wolfCrypt
FIPS 140-3 Non-Proprietary Security Policy

Document Version 1.0

July 9, 2025

wolfSSL Inc.
10016 Edmonds Way
Suite C-300
Edmonds, WA 98020
wolfssl.com
+1 425-245-8247

Table of Contents

List of Tables

List of Figures

1 General

This document defines the Security Policy for the wolfCrypt cryptographic module by wolfSSL Inc., hereafter denoted the Module. The Module meets FIPS 140-3 overall Level 1 requirements, with security levels as described below.

1.1 Cryptographic Boundary

The Module is a cryptography software library, defined as a *software module* per AS02.03 with a multi-chip standalone embodiment. The Module is intended for use by U.S. and Canadian Federal agencies in addition to other markets that require FIPS 140-3 validated cryptographic functionality. The Module version under validation is Software Version v5.2.0.1.

The package/file name is wolfssl-5.7.2-commercial-fips-linuxv5.2.0.1.7z.

Table 1 – Security Levels

ISO/IEC 24759 Section 6. [Number Below]	FIPS 140-3 Section Title	Security Level
1	General	1
2	Cryptographic Module Specification	1
3	Cryptographic Module Interfaces	1
4	Roles, Services, and Authentication	1
5	Software/Firmware Security	1
6	Operational Environment	1
7	Physical Security	N/A
8	Non-Invasive Security	N/A
9	Sensitive Security Parameter Management	1
10	Self-Tests	1
11	Life-Cycle Assurance	1
12	Mitigation of Other Attacks	N/A

In accordance with AS02.05, [ISO19790] §7.7 Physical Security is optional and does not apply to the Module.

In accordance with current CMVP policy, Non-Invasive Security is not applicable.

The Module does not implement attack mitigations outside the scope of [FIPS140-3].

2 Cryptographic Module Specification

The Module conforms to [FIPS140-3_IG] §D.C *References to the Support of Industry Protocols*: while it provides [SP800-56Ar3] conformant schemes and API entry points oriented to TLS and SSH usage, the Module does not contain the full implementation of TLS or SSH. The following caveat is required:

No parts of the TLS and SSH protocols, other than the approved cryptographic algorithms and KDFs, have been tested byt the CAVP or CMVP.

The Module design corresponds to the Module security rules. Security rules enforced by the Module are described in the appropriate context of this document.

2.1 Cryptographic Boundary

Figure 1 depicts the Module operational environment with the cryptographic boundary highlighted in red inclusive of all Module entry points (API calls). The physical perimeter of the module is the general purpose computer on which the software module resides. No components are excluded from [FIPS140-3] requirements. The pre-operational approved integrity test is performed over all components of the cryptographic boundary.

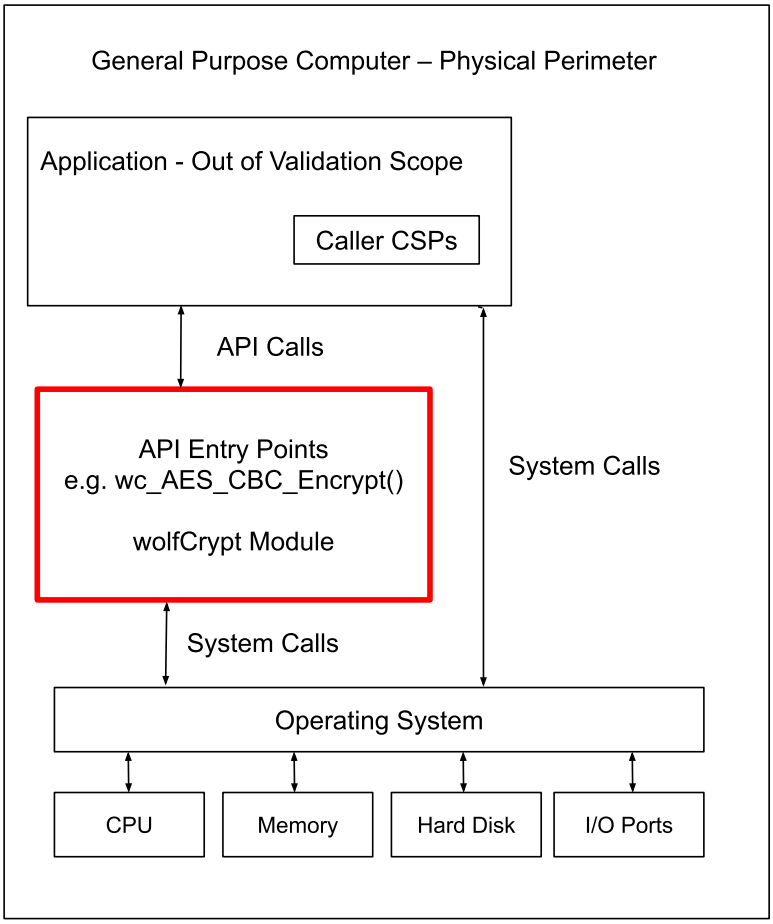

Figure 1 – Module Block Diagram

The source code files listed below result in the corresponding object files that comprise the wolfCrypt Module boundary on each supported operating environment. The extensions of the object file can differ across the environments.

- aes.c: AES algorithm
- aes_asm.s: AES assembler optimizations (Linux)
- aes_asm.asm: AES assembler optimizations (Windows 10)
- cmac.c: CMAC algorithm
- dh.c: Diffie-Hellman
- ecc.c: Elliptic curve cryptography
- fips.c: Pre-operational entry point and API calls
- fips_test.c: Power on self-tests
- hmac.c: HMAC algorithm
- kdf.c: TLS v1.2, v1.3, and SSH v2 KDFs
- random.c: DRBG algorithm

- rsa.c: RSA algorithm
- sha.c: SHA algorithm
- sha256.c: SHA2-256 algorithm
- sha256_asm.s: SHA2-256 assembler optimizations (Linux)
- sha512_asm.s: SHA2-512 assembler optimizations (Linux)
- sha512.c: SHA2-512 algorithm
- sha3.c: SHA-3 algorithm
- wolfcrypt_first.c: First function marking start of cryptographic boundary
- wolfcrypt_last.c: Last function marking end of cryptographic boundary

2.2 Modes of Operation, Security Rules, and Guidance

The Module supports an Approved mode of operation and a non-Approved mode of operation. Approved algorithms are listed in Table 3. Non-Approved algorithms are listed in Table 4. The Module is a cryptographic library providing primitives used by a calling application. The conditions for using the Module in the Approved mode of operation are:

