Trellix FIPS 140-2 Security Policy

Trellix FIPS Provider

Version: 3.0.0a

Date: 30 March 2023

Modification History

Version	Description	Release Date
1.0	Initial Draft	30 March, 2023

Table of Contents

List of Tables

List of Figures

FIPS 140-2 Overview

Federal Information Processing Standards Publication 140-2 — Security Requirements for Cryptographic Modules specifies requirements for cryptographic modules to be deployed in a Sensitive but Unclassified environment. The National Institute of Standards and Technology (NIST) and Canadian Centre for Cyber Security (CCCS) Cryptographic Module Validation Program (CMVP) run the FIPS 140 program. NVLAP accredits independent testing labs to perform FIPS 140-2 testing; the CMVP validates modules meeting FIPS 140-2 validation. Validated is the term given to a module that is documented and tested against the FIPS 140-2 criteria.

More information is available on the CMVP website at:
http://csrc.nist.gov/groups/STM/cmvp/index.html

About this Document

This non-proprietary Cryptographic Module Security Policy for the Trellix FIPS Provider module from Trellix provides an overview and a high-level description of how it meets the overall Level 1 security requirements of FIPS 140-2.

1. Introduction

1.1 Scope

This document describes the non-proprietary cryptographic module security policy for the Trellix FIPS Provider module, hereafter referred to as “the Module.” It contains specification of the security rules, under which the cryptographic module operates, including the security rules derived from the requirements of the FIPS 140-2 standard.

1.2 Module Overview

The Trellix FIPS Provider is a software library providing a C-language application program interface (API) for use by applications using OpenSSL that require cryptographic functionality. The Module is classified under FIPS 140-2 as a software module, with a multi-chip standalone module embodiment. The physical cryptographic boundary is the general-purpose computer on which the module is installed.

The logical cryptographic boundary of the Module is the FIPS Provider, a dynamically loadable library. The Module performs no communication other than with the calling application via APIs that invoke the Module.

The module implements both an Approved and non-Approved mode of operation. Use of the Approved algorithms listed in table 7 and allowed algorithms listed in table 8 will place the module in the Approved mode of operation. Use of the non-Approved algorithms listed in table 9 will place the module in the non-Approved mode of operation.

1.3 Module Boundary

The following block diagram details the Module's physical and logical boundaries.

