modes.

10.3 Handling Self-Test Errors

The module transition to error state when any of self-tests or conditional tests fails. The application must be restarted to recover from these errors. Following are the error messages specific to self-test failure:

FIPS_R_FINGERPRINT_DOES_NOT_MATCH – The integrity verification check failed

FIPS_R_SELFTEST_FAILED – a known answer test failed

FIPS_R_TEST_FAILURE – a known answer test failed (RSA); pairwise consistency test failed (DSA)

FIPS_R_PAIRWISE_TEST_FAILED – a pairwise consistency test failed during EC/DSA or RSA key generation

FIPS_R_DRBG_STUCK – the DRBG generated two same consecutive values

These errors are reported through the regular ERR interface of the module and can be queried by functions such as ERR_get_error(). See the OpenSSL manual page for the function description.

When the module is in error state, output is inhibited and no crypto operations are available. Any calls to the crypto functions in error state will return error message: 'FATAL FIPS SELFTEST FAILURE' on stderr and the application is terminated with the abort() call.

The only way to recover from the error state is to reload the module and restart the application. If failures persist, the module must be reinstalled.

10.4 Enabling Mitigation of Other Attacks

To enable the mechanisms implemented in the module to mitigate other attacks, please refer to Section 11.

11 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 of 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 the Weak Triple-DES Key Detection, the module does not implement the weak key detection by default. The caller may call DES_check_key_parity() and/or DES_is_weak_key() functions or explicitly set the DES_check_key to use DES_set_key_checked() function, which checks the weak Triple-DES key and the correctness of the parity bits when the Triple-DES key is used in Triple-DES operations. The checking of the weak Triple-DES key is implemented in the API function DES_is_weak_key() and the checking of the parity bits is implemented in the API function DES_check_key_parity(). If the Triple-DES key does not pass the check, the module will return -1 to indicate the parity check error and -2 if the Triple-DES key matches to any value listed in the weak_keys variable.

Weak Triple-DES keys are detected per the code excerpt below.

```
/* Weak and semi week keys as taken from
* %A D.W. Davies
* %A W.L. Price
* %T Security for Computer Networks
* %I John Wiley & Sons
* %D 1984
* Many thanks to smb@ulysses.att.com (Steven Bellovin) for the reference
* (and actual cblock values).
*/
#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}};
```

12 Acronyms, Terms and Abbreviations

Term	Definition
AES	Advanced Encryption Standard
AESNI	Advanced Encryption Standard New Instructions
AVX	Advanced Vector Extensions
AVX2	Advanced Vector Extensions 2
CAVP	Cryptographic Algorithm Validation Program
CMVP	Cryptographic Module Validation Program
CSE	Communications Security Establishment
CSP	Critical Security Parameter
DH	Diffie-Hellman
DHE	Diffie-Hellman Ephemeral
DRBG	Deterministic Random Bit Generator
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EDC	Error Detection Code
HMAC	(Keyed) Hash Message Authentication Code
IKE	Internet Key Exchange
KAT	Known Answer Test
KDF	Key Derivation Function
NDRNG	Non-Deterministic Random Number generator
NIST	National Institute of Standards and Technology
PAA	Processor Algorithm Acceleration
POST	Power On Self Test
PR	Prediction Resistance
PSS	Probabilistic Signature Scheme
PUB	Publication
SHA	Secure Hash Algorithm
SSSE3	Supplemental Streaming SIMD Extensions 3
VPAES	AES with Vector Permutations
TLS	Transport Layer Security

13 References

The FIPS 140-2 standard, and information on the CMVP, can be found at http://csrc.nist.gov/groups/STM/cmvp/index.html. More information describing the module can be found on the vendor web site at aws.amazon.com .

This Security Policy contains non-proprietary information. All other documentation submitted for FIPS 140-2 conformance testing and validation is proprietary and is releasable only under appropriate non-disclosure agreements.

Document	Author	Title
DTR for FIPS 140-2	NIST	Derived Test Requirements (DTR) for FIPS 140-2, Security Requirements for Cryptographic Modules
FIPS 140-2	NIST	FIPS 140-2: Security Requirements for Cryptographic Modules
FIPS 140-2 Annex A	NIST	FIPS 140-2 Annex A: Approved Security Functions
FIPS 140-2 Annex B	NIST	FIPS 140-2 Annex B: Approved Protection Profiles
FIPS 140-2 Annex C	NIST	FIPS 140-2 Annex C: Approved Random Number Generators
FIPS 140-2 Annex D	NIST	FIPS 140-2 Annex D: Approved Key Establishment Techniques
FIPS IG	NIST	Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program
FIPS PUB 180-4	NIST	Secure Hash Standard (SHS)
FIPS PUB 186-4	NIST	Digital Signature Standard (DSS)
FIPS PUB 197	NIST	Advanced Encryption Standard
FIPS PUB 198-1	NIST	The Keyed Hash Message Authentication Code (HMAC)
NIST SP 800-131A	NIST	Recommendation for the Transitioning of Cryptographic Algorithms and Key Sizes
NIST SP 800-67	NIST	Recommendation for the Triple Data Encryption Algorithm TDEA Block Cipher
PKCS#1	RSA Laboratories	PKCS#1 v2.1: RSA Cryptographic Standard
RFC 5246	https://tools.ietf.org/html/rfc5246	The Transport Layer Security (TLS) Protocll Version 1.2
RFC 5288	https://tools.ietf.org/html/rfc5288	AES Galois Counter Mode (GCM) Cipher Suites for TLS