1. The Module is a cryptographic library, and it is intended to be used with a calling application. The calling application is responsible for the usage of the primitives in the correct sequence.
2. The keys used by the Module for cryptographic purposes are determined by the calling application. The calling application is required to provide keys in accordance with [SP800-140Dr2], and to destroy the key structures via the corresponding Free calls after use.
3. With the Module installed and configured in accordance with [UG] instructions and Section 11 of this document, only the algorithms listed in Table 3 (Approved algorithms) and Table 4 (non-Approved algorithms) are available. The Approved mode of operation is invoked by calling the services listed in Table 7 below. The module operates in the non-Approved mode when the service in Table 8 is invoked. The Module is in the Approved mode if the following conditions for algorithm use are met:
    a. Adherence to [FIPS140-3_IG] §C.H *Key/IV Pair Uniqueness Requirements from SP 800-38D*. The Module supports both internal IV generation (for use with the [SP800-56Ar3] compliant KAS API entry points) and external IV generation (for TLS KAS usage). For internal IV generation, the Module complies with C.H scenario 2: users MUST specify an IV length of GCM_NONCE_MID_SZ or greater for internal IV generation (specifying any length less than 96 bits means the Module is no longer in an approved mode of operation). For internal IV generation, C.H requires the calling application to use the Module's internal approved DRBG to generate the random IV. For external IV generation, the Module complies with C.H scenario 1(a), tested per option (ii) under C.H **TLS/DTLS 1.2 protocol IV generation**. The Module performs a check for nonce_explicit rollover, returning an error if that condition is encountered. The module is compatible with TLS/DTLS 1.2 protocol and provides the primitives to support the AES-GCM ciphersuites from [SP800-52r2] Section 3.3.1. If the module's power is lost and then restored, the key used for the AES GCM encryption/decryption shall be re-distributed. This condition is not enforced by the module but is met implicitly. The module does not retain any state across reset or power-cycles: AES-GCM key/IVs are not stored in non-volatile persistent memory (i.e., disk), hence no re-connection can occur without a fresh key establishment operation and the associated SSPs.
    b. ECDSA and RSA signature generation must be used with a SHA-2 or SHA-3 hash function.
    c. RSA signature generation and encryption primitives must use RSA keys with k = 2048, 3072 or 4096 bits or greater. If RSA uses keys with k < 2048, the module enters the non-Approved mode.
    d. The calling process shall adhere to all current [SP800-131Ar2] algorithm usage restrictions.
4. Manual key entry is not supported.

5. The module does not support a non-compliant state. Once the module is compiled per the build instructions defined in the wolfCrypt FIPS 140-3 User Guide, it is not possible for the module to be in a non-compliant state.
6. Data output and control output are inhibited during self-tests, zeroisation, and error states.

The Module obtains the [FIPS140-3_IG] §D.F required key agreement assurances in accordance with [SP800-56Ar3] Section 5.6.2. The module complies to [FIPS140-3_IG] §D.F Scenario 2, path 1.

2.3 Degraded Mode Operation

The Module implements a degraded mode of operation: when a CAST fails, the Module enters an error state. The algorithm CAST status is set to FIPS_CAST_STATE_FAILED and the Module runs all CASTs prior to the first operational use of any algorithm, regardless of the CAST having passed previously. Before exiting the error state, the Module status (reported in the Show Status service) is set to FIPS_MODE_DEGRADED. Upon exiting the error state, the Module enters the degraded mode of operation. The sequence of events is in accordance with AS02.26. The algorithm that failed its CAST, initially triggering the error state, will no longer be available for use in the degraded mode of operation and any algorithms that depend on that algorithm will also be unavailable for use. To recover from the degraded mode operation, the CO *shall* power cycle or reload the Module (equivalent to a power cycle).

2.4 Operational Environment

The TOEPP is the General Purpose Computer on which the module is running on.

Operational testing was performed for the following modifiable Operational Environments (with no restrictions on operational environments configuration):

Table 2 – Tested Operational Environments

#	Operating System	Hardware Platform	Processor	PAA/ Acceleration
1	Linux 4.4 (Ubuntu 16.04 LTS)	Intel Ultrabook 2 in 1	Intel® Core™ i5-5300U CPU @2.30GHz x 4	PAA
2	Linux 4.4 (Ubuntu 16.04 LTS)	Intel Ultrabook 2 in 1	Intel® Core™ i5-5300U CPU @2.30GHz x 4	None
3	Windows 10	Intel Ultrabook 2 in 1	Intel® Core™ i5-5300U CPU @2.30GHz x 4	PAA
4	Windows 10	Intel Ultrabook 2 in 1	Intel® Core™ i5-5300U CPU @2.30GHz x 4	None

A unique set of object files was compiled for each operating environment listed above, with a total of four (4) sets. The Module conforms to [FIPS140-3_IG] §2.3.C *Processor Algorithm Accelerators (PAA) and Processor Algorithm Implementation (PAI).* The Intel Processor AES-NI functions are identified by [FIPS140-3_IG] §2.3.C as a known PAA.

No vendor affirmed operational environments are claimed for this validation of the Module.

2.5 Approved and Allowed Cryptographic Functionality

The Module implements the Approved and allowed cryptographic functions listed in the table below. Equivalent strength in bits is given for each key or algorithm type (as some algorithms do not use or produce keys). The term *s* is used throughout to indicate security strength, following the notation used in the majority of the sources (refer to the notes below Table 3). This table is referenced by Table 9 (SSPs).