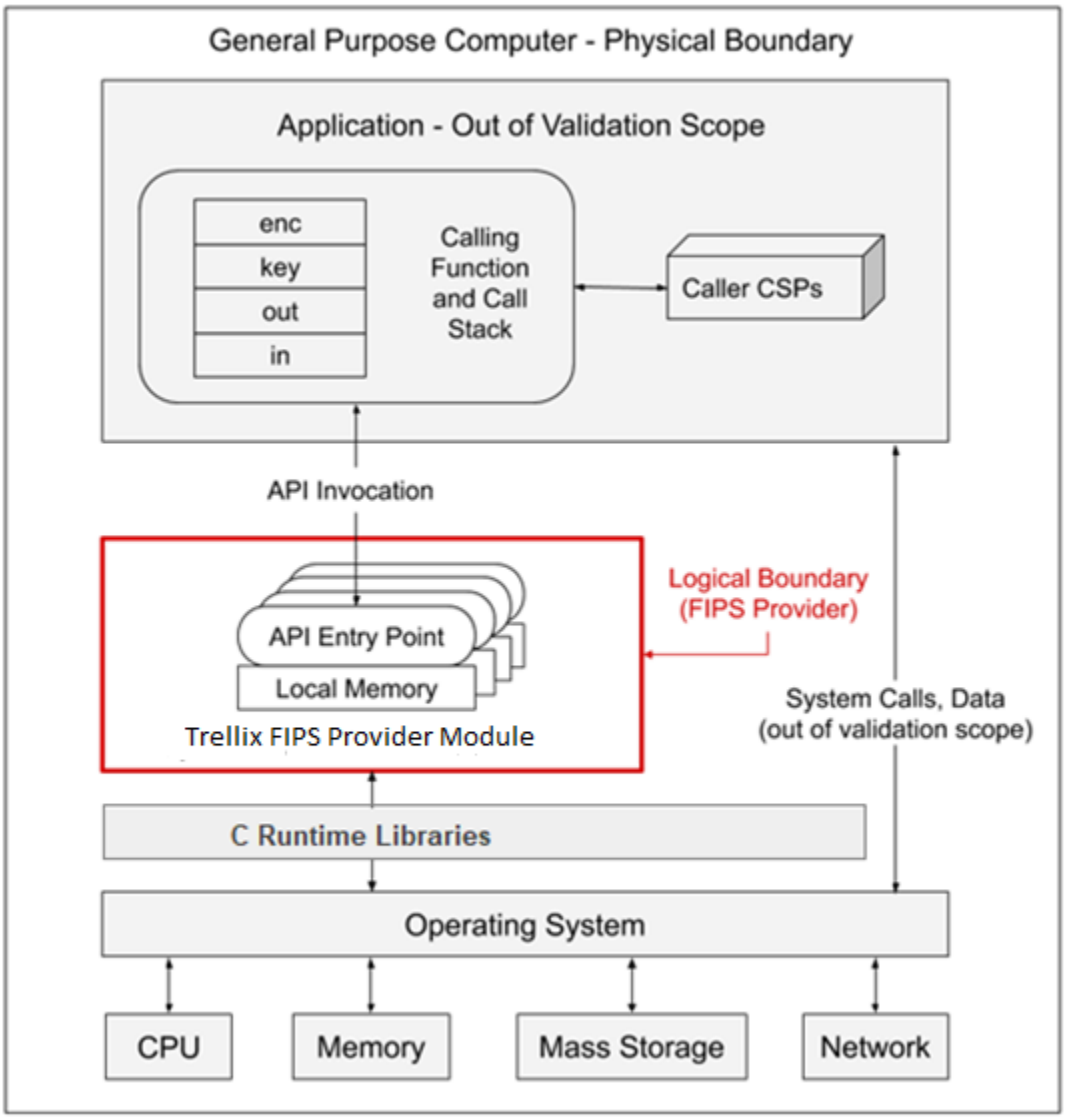

Figure 1 – Module Block Diagram

2. Security Level

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

FIPS 140-2 Security Requirement Areas	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	3
Mitigation of Other Attacks	1
Overall Level	1

Table 1 – Security Levels for each FIPS 140-2 Area

The Module meets the overall security level requirements of Level 1. The Module's software version for this validation is 3.0.0a.

3. Tested Configurations

The Module has been tested on the platforms listed below in Table 2.

#	Operating System/Hypervisor	Hardware Platform	Processor	Optimizations (Target)
1	Photon OS 4.0 on ESXi 7.0	Dell PowerEdge R740	Intel Xeon Gold 6230R (x64)	PAA (AES-NI)
2	Windows Server 2019 on ESXi 7.0	Dell PowerEdge R740	Intel Xeon Gold 6230R (x64)	PAA (AES-NI)
3	Oracle Solaris 11.4	Oracle SPARC T8-1	Oracle SPARC M8-1 (x64)	PAA (SPARC)
4	Ubuntu Linux 20.04.1 Server	Dell Inspiron 7573	Intel i7 (x64)	PAA (AES-NI)
5	Windows 10	Dell Inspiron 7591	Intel i7 (x64)	PAA (AES-NI)
6	FreeBSD stable/13	NetApp HCI H700E	Intel Xeon E5-2695v4 (Broadwell) (x64)	PAA (AES-NI)
7	Debian 10	Fujitsu CX400 Blade	Intel Xeon Silver 4116 (Cascade Lake) (x64)	PAA (AES-NI)
8	Custom Ubuntu Linux 18.04	Akamai X8 system	Intel Xeon D1541 (x64)	PAA (AES-NI)
9	macOS 11.5.2	Apple M1 Mac Mini	M1	None
10	macOS 11.5.2	Apple M1 Mac Mini	M1	PAA (AES-NI)
11	macOS 11.5.2	Apple i5 Mac Mini	Intel i7	None
12	macOS 11.5.2	Apple i5 Mac Mini	Intel i7	PAA (AES-NI)

Table 2 – Tested Configurations

See Appendix A for additional information on installation. See Appendix B for a list of the specific compilers used to generate the Module for the respective operational environments.

4. Ports and Interfaces

The physical ports of the Module are the same as the computer system on which it is executing. The logical interface is a C-language application program interface (API), the mapping of which is described in the following table:

Logical Interface Type	Description
Data Input	API entry point data input stack parameters
Data Output	API entry point data output stack parameters
Control Input	API entry point and corresponding stack parameters
Status Output	API entry point return values and status stack parameters

Table 3 – Physical Port and Logical Interface Mapping

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 logical data output interface is inhibited. In error scenarios, the module returns only an error value (no data output is returned).

5. Roles, Services and Authentication

5.1 Roles

The Module implements both a User Role (User) as well as the Crypto Officer (CO) role. The Module does not support authentication and does not allow concurrent operators. The User and Crypto Officer roles are implicitly assumed by the application accessing services implemented by the Module.

5.2 Services

All the services provided by the module can be accessed by both the User and the Crypto Officer roles. The User Role (User) can load the Module and call any of the API functions. The Crypto Officer Role (CO) is responsible for installation of the Module on the host computer system and calling of any API functions. The module provides the following Approved services which utilize algorithms listed in Tables 6 and 7:

Service	Roles (User/CO)	Description
Initialize	X	Module initialization. Does not access CSPs.
Self-Test	X	Perform POST self-tests (SELF_TEST_post( )) on demand. Does not access CSPs.
Show Status	X	• The Module's status can be verified by querying the "status" parameter. Does not access CSPs.
CSP/Key Zeroization	X	All services automatically overwrite CSPs stored in allocated memory. Stack cleanup is the responsibility of the calling application.
Random Number Generation	X	Used for random number and symmetric key generation. • Seed or reseed a DRBG instance • Determine security strength of a DRBG instance • Obtain random data Uses and updates Hash_DRBG CSPs, HMAC_DRBG CSPs, CTR_DRBG CSPs
Asymmetric Key Generation	X	Used to generate DSA, ECDSA, RSA, DH, ECDH, X25519 and X448 keys: • RSA SGK, RSA SVK; DSA SGK, DSA SVK; ECDSA SGK, ECDSA SVK; DH