Table 3 – Approved Algorithms

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Size(s)/ Key Strength(s)	Use/Function
A2461	AES [FIPS197], [SP800-38A]	AES-CBC, AES-CTR, AES-ECB, AES-OFB	AES-128 (s = 128), AES-192 (s = 192), AES-256 (s = 256)	Encryption, decryption
A2461	AES [SP800-38C], [SP800-38D]	AES-CCM, AES-GCM	AES-128 (s = 128), AES-192 (s = 192), AES-256 (s = 256)	Authenticated encryption, authenticated decryption, message authentication
A2461	AES [SP800-38B]	AES-CMAC	AES-128 (s = 128), AES-192 (s = 192), AES-256 (s = 256)	Message authentication generation, verification
A2461	AES [SP800-38D]	AES-GMAC	AES-192 (s = 192), AES-256 (s = 256)	Message authentication generation, verification
Vendor Affirmed	CKG [SP800-133r2]	§4: Using the Output of a Random Bit Generator §5: Generation of Key Pairs for Asymmetric-Key Algorithms §6.2: Derivation of Symmetric Keys	N/A	Cryptographic key generation
A2461	DSA KeyGen [FIPS186-4]	FFC key generation	L = 2048, N = 256 (s = 112) See Note 4 and Note 7	FFC key generation
A2461	ECDSA KeyGen [FIPS186-4]	Secret generation mode: Extra Bits	P-224 (s ~= 112), P-256 (s ~= 128), P-384 (s ~= 192), P-521 (s ~= 256) See Note 2	ECC key generation
A2461	ECDSA KeyVer [FIPS186-4]	Public Key Validity	P-192 (s < 112), P-224 (s ~= 112), P-256 (s ~= 128), P-384 (s ~= 192), P-521 (s ~= 256) See Note 2	ECC public key validation (curve P-192 is for legacy use only)
A2461	ECDSA SigGen [FIPS186-4]	SigGen (tested with SHA2-224, SHA2-256, SHA2-384, SHA2-512)	P-224 (s ~= 112), P-256 (s ~= 128), P-384 (s ~= 192), P-521 (s ~= 256) See Note 2	ECC signature generation
A2461	ECDSA SigVer [FIPS186-4]	SigVer (tested with SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	P-192 (s < 112), P-224 (s ~= 112), P-256 (s ~= 128), P-384 (s ~= 192), P-521 (s ~= 256) See Note 2	ECC signature verification. (verification with SHA-1 and curve P-192 is for legacy use only)
A2461	Hash DRBG [SP800-90Ar1]	No prediction resistance	SHA2-256 (s = 256)	Random bit generation

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Size(s)/ Key Strength(s)	Use/Function
A2461	HMAC-SHA-1 [FIPS198-1]	Generate HMAC-SHA-1 MAC, with SHA-1 mode	SHA-1 (s = 160)	Generation, verification, message authentication
A2461	HMAC-SHA2 [FIPS198-1]	Generate HMAC-SHA2 MAC with the listed SHA2 modes	SHA2-224 (s = 224), SHA2-256 (s = 256), SHA2-384 (s = 384), SHA2-512 (s = 512)	Generation, verification, message authentication
A2461	HMAC-SHA3 [FIPS198-1]	Generate HMAC-SHA3 MAC with the listed SHA3 modes	SHA3-224 (s = 224), SHA3-256 (s = 256), SHA3-384 (s = 384), SHA3-512 (s = 512)	Generation, verification, message authentication
A2461	KAS-ECC-SSC [SP800-56Ar3]	Scheme: ephemeralUnified KAS Role: Initiator, responder	P-256 (s ~= 128), P-384 (s ~= 192), P-521 (s ~= 256) See Note 2 and Note 3	Shared secret computation
A2461	KAS-FFC-SSC [SP800-56Ar3]	Scheme: dhEphem KAS Role: Initiator, responder	ffdhe2048 (s = 112), ffdhe3072 (112 ≤ s ≤ 128), ffdhe4096 (112 ≤ s ≤ 152), ffdhe6144 (112 ≤ s ≤ 176), ffdhe8192 (112 ≤ s ≤ 200) See Note 5	Shared secret computation
CVL A2461	KDF SSH [SP800-135r1]	Derivation of key blocks for the listed AES cipher key types and hash algorithms	AES-128 (s = 128), AES-192 (s = 192), AES-256 (s = 256); SHA-1 (s = 160), SHA2-256 (s = 256), SHA2-384 (s = 384), SHA2-512 (s = 512)	Key derivation for use with the SSH v2 protocol
CVL A2461	KDF TLS [SP800-135r1]	TLS key derivation using the listed hash algorithms	SHA2-256 (s = 256), SHA2-384 (s = 384), SHA2-512 (s = 512)	Key derivation for use with the TLS v1.2 protocol
CVL A2461	RSA Decryption Primitive [SP800-56Br2]	RSA primitive operations only (no claims of key transport)	k = 2048 (s ~= 112) See Note 6	Key transport primitive RSADP
Vendor Affirmed	RSA Encryption Primitive [SP800-56Br2]	RSA primitive operations only (no claims of key transport)	k=2048 (s ~= 112), k=3072 (s ~= 128), k=4096 (s ~= 152) See Note 6	Key transport primitive RSAEP
A2461	RSA KeyGen [FIPS186-4]	Key generation mode: B.3.3 Primality tests per Table C.2, with listed moduli	k=2048 (s ~= 112), k=3072 (s ~= 128), k=4096 (s ~= 152) See Note 4, Note 6, and Note 10	Key generation
A2461	RSA SigGen [FIPS186-4]	Signature types: PKCS 1.5 and PKCSPSS tested with the listed moduli and the following hash algorithms: SHA2-224, SHA2-256, SHA2-384, SHA2-512	k=2048 (s ~= 112), k=3072 (s ~= 128), k=4096 (s ~= 152) See Note 4, Note 6, and Note 10	Signature generation

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Size(s)/ Key Strength(s)	Use/Function
		Signature types: PKCS 1.5 and PKCSPSS with the listed moduli and the following hash algorithms: SHA3-224, SHA3-256, SHA3-384, SHA3-512 (no ACVP testing is currently available, See Note 9)		
A2461	RSA SigVer [FIPS186-4]	Signature types: PKCS 1.5 and PKCSPSS tested with the listed moduli and the following hash algorithms: SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	k=1024 (s ≤ 112), k=2048 (s ~= 112), k=3072 (s ~= 128), k=4096 (s ~= 152) See Note 4 and Note 6	Signature verification (verification with SHA-1 and modulus length k=1024 is for legacy use only)
		Signature types: PKCS 1.5 and PKCSPSS with the listed moduli and the following hash algorithms: SHA3-224, SHA3-256, SHA3-384, SHA3-512 (no ACVP testing is currently available, See Note 9)		
A2461	SHA-1 [FIPS180-4]	SHA-1 mode listed at right	SHA-1 (s = 160) See Note 1	Message digest generation
A2461	SHA2 [FIPS180-4]	SHA2 modes listed at right	SHA2-224 (s = 224), SHA2-256 (s = 256), SHA2-384 (s = 384), SHA2-512 (s = 512) See Note 1	Message digest generation
A2461	SHA3 [FIPS202]	SHA3 modes listed at right See Note 9	SHA3-224 (s = 224), SHA3-256 (s = 256), SHA3-384 (s = 384), SHA3-512 (s = 512) See Note 1	Message digest generation
CVL A2461	TLS v1.2 KDF [RFC7627]	TLS [RFC7627] key derivation with Extended Master Secret (EMS) support, using the listed hash algorithms	SHA2-256 (s = 256), SHA2-384 (s = 384), SHA2-512 (s = 512)	Key derivation for use with the TLS v1.2 protocol
CVL A2461	TLS v1.3 KDF [RFC8446]	KDF running modes: DHE, PSK, PSK-DHE, using the listed HMAC algorithms	HMAC-SHA2-256 (s = 256), HMAC-SHA2-384 (s = 384)	Key derivation for use with the TLS v1.3 protocol