Service	Roles (User/CO)	Description
		Private, DH Public, ECDH Private, ECDH Public; X25519 Private, X25519 Public, X448 Private and X448 Public keys There is one supported entropy strength for each mechanism and algorithm type, the maximum specified in SP 800-90Ar1
Key Derivation	X	Used to derive keys using KBKDF, PBKDF2, HKDF, SP 800-56Cr2 One-Step KDF (KDA), SP 800-135 TLS 1.2, SSHv2, ANSI X9.6-2001, ANSI X9.42-2001 KDFs and TLS 1.3 KDF.
Symmetric Encrypt/Decrypt	X	Used to encrypt or decrypt data. Executes using AES EDK, TDES EDK (passed in by the calling application).
Symmetric Digest	X	Used to generate or verify data integrity with CMAC. Executes using AES CMAC Key (passed in by the calling application).
Message Digest	X	Used to generate a SHA-1, SHA-2, or SHA-3 message digest. Does not access CSPs
Keyed Hash	X	Used to generate or verify data integrity with HMAC or KMAC. Executes using HMAC or KMAC Key (passed in by the calling application)
Key Transport	X	Used to encrypt or decrypt a key value on behalf of the calling application (does not establish keys into the module). Executes using RSA KDK, RSA KEK (passed in by the calling application).
Key Wrapping	X	Used to encrypt a key value on behalf of the calling application Executes using AES Key Wrapping Key (passed in by the calling application).
Key Agreement	X	Used to perform key agreement primitives on behalf of the calling application (does not establish keys into the module). Executes using DH Private, DH Public, EC DH Private, EC DH Public, X25519 Private, X25519 Public, X448 Private and X448 Public, RSA SGK, RSA SVK (passed in by the calling application).
Digital Signature	X	Used to generate or verify RSA, DSA, ECDSA digital signatures.

Service	Roles (User/CO)	Description
		Executes using RSA SGK, RSA SVK; DSA SGK, DSA SVK; ECDSA SGK, ECDSA SVK (passed in by the calling application).
Utility	X	Miscellaneous helper functions. Does not access CSPs.

Table 4 – Approved Services and Role Allocation

The module provides the following non-Approved services which utilize algorithms listed in Table 5:

Service	Roles (User/CO)	Description
Digital Signature	X	Used to generate or verify Ed25519 or Ed448 digital signatures.

Table 5 - Non-Approved Services and Role Allocation

6. Physical Security

The physical boundary of the Module is the general-purpose computer on which the module is installed. The Module meets all physical security requirements of a Security Level 1 software module under FIPS 140-2 requirements.

7. Operational Environment

The tested operating systems, listed in Table 2, segregate applications into separate spaces. Each application space is logically separated from all other applications by the operating system software and hardware. The Module functions entirely within the operating system provided space for the calling application, and implicitly satisfies the FIPS 140-2 requirement for a single-user mode of operation.

8. Cryptographic Algorithms and Key Management

8.1 Cryptographic Algorithms

The module implements the following ***Approved*** algorithms:

CAVP Cert #	Algorithm	Standard	Sizes/Curves	Mode/Method	Use
A1938	AES [FIPS 197]	SP 800-38A	128, 192, 256 bits	ECB, CBC,CBC-CS1, CS2, CS3, OFB, CFB 1, CFB 8, CFB 128, CTR	Encryption, Decryption and CMAC Generate/Verify
		SP 800-38B		CMAC	
		SP 800-38C		CCM	
		SP 800-38D		GCM, GMAC	
		SP 800-38F		KW, KWP (cipher, inverse)	
		SP 800-38E	128, 256 bits	XTS	
A1938	Triple-DES	SP 800-67r2	3-Key TDES	ECB, CBC	Encryption, Decryption
A1938	DSA	FIPS 186-4	L = 2048, N = 224 L = 2048, N = 256 L = 3072, N = 256	Key Pair Gen	Digital Signature and Asymmetric Key Generation
			L = 2048, N = 224 L = 2048, N = 256 L = 3072, N = 256	PQG Gen Sig Gen	

CAVP Cert #	Algorithm	Standard	Sizes/Curves		Mode/Method	Use
			with all SHA-2 sizes			
			L = 1024, N = 160 L = 2048, N = 224 L = 2048, N = 256 L = 3072, N = 256 with all SHA sizes		PQG Ver Sig Ver	
A1938	ECDSA	FIPS 186-4	P-224, 256, 384, 521 K-233, 283, 409, 571 B-233, 283, 409, 571 Testing Candidates		Key Gen	Digital Signature and Asymmetric Key Generation
			P-192, P-224, 256, 384, 521 K-163, 233, 283, 409, 571 B-163, 233, 283, 409, 571		PKV	
			P-224, 256, 384, 521	SHA2-224, 256, 384, 512, 512/224, 512/256	SigGen	
			K-233, 283, 409, 571			
			B-233, 283, 409, 571			
			P-192, 224, 256, 384, 521	SHA-1, SHA2-224, 256, 384, 512, 512/224, 512/256	SigVer	
			K-163, 233, 283, 409, 571			

CAVP Cert #	Algorithm	Standard	Sizes/Curves	Mode/Method	Use
			B-163, 233, 283, 409, 571		
A1938	CVL	FIPS 186-4	ECDSA SigGen Component	SHA2-224, 256, 384, 512, 512/224, 512/256 with P-224, 256, 384, 521, K-233, 283, 409, 571, B-233, 283, 409, 571	Digital Signature Generation
A1938	RSA	FIPS 186-4	2048, 3072, 4096	Gen Key 9.31	Digital Signature and Asymmetric Key Generation
			1024, 2048, 3072, 4096 (SHA-1, 224, 256, 384, 512, 512/224, 512/256)	Sig Ver 9.31	
			2048, 3072, 4096 (SHA-1, 256, 384, 512)	Sig Gen 9.31	
			2048, 3072, 4096 (SHA-224, 256, 384, 512)	Sig Gen PKCS 1.5	
			1024, 2048, 3072, 4096 (SHA-1, SHA-224, 256, 384, 512, 512/224, 512/256)	Sig Ver PKCS 1.5	
			2048, 3072 (SHA-224, 256, 384, 512, 512/224, 512/256)	Sig Gen PSS	
			4096 (SHA-224, 256, 384, 512)		

CAVP Cert #	Algorithm	Standard	Sizes/Curves	Mode/Method	Use
			1024, 2048, 3072, 4096 (SHA-1, SHA-224, 256, 384, 512, 512/224, 512/256)	Sig Ver PSS	
A1938	CVL	FIPS 186-4	RSASP1 (mod 2048)		Signature Generation Primitive
A1938	KAS-FFC-SSC	SP 800-56Ar3	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192 safe prime groups per SP 800-56Ar3	dhEphem Domain Parameter Generation Domain Parameter Validation Key Pair Generation Full Public Key Validation Partial Public Key Validation	Key Agreement
	KAS-ECC-SSC		All P, B, K curves (per Appendix D of SP 800-56Ar3) with all SHA sizes	Ephemeral Unified Domain Parameter Generation Domain Parameter Validation Key Pair Generation Full Public Key Validation	
A1938	KAS-RSA-SSC	SP 800-56Br2	2048, 3072, 4096, 6144, 8192 with SHA2-224, 256, 384, 512, 512/224, 512/256, SHA-3-224, 256, 384, 512	KAS1, KAS2 Key Generation Methods: rsakpg1-crt, rsakpg2-crt, rsakpg1-basic,	Key Agreement

CAVP Cert #	Algorithm	Standard	Sizes/Curves	Mode/Method	Use
				rsakpg2-basic, rsakpg1-prime-factor, rsakpg2-prime-factor	
A1938	CVL	SP 800-56Ar3	KAS ECC CDH with P-224, 256, 384, 521 , K-233, 283, 409, 571, B-233, 283, 409, 571		Section 5.7.1.2 ECC CDH Primitive used in Shared Secret Computation
N/A	KTS	SP 800-38F	KTS (AES Cert. # A1938; key establishment methodology provides between 128 and 256 bits of encryption strength)	AES KW, KWP AES CCM AES GCM	Key Transport
			KTS (AES Cert. #A1938 and HMAC Cert. #A1938; key establishment methodology provides between 128 and 256 bits of encryption strength)	AES (any mode) and HMAC	
			KTS (AES Cert. #A1938 and AES Cert. #A1938; key establishment methodology provides between 128 and 256 bits of encryption strength)	AES (any mode) and CMAC, GMAC	
			KTS (Triple-DES Cert. #A1938 and HMAC Cert. #A1938; key establishment methodology provides 112 bits of encryption strength)	Triple-DES (ECB, CBC) and HMAC	
A1938	KTS-RSA	SP 800-56Br2	2048, 3072, 4096 KTS-RSA (#A1938; key establishment methodology provides between 112 and 128 bits of encryption strength)	RSA-OAEP, RSADP, RSAEP	Key Transport

CAVP Cert #	Algorithm	Standard	Sizes/Curves	Mode/Method	Use
A1938	KDA	SP 800-56Cr2	One-Step KDF (Section 4), Two Step KDF (Section 5)		Key Derivation Function
A1938	HKDF	SP 800-56Cr2	HMAC-based Extract-and-Expand Key Derivation Function		Key Derivation Function
A1938	PBKDF2	SP 800-132	HMAC-SHA-1, SHA2-224, 256, 384, 512, 512/224, 512/256 (C = 1 – 10,000, sLen = 16 - 512 bytes)	Option 1a	Password Based Key Derivation
A1938	KBKDF	SP 800-108	CMAC AES128, CMAC AES192, CMAC AES256, HMAC-SHA-1, SHA2-224, 256, 384, 512	Counter, Feedback	Key-Based Key Derivation Function
A1938	CVL	SP 800-135	TLS 1.2 KDF (SHA2-256, 384, 512), SSHv2 KDF (SHA-1, SHA2-224, 256, 384, 512), ANSI X9.63-2001 KDF (SHA2-224, 256, 384, 512), ANSI X9.42-2001 KDF (SHA-1, SHA2-224, 256, 384, 512, 512/224, 512/256, SHA3-224, 256, 384, 512)		Key Derivation Function
A1938	CVL	RFC 8446 (Section 7.1)	TLS 1.3 KDF (SHA2-256, 384)		Key Derivation Function
A1938	SHA-3, SHAKE	FIPS 202	SHA3-224, 256, 384, 512 SHAKE-128, 256	SHA-3	Message Digests
A1938	SHS	FIPS 180-4	SHA-1 SHA2- 224, 256, 384, 512, 512/224, 512/256	SHA-1 SHA-2	Message Digests
A1938	HMAC	FIPS 198-1	SHA-1 SHA2-224, 256, 384, 512, 512/224, 512/256	SHA-1 SHA-2	Keyed Hash