Note 1: Preimage resistance strength applies to hash algorithms used in DRBG, KDFs. Described also in [SP800-57P1r5] Table 3.

Note 2: Elliptic curve strengths are annotated as approximate (i.e., s ~=) since [SP800-186] Table 1 provides approximate security strengths.

Note 3: Approved elliptic curves for ECC key agreement are given in [SP800-56Ar3] Table 24.

Note 4: In Digital Signature applications, security strength is primarily associated with the asymmetric key pair specification. The hash function used must have equivalent strength equal to or greater than the security strength of the associated key pair.

Note 5: Approved key types for FFC key agreement are given in [SP800-56Ar3] Tables 25, 26. The group notation of Table 26 is used for consistency with CAVP algorithm listings and ACVP capability registration.

Note 6: Estimated security strengths of common RSA moduli are given in [SP800-56Br2] Table 4. IFC key types approved for Digital Signature Generation and Verification are given also in [SP800-57P1r5] Table 2. Equivalent strengths are annotated as approximate (i.e., s ~=) since [SP800-56Br2] Table 4 provides approximate security strengths.

Note 7: Security strength for L=2048/N=256 is determined in accordance with [FIPS140-3_IG] D.B *Strength of SSP Establishment Methods* as y = min(x, N/2), where x is 112 and therefore y = min(112, 128) = 112.

Note 8: The Module is compliant with [FIPS140-3_IG] C.F. extending prime generation and signature generation techniques to the RSA modulus sizes not listed in [FIPS186-4]. With regards to primality testing the Module tests p and q (no auxiliary primes) with 8 rounds of MR Test Only regardless of p and q size. 8 Rounds is three more rounds than the minimum required when p and q are 1024-bits respectively and double the rounds required when p and q are 1536-bits (or greater) respectively. The minimum requirements come from in [FIPS186-5] Table B.1 as related to using a prime generation method that does not rely on the use of auxiliary primes. The Module does more than the minimum required rounds in all cases of p and q size.

Note 9: SHA3 was CAVP tested as standalone functions. SHA3 has not been CAVP tested with ECDSA or RSA as no CAVP testing is currently available. This meets the use and testing requirements in [FIPS140-3_IG] C.C.

Note 10: The Module cannot generate signatures in the Approved mode using a 1024-bit key and cannot generate 1024-bit keys. The Module can verify signatures signed with a 1024-bit key.

Reference sources for the strengths provided in Table 3 are as follows:

- AES (AES-128, AES-192, AES-256): [SP800-57P1r5] Table 2.
- ECC (P-192, P-224, P-256, P-384, P-521): [SP800-186] Table 1.
- FFC (L=1024/N=160, L=2048/N=224, L=2048/N=256, L=3072/N=256): [SP800-57P1r5] Table 2.
- FFC (ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192): [SP800-56Ar3] Table 26.
- IFC (k=1024, k=2048, k=3072, k=4096): [SP800-56Br2] Table 4.
- SHA-1, SHA2 (SHA2-224, SHA2-256, SHA2-384, SHA2-512): [SP800-107] Table 1.
- SHA3 (SHA3-224, SHA3-256, SHA3-384, SHA3-512): [SP800-57P1r5] Table 3.

Table 4 - Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

Algorithm/Function	Use/Function
RSA SigGen using 1024 bit keys	RSA Signature Generation

The Module does not implement the following:

- Non-Approved Algorithms Allowed in the Approved Mode of Operation.
- Non-Approved Algorithms Allowed in the Approved Mode of Operation with No Security Claimed.

3 Cryptographic Module Interfaces

Table 5 defines the Module's [FIPS140-3] logical interfaces; the Module does not interact with physical ports.

Table 5 – Ports and Interfaces

Physical Port	Logical Interface	Data that Passes over Port/Interface
N/A: Internal (call stack)	Control In	API entry point: stack frame including non-sensitive parameters
N/A: Internal (call stack)	Control Out	API call parameters passed by reference for structures allocated by wolfCrypt
N/A: Internal (call stack)	Data In	API call parameters passed by reference or value for cryptographic service input
N/A: Internal (call stack)	Data Out	API call parameters passed by reference for cryptographic service output
N/A: Internal (call stack)	Status Out	API return value: enumerated status resulting from call execution

4 Roles, Services, and Authentication

The Module supports the Cryptographic Officer (CO) operator role, and does not support multiple concurrent operators, a maintenance role or bypass capability.

The Module does not provide an authentication or identification method of its own. The CO role is implicitly identified by the service requested.

The cryptographic module does not support loading software from an external source.

4.1 Approved Services

All services implemented by the Module are listed in Table 6. The calling application may use the Show status service (wolfCrypt_GetStatus_fips call) to determine the status of the Module. A return code of FIPS_MODE_NORMAL means the Module is in a state without errors; the return code FIPS_MODE_DEGRADED means a CAST has failed – see also §2.3 above.

See [UG] for additional information on the cryptographic services listed in this section.