CAVP Cert #	Algorithm	Standard	Sizes/Curves	Mode/Method	Use
			SHA3-224, 256, 384, 512	SHA-3	
A1938	KMAC	SP 800-185	KMAC-128 KMAC-256		Keyed Hash
Vendor Affirmed	CKG	SP 800-133r2	Cryptographic Key Generation		Key Generation Section 4 (Using the Output of a Random Bit Generator), Section 6.1 (Direct Generation of Symmetric Keys) and Section 6.2 (Derivation of Symmetric Keys)
A1938	DRBG	SP 800-90A	SHA-1 SHA2-224, 256, 384, 512, 512/224, 512/256 SHA3-224, 256, 384, 512	Hash DRBG	Random Number Generation; Symmetric Key Generation
			SHA-1 SHA2-224, 256, 384, 512, 512/224, 512/256 SHA3-224, 256, 384, 512	HMAC DRBG	
			AES-128, AES-192, AES-256	CTR DRBG	

Table 6 – FIPS Approved Algorithms

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 per requirements in IG 9.10. The power-on self-tests run during the call to the Module's OSSL_provider_init() entry point.

The Module is a cryptographic library, which can be used only in conjunction with additional software.

The module implements the following ***Allowed*** algorithms:

Algorithm	Use
X25519 (curve25519 with 128 bits of security strength)	Key Agreement
X448 (curve448 with 224 bits of security strength)	Key Agreement

Table 7 – Allowed Algorithms

The module implements the following non-Approved algorithms:

Algorithm	Use
Ed448	Digital Signature Generation
Ed25519	Digital Signature Generation

Table 8 – Non-Approved Algorithms

These algorithms shall not be used when operating in the FIPS Approved mode of operation. Use of the non-Approved algorithms listed in the table above will place the module in the non-Approved mode of operation.

8.2 Critical Security Parameters (CSP's) and Public Keys

The Module supports the following CSPs listed below in Table 7. The CSP access policy is denoted in Table 4 above.

Keys or CSP Name	Description
RSA SGK	RSA (2048 to 16384 bits) signature generation key
RSA KDK	RSA (2048 to 16384 bits) key decryption (private key transport) key
DSA SGK	DSA (2048/3072) signature generation key
ECDSA SGK	ECDSA (All NIST defined B, K, and P curves) signature generation key
DH Private	DH (256-512 bits) private key agreement key
EC DH Private	EC DH (All NIST defined B, K, and P curves) private key agreement key
X25519 Private	X25519 private key agreement key

Keys or CSP Name	Description
X448 Private	X448 private key agreement key
AES EDK	AES (128/192/256) encrypt / decrypt key
AES CMAC	AES (128/192/256) CMAC generate / verify key
AES GCM	AES (128/192/256) encrypt / decrypt key
AES XTS	AES (128/256) XTS encrypt / decrypt key
AES Key Wrapping	AES (128/192/256) key wrapping key
TDES EDK	TDES (3-Key) encrypt / decrypt key
HMAC Key	Keyed hash key (160/224/256/384/512)
KMAC Key	Keyed hash key (128-1024 bits)
Hash_DRBG CSPs	V (440/888 bits) and C (440/888 bits), entropy input (length dependent on security strength)
HMAC_DRBG CSPs	V (160/224/256/384/512 bits) and Key (160/224/256/384/512 bits), entropy input (length dependent on security strength)
CTR_DRBG CSPs	V (128 bits) and Key (AES 128/192/256), entropy input (length dependent on security strength)
KDF Secret	The secret value used for constructing the key for the PRF used for key derivation (SP 800-108 KBKDF, SP 800-132 PBKDF, HKDF, KDA, SP 800-135 KDFs, TLS 1.3 KDF).

Table 6 – Critical Security Parameters

The Module does not output intermediate key generation values. The Module supports the following Public Keys listed below in Table 8.

Key/Parameter Name	Description
RSA SVK	RSA (1024 to 16384 bits) signature verification public key
RSA KEK	RSA (2048 to 16384 bits) key encryption (public key transport) key
DSA SVK	DSA (1024/2048/3072) signature verification key
ECDSA SVK	ECDSA (All NIST defined B, K and P curves) signature verification key
EC DH Public	EC DH (All NIST defined B, K, and P curves) public key agreement key
DH Public	DH (2048/3072/4096/6144/8192) public key agreement key
X25519 Public	X25519 public key agreement key
X448 Public	X448 public key agreement key

Table 7 – Public Keys

For all CSPs and Public Keys:

- **Storage:** RAM, associated to entities by memory location. The Module stores DRBG state values for the lifetime of the DRBG instance. The module uses CSPs passed in by the calling application

on the stack. The Module does not store any CSP persistently (beyond the lifetime of an API call), except for DRBG state values used for the Module's default key generation service.