Table 6 – Roles, Service Commands, Input and Output

Role	Service	Input	Output
CO	Digital signature	Sign: signing key; message Verify: signature value; flags; sizes	Status return; Signature value Status return
CO	Generate key pair	FFC, ECC: curve identifier RSA: modulus size	Status return; general digital signature private and public keys
CO	Key agreement	Key structs (key agreement keys); flags	Status return; key agreement shared secret
CO	Key derivation	Key agreement shared secret; flags	Status return; derived keying material
CO	Key transport primitives	Decrypt primitive: Key structs; encapsulated keying material; flags	Status return; keying material
CO	Keyed hash	Keyed hash key	Status return; Tag value
CO	Message digest	Message; flags	Status return; Hash value
CO	Random	DRBG struct (RGB_State); RBG_Seed	Status return; Random value
CO	Self-test	Flags	Status return
CO	Show status	None	Status return (includes Module version)
CO	Symmetric cipher	Encryption or decryption key; flags; plaintext or ciphertext	Status return; Plaintext or ciphertext
CO	Zeroise	FreeRNG destroys the DRBG struct (RGB_State), and is used on module shutdown for the internal DRBG. Key structures allocated by the caller are zeroized by the Free call corresponding to the allocated structure.	Status return

Table 7 describes Module service access to SSPs. In each cell below, the following annotations indicate the type of access by the Module service:

- **G = Generate**: The Module generates or derives the SSP.
- **R = Read**: The SSP is read from the Module (e.g. the SSP is output).
- **W = Write**: The SSP is updated, imported, or written to the Module.
- **E = Execute**: The Module uses the SSP in performing a cryptographic operation.
- **Z = Zeroise**: The Module zeroises the SSP.

The text '--' indicates the table cell contents are intentionally not present.

- f – return a zero on success, or a negative value for error code.
- t - returns FIPS_MODE_INIT (0), FIPS_MODE_NORMAL (1), or FIPS_MODE_FAILED (3)
- k – return a size on success, or a negative value for error code. For k = 1024 for RSA SigGen, the module is in the non-Approved mode.

Table 7 – Approved Services

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access Rights to Keys and/or SSPs	Indicator
Digital signature	Generate or verify digital signatures	ECDSA SigGen, ECDSA SigVer, RSA SigGen, RSA SigVer	DS_SGK/ECC DS_SVK/ECC DS_SGK/IFC DS_SVK/IFC	CO	E,W E,W E,W E,W	k
Generate key pair†	Generate asymmetric key pairs	ECDSA KeyGen, ECDSA KeyVer, RSA KeyGen, DSA KeyGen	GKP_Private/ECC GKP_Public/ECC GKP_Private/IFC GKP_Public/IFC GKP_Private/FFC GKP_Public/FFS	CO	G,W G,W G,W G,W G,W G,W	f
Key agreement†	DH key agreement primitives	KAS-ECC-SSC, KAS-FFC-SSC	KAS_Private/ECC KAS_Public/ECC KAS_Private/FFC KAS_Public/FFC KAS_SS/ECC KAS_SS/FFC	CO	E,W E,W E,W E,W G,R G,R	f
Key derivation†	Derive keying material from a shared secret	KDF SSH, KDF TLS, TLS v1.2 KDF RFC7627, TLS v1.3 KDF	KAS_SS/ECC KAS_SS/FFC KD_DKM, KTS_SS/IFC	CO	E,W E,W G,R E,W	f
Key transport primitives ‡	Encapsulate or decapsulate key material on behalf of the calling process	RSA Decryption Primitive, RSA Encryption Primitive	KTS_KDK/IFC KTS_KEK/IFC KTS_SS/IFC	CO	E, E, R	f
Keyed hash	Generate or verify message integrity	AES-CMAC, AES-GMAC, HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	KH_Key/AES-CMAC KH_Key/AES-GMAC KH_Key/HMAC	CO	E,W E,W E,W	f
Message digest	Generate a message digest	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	--	CO	--	f
Random	Generate random bits using the DRBG	CKG, Hash DRBG	RBG_Seed RBG_State Entropy Input	CO	E,W E,G E,W	f
Self-test	Perform the designated self-test	--	--	CO	--	f
Show status	Provide Module status (includes Module version)	--	--	CO	--	t

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access Rights to Keys and/or SSPs	Indicator
Symmetric cipher	Encrypt or decrypt data, including AEAD modes (CCM, GCM)	AES-CBC, AES-CCM, AES-CTR, AES-ECB, AES-GCM, AES-OFB	SC_EDK/AES	CO	E,W	f
Zeroise	wc_FreeRng_fips() destroys RNG CSPs All functions zeroise CSPs using function ForceZero() (overwriting with zeroes) within the function scope after use Caller stack cleanup is the duty of the application GPC restart/power-cycle clears all CSPs in RAM	--	Entropy Input RBG_State DS_SGK/ECC DS_SGK/IFC GKP_Private/ECC GKP_Private/IFC KAS_Private/ECC KAS_Private/FFC KAS_SS/ECC KAS_SS/FFC KH_Key/AES-CMAC KH_Key/AES-GMAC KH_Key/HMAC RBG_Seed SC_EDK/AES	CO	Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z	f

Note that the caller provides the KAS_Private and KAS_Public keys for shared secret computation; the caller's exchange and assurance of SSPs with the remote participant is outside the scope of the Module.

† Consistent with [FIPS140-3_IG] §9.5.A, available only if the *private_key_read_enable* property is set to TRUE.

‡ Not claiming key transport, but the RSADP and RSAEP are available for interoperation with peers using the TLS protocol stack without an approved cryptography implementation.

Table 8 – Non-Approved Services

Service	Description	Algorithms Accessed	Role	Indicator
Digital signature	Generate digital signatures	RSA SigGen using 1024 bit keys	CO	1024

5 Software/Firmware Security

5.1 Integrity Techniques

The Module uses HMAC-SHA2-256 with a 256-bit key (HMAC Cert. #A2461) as the approved integrity technique. The object files from section 2 are linked into the library. The code section is delimited by two functions that do not perform any actions; they are used only for their addresses. The constant data section is delimited by two constant arrays of unused data; they are only used for their addresses. The code is first added to the hash, then the constant data is added to the hash. The verify hash stored in the code is excluded from the HMAC-SHA2-256 calculation. The calculated HMAC is compared to the HMAC stored in the constant data section.

Before the integrity technique is executed, the Module performs an HMAC-SHA2-256 KAT.

5.2 Initiate on Demand

The operator can initiate the integrity test on demand by reloading the Module.

5.3 Open-Source Parameters

While the Module is not “open-source” since it is only shipped under a commercial license, the open-source practice of source code delivery with a commercial license is standard for the Module. While not required to do so, the Module will abide by [ISO19790] §B.2.5. Please see details in the wolfCrypt FIPS 140-3 User Guide [UG] for the OEs listed on the validation certificate. Details will include information about compiler, compiler configuration settings and methods to compile the source code into an executable form in an approved mode of operation. See also §11.1 below.