- **Generation:** The Module implements SP 800-90 compliant DRBG services for creation of symmetric keys, and for generation of DSA, elliptic curve, and RSA keys as shown in Table 4. The calling application is responsible for storage of generated keys returned by the module.
- **Entry:** All CSPs enter the Module's logical boundary in plaintext as API parameters, associated by memory location. However, none cross the physical boundary.
- **Output:** The Module does not output CSPs, other than as explicit results of key generation services or keys passed into the module by the calling application. However, none cross the physical boundary.
- **Destruction:** Zeroization of sensitive data is performed automatically by API function calls for temporarily stored CSPs. The calling application is responsible for parameters passed in and out of the module.

8.3 Key Generation and Entropy

Private and secret keys as well as seeds and entropy input are provided to the Module by the calling application and are destroyed when released by the appropriate API function calls. Keys 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 application space from unauthorized access. Only the calling application that creates or imports keys can use or export such keys. All API functions (Module Services) are executed by the calling application invoking an API. Each API either succeeds or fails and is logically non-interruptible from the point of view of the calling application.

The module supports generation of ECDSA, RSA, DSA, EC Diffie-Hellman and Diffie-Hellman key pairs per Section 5 in NIST SP 800-133. A NIST SP 800-90Ar1 random bit generator is used for generating the seed used in asymmetric key generation.

Applications shall use entropy sources that meet the security strength required for the random number generation mechanism as shown in [SP 800-90Ar1] Table 2 (Hash_DRBG, HMAC_DRBG, CTR_DRBG). A minimum of 112-bits of entropy must be supplied. This entropy is supplied by means of callback functions. Those functions must return an error if the minimum entropy strength cannot be met.

9. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The tested platforms listed in Table 2, on which the Module operates, are compliant with 47 code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators.

10. Self-tests

FIPS 140-2 requires self-tests to test the integrity of the operational environment at start-up. Some functions also require additional conditional tests during operation of the module.

The Module performs the self-tests listed below upon invocation of Initialize or on-demand Self-test.

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, and all data output is inhibited.

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

Algorithm	Type	Test Attributes
Software Integrity	KAT	HMAC-SHA-256
SHS	KAT	SHA-1, SHA2-512, SHA-3-256
AES	KAT	Decrypt, ECB mode, 128-bit key
AES GCM	KAT	Encrypt, Decrypt, 256-bit key
Triple-DES	KAT	Encrypt, Decrypt, 3-Key, CBC mode
RSA	KAT	Sign, Verify using 2048-bit key, SHA-256, PKCS#1
DSA	PCT	Sign, Verify using 2048-bit key, SHA-256
DRBG	KAT	CTR_DRBG: AES, 128-bit with derivation function HASH_DRBG: SHA-256 HMAC_DRBG: SHA-1 Generate, Reseed, Instantiate functions (per Section 11.3 of SP 800-90Ar1)
ECDSA	PCT	Sign, Verify using P-224, K-233
KBKDF	KAT	Counter Mode (HMAC-SHA-256)
PBKDF2	KAT	Derivation of the Master Key (MK) (per Section 5.3 of SP 800-132).
SP 800-135 KDFs	KAT	TLS 1.2, SSHv2, ANSI X9.63-2001 and ANSI X9.42-2001 KDFs.
TLS v1.3 KDF	KAT	TLS v1.3 KDF (per Section 7.1 of RFC 8446)
KAS FFC-SSC	KAT	dhEphem Shared Secret (Z) Computation (per Section 6 of SP 800-56Ar3)
KAS ECC-SSC	KAT	Ephemeral Unified Shared Secret (Z) Computation (per Section 6 of SP 800-56Ar3), P-256
KAS-RSA-SSC	KAT	RSA Primitive Computation (per Scenario 2 of IG D.8 and Section 8.2.2 in SP 800-56Br2)
KDA	KAT	One-step KDF (per Section 4 of SP 800-56Cr2) and Two-Step KDF (per Section 5 of SP 800-56Cr2
KTS-RSA	KAT	Encrypt and Decrypt for Basic, Decrypt for CRT (per IG D.9 and SP 800-56Br2)

Table 8 – Power On Self-Tests

The Module is installed using one of the set of instructions in Appendix A, as appropriate for the operational environment.

The SELF_TEST_post() function performs all power-up self-tests listed above with no operator intervention required when the module loads, returning a "1" if all power-up self-tests succeed, and a "0" otherwise. The power-up self-tests may also be performed on-demand by calling this function and interpretation of the return code is the responsibility of the calling application.

If any component of the power-up self-test fails, an internal flag is set to prevent subsequent invocation of any cryptographic function calls. The module will only enter the FIPS Approved mode if the module is reloaded and the call to SELF_TEST_post() succeeds.

10.2 Conditional Self-Tests

Conditional self-tests are run under specific conditions, such as during key generation. The Module implements the following conditional tests:

Algorithm	Test
Entropy Source	FIPS 140-2 continuous test for stuck fault
DSA	Pairwise consistency test on generation of a key pair
ECDSA	Pairwise consistency test on generation of a key pair
RSA	Pairwise consistency test on generation of a key pair

Table 9 – Conditional Tests

In the event of a DRBG self-test failure, the calling application must uninstantiate and reinstantiate the DRBG per the requirements of [SP 800-90]; this is not something the Module can do itself.