6 Operational Environment

Table 2 lists the operational environments on which the Module was tested. The module runs on a modifiable operating environment. The operational environment that is designed to accept functional changes that may contain non-controlled software.

For Linux builds, the configure script provided with the package detects the environment and sets the required flags. On Windows, a header file is provided to set the required flags.

7 Physical Security

N/A. The Module does not implement physical security.

8 Non-Invasive Security

N/A. The Module does not implement non-invasive security mechanisms.

9 Sensitive Security Parameter Management

All Sensitive Security Parameters (SSPs) used by the Module are described in this section, arranged for consistency with Table 7. The text '--' indicates the table cell contents are intentionally not present.

Table 9 – SSPs

Key/SSP Name/Type	Strength[1]	Security Function and Cert Number	Generation	Import/Export	Establishment	Storage	Zeroisation	Use & Related Keys
DS_SGK/ECC	112, 128, 192, 256	ECDSA SigGen #A2461	--	IE1	--	S1	Z1	SigGen (private) key; related to DS_SVK/ECC
DS_SVK/ECC	<112, 112, 128, 192, 256	ECDSA SigVer #A2461	--	IE1	--	S1	Z1	SigVer (public) key, related to DS_SGK/ECC; key type with security strength <112 bits (P-192) used only for legacy signature verification
DS_SGK/IFC	112, 128, 152	RSA SigGen #A2461	--	IE1	--	S1	Z1	SigGen (private) key, related to DS_SVK/IFC
DS_SVK/IFC	<112, 112, 128, 152	RSA SigVer #A2461	--	IE1	--	S1	Z1	SigVer (public) key, related to DS_SGK/IFC; key type with security strength <112 bits (k=1024) used only for legacy signature verification
GKP_Private/ECC	112, 128, 192, 256	ECDSA KeyGen #A2461, ECDSA KeyVer #A2461	G2	IE2	--	S1	Z1	General ECDSA (private) key, related to GKP_Public/ECC
GKP_Public/ECC	<112, 112, 128, 192, 256	ECDSA KeyGen #A2461, ECDSA KeyVer #A2461	G2	IE2	--	S1	Z1	General ECDSA (public) key, related to GKP_Private/ECC; key type with security strength <112 bits (P-192) used only for legacy public key validation (KeyVer)
GKP_Private/FFC	112	DSA KeyGen #A2461	G3	IE2	--	S1	Z1	General DSA (private) key, related to GKP_Public/FFC
GKP_Public/FFC	112	DSA KeyGen #A2461	G3	IE2	--	S1	Z1	General DSA (public) key, related to GKP_Private/FFC
GKP_Private/IFC	112, 128, 152	RSA KeyGen #A2461	G1	IE2	--	S1	Z1	General RSA (private) key, related to GKP_Public/IFC
GKP_Public/IFC	112, 128, 152	RSA KeyGen #A2461	G1	IE2	--	S1	Z1	General RSA (public) key, related to GKP_Private/IFC
KAS_Private/ECC	128, 192, 256	KAS-ECC-SSC #A2461	--	IE1	--	S1	Z1	Key pair component used for shared secret generation
KAS_Public/ECC	128, 192, 256	KAS-ECC-SSC #A2461	--	IE1	--	S1	Z1	Peer key pair component used for shared secret generation
KAS_Private/FFC	112 ≤ s ≤ 200	KAS-FFC-SSC #A2461	--	IE1	--	S1	Z1	Key pair component used for shared secret generation
KAS_Public/FFC	112 ≤ s ≤ 200	KAS-FFC-SSC #A2461	--	IE1	--	S1	Z1	Peer key pair component used for shared secret generation
KAS_SS/ECC	128, 192, 256	KAS-ECC-SSC #A2461	--	--	E2	S1	Z1	Shared secret calculation z output value (for KDF)
KAS_SS/FFC	112 ≤ s ≤ 200	KAS-FFC-SSC #A2461	--	--	E1	S1	Z1	Shared secret calculation z output value (for KDF)

[1] Strength is provided in bits. Please refer to Table 3 and the notes below it for the strength provenance (traceability to applicable standards and special publications).

Key/SSP Name/Type	Strength[1]	Security Function and Cert Number	Generation	Import/Export	Establishment	Storage	Zeroisation	Use & Related Keys
KD_DKM	160, 128, 192, 256, 384, 512	KDF SSH #A2461, KDF TLS #A2461, TLS v1.2 KDF RFC7627 #A2461, TLS v1.3 KDF #A2461	--	--	E3	S1	Z1	Key Derivation derived keying material[2]
KH_Key/ AES-CMAC	128, 192, 256	AES-CMAC #A2461	--	IE1	--	S1	Z1	Keyed Hash key
KH_Key/ AES-GMAC	128, 192, 256	AES-GMAC #A2461	--	IE1	--	S1	Z1	Keyed Hash key
KH_Key/HMAC	112 to 1024 in 8-bit increments	HMAC SHA-1 HMAC SHA2-224 HMAC SHA2-256 HMAC SHA2-384 HMAC SHA2-512 HMAC SHA3-224 HMAC SHA3-256 HMAC SHA3-384 HMAC SHA3-512 #A2461	--	IE1	--	S1	Z1	Keyed Hash key
KTS_KDK/IFC	112, 128, 152	RSA Decryption Primitive #A2461	--	IE1	--	S1	Z1	RSA key de-encapsulation Key (for KDF)
KTS_KEK/IFC	112, 128, 152	RSA Encryption Primitive	--	IE1	--	S1	Z1	RSA key encapsulation Key (for KDF)
KTS_SS/IFC	112, 128, 152	RSA Decryption Primitive #A2461, RSA Encryption Primitive	--	IE1	--	S1	Z1	RSA key transport shared secret (for KDF)
Entropy Input	112	External source (see Table 9)	--	IE1	--	S1	Z1, Z2	Entropy input string
RBG_Seed	112	Hash DRBG #A2461	G4	--	--	S1	Z1, Z2	DRBG seed used for DRBG instantiate and reseed
RBG_State	256	Hash DRBG #A2461	G5	--	E4	S1	Z1, Z2	Hash DRBG (SHA2-256) state: V (440) and C (440)
SC_EDK/AES	128, 192, 256	AES-CBC #A2461, AES-CCM #A2461, AES-CTR #A2461, AES-ECB #A2461, AES-GCM #A2461, AES-OFB #A2461	--	IE1	--	S1	Z1	AES key used for symmetric encryption and decryption (including AES authenticated encryption and decryption)

Legend

Generation
G1: [FIPS186-4] RSA keypair generation, [SP800-133r2] Section 5 compliant
G2: [FIPS186-4] ECDSA keypair generation [SP800-133r2] Section 5 compliant
G3: [FIPS186-4] DSA keypair generation, [SP800-133r2] Section 5 compliant.
G4: [SP800-90B] DRBG seed material, [SP800-133r2] Section 4 compliant

Establishment
E1: [SP800-56Ar3] §5.7.1.1 FFC DH
E2: [SP800-56Ar3] §5.7.1.1 ECC CDH
E3: [SP800-56Cr2] Extract-then-expand KDF
E4: [SP800-90Ar1] Hash_df; Instantiate; Generate; Reseed

Storage
S1: RAM in plaintext

Zeroisation
Z1: Cleared after use via Free, or as part of function cleanup before return.
Z2: the internal RBG is zeroized upon module shutdown.

[2] The separation into specific keys is done outside the scope of the module but must be conformant to [SP800-56Cr2].

G5: [SP800-90Ar1] DRBG state (instantiation or update); [SP800-133r2] Section 4 compliant.

Import/Export
IE1: Call stack (API) input parameters (electronic entry)
IE2: Call stack (API) output parameters (electronic entry)

The Module:

- Produces random values in accordance with [SP800-133r2] Section 4, in that the DRBG output is provided directly as the random output.
- Does not provide any service beyond random value generation for symmetric key generation. SSPs used with symmetric key algorithms are provided by the calling application.
- Produces asymmetric keys in accordance with [SP800-133r2] Section 5, in that all asymmetric keys generated by the module (the Key management service) provide the output of the approved key generation algorithm with no post-processing or manipulation of the generated key pairs. As noted in the previous item, random values used in the asymmetric key generation algorithms are direct outputs of the DRBG. Keys produced by the module use an internal Counter DRBG for which the minimum key size and equivalent security strength is 128 bits.
- Supports symmetric key derivation in accordance with [SP800-133r2] Section 6.2, using the approved and CAVP listed KDF algorithms.

Table 10 – Non-Deterministic Random Number Generation Specification

Entropy Sources	**Minimum Number of Bits of Entropy**	**Details**
Calling application	112 to 256 inclusive	The Module passively obtains entropy via callback functions outside the Module boundary, while exercising no control over the amount or quality of the obtained entropy; the following caveat is applicable to this scenario: *No assurance of the minimum strength of generated SSPs (e.g., keys)*

10 Self-Tests

10.1 Pre-Operational and Conditional Self-Tests

Each time the Module is powered up, it tests that the cryptographic algorithms still operate correctly, and that sensitive data has not been damaged. The pre-operational self-tests are available on demand by reloading the Module.

On instantiation, the Module performs the pre-operational self-tests described below. All cryptographic functions include a check of a self-test flag; a self-test will be invoked if it has not yet been performed. All KATs must complete successfully prior to any other use of cryptography by the Module. The error state is persistent, and no services are available. All attempts to use the Module's services result in the return of a non-zero error code, FIPS_NOT_ALLOWED_E (-197). To recover from an error state, reload the Module into memory. The module error state is called "Err".

Once the Module is powered on and has passed the pre-operational self-tests, calls to any cryptographic algorithm will trigger the CAST on first operational use of the algorithm. The CASTs are available on demand after power-on and can be executed by the Cryptographic Officer (CO) at any time. The CO may optionally invoke any CAST ahead of algorithm use at a more convenient time rather than letting it run automatically on first use. Regardless of the CAST running manually or automatically, once it has passed, the CO may manually re-run any CAST at any time in a periodic fashion. A CAST will no longer run automatically after it has passed the first time.

wolfSSL Inc. *highly recommends* a periodic power cycle or reload of the Module (once in a 24-hour period) as a best security practice. If a periodic power cycle is not possible, a periodic call to the function wc_RunAllCast_fips() is recommended as an alternative (at least once in a 24-hour period). See §10.2 below for details on proper use of the API from a calling application.

Pre-Operational Self-Tests

- Software Integrity: HMAC-SHA2-256 (#A2461) with a 256-bit key.

Conditional Cryptographic Algorithm Self-Tests (CASTs)

- AES-CBC #A2461: Encryption KAT, CBC mode, 128-bit key. Covers self-test requirements for the Table 3 *Algorithm and Standard* entry AES [FIPS197], [SP800-38A].
- AES-CBC #A2461: Decryption KAT, CBC mode, 128-bit key. Covers self-test requirements for the Table 3 *Algorithm and Standard* entry AES [FIPS197], [SP800-38A].
- AES-GCM #A2461: Authenticated encryption KAT, GCM mode, 128-bit key. Covers self-test requirements for the Table 3 *Algorithm and Standard* entries AES [SP800-38B], [SP800-38C], [SP800-38D].
- AES-GCM #A2461: Authenticated decryption KAT, GCM mode, 128-bit key. Covers self-test requirements for the Table 3 *Algorithm and Standard* entries AES [SP800-38B], [SP800-38C], [SP800-38D].
- ECDSA SigGen #A2461: ECDSA signature generation KAT using the P-256 curve.
- ECDSA SigVer #A2461: ECDSA signature verification KAT using the P-256 curve.
- Hash DRBG #A2461: [SP800-90Ar1] §11.3 Instantiate, Generate, Reseed health tests for SHA2-256 Hash DRBG.
- HMAC-SHA-1 #A2461: HMAC-SHA-1 (160-bit key) KAT. Per [FIPS140-3_IG] §10.3.A and §10.3.B, inclusive of corresponding SHA-1 CAST.
- HMAC-SHA2-256 #A2461: HMAC-SHA2-256 (256-bit key) KAT. Per [FIPS140-3_IG] §10.3.A and §10.3.B, inclusive of corresponding SHA2 CAST. Performed prior to its use in the integrity test as required by AS10.20.
- HMAC-SHA2-512 #A2461: HMAC-SHA2-512 (512-bit key) KAT. Per [FIPS140-3_IG] §10.3.A and §10.3.B, inclusive of corresponding SHA2 CAST.
- HMAC-SHA3 #A2461: HMAC-SHA3-256 (256-bit key) KAT. Per [FIPS140-3_IG] §10.3.A and §10.3.B, inclusive of corresponding SHA3 CAST.
- KAS-ECC-SSC #A2461: [SP800-56Ar3] Section 5.7.1.2 primitive “Z” computation KAT, per [FIPS140-3_IG] §D.F, using P-256.
- KAS-FFC-SSC #A2461: [SP800-56Ar3] Section 5.7.1.1 primitive “Z” computation KAT, per [FIPS140-3_IG] §D.F, using L = 2048 N = 256.
- KDF SSH #A2461: [SP800-135r1] SSH v2 KDF KAT. Hash type SHA2-256.
- KDF TLS #A2461: [SP800-135r1] TLS v1.2 KDF KAT. Covers self-test requirements for TLS v1.2 KDF RFC7627. Hash type SHA2-256.
- RSA SigGen #A2461: Signature generation KAT (k = 2048), inclusive of the embedded SHA2-256 self-test.
- RSA SigVer #A2461: Signature verification KAT (k = 2048), inclusive of the embedded SHA2-256 self-test.
- TLS v1.3 KDF #A2461: TLS v1.3 KDF KAT.