10.3 Assurances

The Module obtains the following assurances per SP 800-56Ar3 and SP 800-56Br2:

Standard	Assurances
SP 800-56Ar3	Per Section 5.6.2 of SP 800-56Ar3, required per IG D.8
SP 800-56Br2	Per Section 6.4 of SP 800-56Br2, required per IG D.8

Table 10 – Assurances

10.4 Critical Function Tests

The module does not implement any specific critical function tests.

11. Mitigation of Other Attacks

The Module implements two mitigations against timing-based side-channel attacks, namely Constant-time Implementations and Blinding.

Constant-time Implementations protect cryptographic implementations in the Module against timing analysis since such attacks exploit differences in execution time depending on the cryptographic operation, and constant-time implementations ensure that the variations in execution time cannot be traced back to the key, CSP or secret data.

Numeric Blinding protects the RSA, DSA and ECDSA algorithms from timing attacks. These algorithms are vulnerable to such attacks since attackers can measure the time of signature operations or RSA decryption. To mitigate this the Module generates a random blinding factor which is provided as an input to the decryption/signature operation and is discarded once the operation has completed and resulted in an output. This makes it difficult for attackers to attempt timing attacks on such operations without the knowledge of the blinding factor and therefore the execution time cannot be correlated to the RSA/DSA/ECDSA key.

12. Crypto Officer and User Guidance

Please see Appendix A for installation and usage guidance for module operators.

12.1 AES-GCM Usage

The Module does not implement the TLS protocol itself, however, it provides the cryptographic functions required for implementing the protocol. AES GCM encryption is used in the context of the TLS protocol versions 1.2 and 1.3 (per Scenario 1 and Scenario 5 in FIPS 140-2 A.5 respectively). For TLS v1.2, the mechanism for IV generation is compliant with RFC 5288. The counter portion of the IV is strictly increasing. When the IV exhausts the maximum number of possible values for a given session key, this results in a failure in encryption and a handshake to establish a new encryption key will be required. It is the responsibility of the user of the module i.e., the first party, client or server, to encounter this condition, to trigger this handshake in accordance with RFC 5246. For TLS v1.3, the mechanism for IV generation is compliant with RFC 8446.

The Module also supports internal IV generation using the module's Approved DRBG. The IV is at least 96-bits in length per NIST SP 800-38D, Section 8.2.2. Per FIPS 140-2 IG A.5 Scenario 2 and NIST SP 800-38D, the approved DRBG generates outputs such that the (key, IV) pair collision probability is less than 2^{-32}.

In the event that the Module power is lost and restored the user must ensure that the AES-GCM encryption/decryption keys are re-distributed.

The Module also supports importing of GCM IVs when an IV is not generated within the Module. In the FIPS approved mode, an IV must not be imported for encryption from outside the cryptographic boundary of the Module as this will result in a non-conformance.

12.2 Triple-DES Usage

The calling application shall ensure that a given Triple-DES key is used to encrypt no more than 2^{16} 64-bit blocks of data.

12.3 Miscellaneous

- The module performs run-time checks related to enforcement of security parameters such as the minimum-security strength of keys, valid key sizes, and usage of approved curves. These checks shall not be disabled (by using OPENSSL_NO_FIPS_SECURITYCHECKS or any other method).
- Validation of domain parameters prior to generating keys using functions provided by the module is the responsibility of the Cryptographic Officer and not enforced by the module itself.

Appendix A: Installation and Usage Guidance

The Module is installed as part of the OpenSSL 3.0.0 library. The source distribution package is located at https://www.openssl.org/source/openssl-3.0.0.tar.gz.

The FIPS Provider can be installed on the Tested Configurations listed in Table 2 by performing the following steps:

1. Build and install OpenSSL 3.0.0 to the default location:

The FIPS provider i.e., the Module does not get built and installed automatically. To install the module automatically during the normal OpenSSL 3.0.0 installation process it must be enabled by configuring OpenSSL using the 'enable-fips' option.

Unix/Linux/macOS:

$ ./Configure enable-fips

$ make

$ make install

Windows:

$ perl Configure enable-fips

$ nmake

$ nmake install

The 'install_fips' make target can also be invoked explicitly to install the FIPS provider independently, without installing the rest of OpenSSL:

$ make install_fips

Note: The instructions for building and installing OpenSSL 3.0.0 on other platforms can be found in the platform-specific guidance provided in INSTALL.md and README-FIPS.md in the OpenSSL 3.0.0 distribution package. Please see Appendix B for further information on porting the Module to platforms apart from the Tested Configurations in Table 2.

2. Verify the version:

$ openssl version -v

The Installation of the FIPS provider that occurs as a result of Step 1 above essentially consists of two steps. In the first step, the shared library is copied to its installed location. In the second step, the 'openssl fipsinstall' command is executed, which completes the installation by doing the following two things:

- Runs the Module's self-tests.
- Generates the Module config file output containing information about the Module (such as the self-test status, and the Module checksum).

To install the FIPS configuration file to a non-default location, this can be achieved by running the 'fipsinstall' command line application manually:

$ openssl fipsinstall

Please see the manual page for options supported for the 'openssl fipsinstall' command.

Note: The Module shall have the self-tests run, and the Module config file output generated on each platform where it is intended to be used. The Module config file output data shall not be copied from one machine to another.

Note: Two integrity checks are performed, the software integrity check (per Section 10.1 of this document) during the installation and an additional integrity check post installation of the Module. The software integrity check is performed using HMAC-SHA-256 on the Module file to validate that the Module has not been modified. The integrity value is compared to a value written to the config file during installation.

The other integrity check is performed once the Module has been installed using HMAC-SHA-256 on a fixed string to validate that the installation process has already been performed and that the self-tests have been executed. The integrity value is compared to a value written to the config file after successfully running the self-tests during installation.

Appendix B: Compilers

This appendix lists the specific compilers used to generate the Module for the respective operational environments. Note this list does not imply that use of the Module is restricted to only the listed compiler versions And operational environments, as per FIPS 140-2 Implementation Guidance G.5, compliance is maintained for other versions of the respective operational environments and compilers provided the module source code is unchanged. The CMVP makes no statement as to the correct operation of the module when so ported if the specific operational environment is not listed on the validation certificate.

#	Operational Environment (Operating System)	Compilers
1	Photon OS 4.0	gcc 7.3.0
2	Windows Server 2019	Visual Studio 2017
3	Oracle Solaris 11.4	gcc 7.3.0
4	Ubuntu Linux 20.04.1 Server	gcc 9.3.0
5	Windows 10	Visual Studio 2019
6	FreeBSD stable/13	clang 11.0.1
7	Debian 10	gcc 8.3.0
8	Custom Ubuntu Linux 18.04	gcc 8.2
9	macOS 11.5.2	clang 12.0.5

Table 11 – Compilers Used for Each Operational Environment

Appendix C: Glossary

Term	Description
AES	Advanced Encryption Standard
API	Application Program Interface
CAVP	Cryptographic Algorithm Validation Program
CCCS	Canadian Centre for Cyber Security
CMVP	Cryptographic Module Validation Program
CMAC	Cipher-based message authentication code
CSP	Critical Security Parameter
CTR	Counter Mode
DRBG	Deterministic Random Number Generator
DH	Diffie-Hellman
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDH	Elliptic Curve Diffie-Hellman
EdDSA	Edwards Curve Digital Signature Algorithm
EDK	Encrypt/Decrypt Key
FIPS	Federal Information Processing Standards
GCM	Galois/Counter Mode
GMAC	Galois Message Authentication Code
GPC	General Purpose Computer
HKDF	HMAC-based Extract-and-Expand Key Derivation Function
HMAC	Hashed Message Authentication Code
IG	Implementation Guidance
IV	Initialization Vector
KAT	Known answer test
KBKDF	Key Based Key Derivation Function
KDA	Key Derivation Algorithm
KDK	Key Derivation Key

KDF	Key Derivation Function
KEK	Key Encryption Key
KMAC	KECCAK Message Authentication Code
NIST	National Institute of Standards and Technology
NVLAP	National Voluntary Laboratory Accreditation Program
PAA	Processor Algorithm Acceleration
PBKDF2	Password Based Key Derivation Function
PCT	Pairwise Consistency Test
PKG	Private Key Generator
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
SHA-3	Secure Hash Algorithm 3
SHAKE	Secure Hash Algorithm Keccak
SGK	Signature Generation Key
SVK	Signature Verification Key
TDES	Triple Data Encryption Algorithm

Table 12 – Glossary of Terms

Appendix D: Table of References

Reference	Full Specification Name
[ANSI X9.42-2001]	Public Key Cryptography For The Financial Services Industry: Agreement Of Symmetric Keys Using Discrete Logarithm Cryptography
[ANSI X9.63-2001]	Public Key Cryptography For The Financial Services Industry, Key Agreement And Key Transport Using Elliptic Curve Cryptography
[FIPS 140-2]	Security Requirements for Cryptographic modules, May 25, 2001
[FIPS 180-4]	Secure Hash Standard
[FIPS 202]	SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
[FIPS 186-4]	Digital Signature Standard
[FIPS 197]	Advanced Encryption Standard
[FIPS 198-1]	The Keyed-Hash Message Authentication Code (HMAC)
[PKCS#1]	Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
[RFC 5288]	AES Galois Counter Mode (GCM) Cipher Suites for TLS
[RFC 8446]	The Transport Layer Security (TLS) Protocol Version 1.3
[SP 800-38A]	Recommendation for Block Cipher Modes of Operation: Methods and Techniques
[SP 800-38B]	Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication
[SP 800-38C]	Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality
[SP 800-38D]	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
[SP 800-56Ar3]	Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography
[SP 800-56Br2]	Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography
[SP 800-56Cr2]	Recommendation for Key-Derivation Methods in Key-Establishment Schemes
[SP 800-67r2]	Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher
[SP 800-90Ar1]	Recommendation for Random Number Generation Using Deterministic Random Bit Generators

Reference	Full Specification Name
[SP 800-132]	Recommendation for Password-Based Key Derivation
[SP 800-133r2]	Recommendation for Cryptographic Key Generation
[SP 800-135r1]	Recommendation for Existing Application-Specific Key Derivation Functions

Table 13 – Standards and Publications Referenced within this Security Policy

Appendix E: Trademarks

Trademark	Description
Linux®	Linux is the registered trademark of Linus Torvalds in the U.S. and other countries
Unix®	UNIX is a registered trademark of The Open Group
Microsoft Windows®	Windows is a registered trademark of Microsoft Corporation in the United States and other countries

Table 14 – Trademarks Referenced within this Security Policy