Conditional Pairwise Consistency Tests (PCTs)

- DSA KeyGen #A2461: FFC Key Generation Pairwise Consistency Test, performed on FFC key pair generation. FFDHE 2048, FFDHE 3072, FFDHE 4096. [RFC7919]

- ECDSA KeyGen #A2461: ECC Key Generation Pairwise Consistency Test, performed on ECC (ECDSA, KAS-ECC-SSC) key pair generation. Covers self-test requirements for the Table 3 *Algorithm and Standard* entry ECDSA KeyVer [FIPS186-4]. Curves: NIST P-224, NIST P-256, NIST P-384, NIST P-521.
- RSA KeyGen #A2461: RSA Key Generation Pairwise Consistency Test, performed on RSA key pair generation.Key size 2048, 3072, 4096.

10.2 Operator Initiation of Self-Tests

For calling applications, the following is required:

1. Include the library configuration header <wolfssl/options.h> (or user_settings.h via wolfssl/wolfcrypt/settings.h) first.
2. After including the library configuration header, include <wolfssl/wolfcrypt/fips_test.h>, then use the API specified below to execute a given self-test.

The CO may initiate all CASTs at once. The API wc_RunAllCast_fips() is provided as a public API to applications using the Module that have included the headers above in the proper order.

The CO may initiate CASTs individually using the API wc_RunCast_fips(algorithm type) with any of the below algorithm type inputs:

- FIPS_CAST_AES_CBC
- FIPS_CAST_AES_GCM
- FIPS_CAST_HMAC_SHA1
- FIPS_CAST_HMAC_SHA2_256
- FIPS_CAST_HMAC_SHA2_512
- FIPS_CAST_HMAC_SHA3_256
- FIPS_CAST_DRBG
- FIPS_CAST_RSA_SIGN_PKCS1v15
- FIPS_CAST_ECC_CDH
- FIPS_CAST_ECC_PRIMITIVE_Z
- FIPS_CAST_DH_PRIMITIVE_Z
- FIPS_CAST_ECDSA
- FIPS_CAST_KDF_TLS12
- FIPS_CAST_KDF_TLS13
- FIPS_CAST_KDF_SSH

The CO may re-run the pre-operational self-tests at any time after power-on using the public API wolfCrypt_IntegrityTest_fips(). This function always returns a value of zero regardless if the integrity check passed or failed, so the CO *shall* then check the status of the Module using the API wolfCrypt_GetStatus_fips(). The return value of the wolfCrypt_GetStatus_fips() API *shall* then be checked against the status indicators below:

- FIPS_MODE_INIT status indicator value is 0. This indicator means the integrity test has not completed and is likely running in another thread (multi-threaded).
- FIPS_MODE_NORMAL status indicator value is 1. This indicator means the integrity test passed and the Module is in a state without errors.
- FIPS_MODE_FAILED status indicator value is 3. This indicator means the integrity test failed and the Module is unusable. The CO *shall* power cycle or reload (equivalent to power cycle) to restore the Module.

11 Life-Cycle Assurance

11.1 Installation, Initialization, and Startup Procedures

The CO *shall* use the provided FIPS 140-3 User Guide, hereafter referred to as [UG]. A common name for this document is also the Cryptographic Officer Guidance Manual (COGM). [UG] and COGM are one and the same for this Module. The [UG] has a section specific to each Operational Environment (OE), also referred to as the "Tested Configuration", that appears on the Module's certificate and in the above Table 2 – Tested Operational Environments. The instructions provided in the [UG] *shall* be followed or the Module will *not* be considered a FIPS 140-3 validated module.

- The [UG] includes library configuration settings that are: *required*, *allowed*, and *not allowed.*
- For any setting that is not specifically covered, the CO shall contact wolfSSL by emailing "support at wolfssl dot com" for clarification about that settings impact on compiling the compliant module. The [UG] includes details about the toolchain, compiler, compiler configuration settings, and methods to compile the source code into an executable form.
    - While the Module is not "open-source" since it is only shipped under a commercial license, open-source practice of source code delivery with a commercial license is standard for the Module. As such the Module (while not required to) will abide by [ISO19790] §B.2.5: "If the module is open source, specify the compilers and control parameters required to compile the code into an executable format."

The following initialization instructions apply to all use-cases for the Module generically by a consuming application. OE-specific details are covered in the [UG].

- When planning on using the Module, the CO *shall* first include the library settings headers so the application knows how the library was configured.
    - On Linux systems where autotools was used to configure the library (./configure && make), the CO *shall* include <wolfssl/options.h> as the very first header.
    - When working with IDEs or Makefile setups, the CO *shall* include <wolfssl/wolfcrypt/settings.h> as the very first header and ensure that the define WOLFSSL_USER_SETTINGS is set globally at the project level. No other wolfSSL specific build options should be set globally; all configurations will be managed by a custom user_settings.h header that is included anytime WOLFSSL_USER_SETTINGS is defined globally.
        - Once the library configuration settings have been included, only then *shall* the CO include other wolfSSL headers as needed; any other headers *shall* always come after the configuration settings header.

```
static void myFipsCb(int ok, int err, const char* hash)
{
    printf("in my Fips callback, ok = %d, err = %d\n", ok, err);
    printf("message = %s\n", wc_GetErrorString(err));
    printf("hash = %s\n", hash);

    if (err == IN_CORE_FIPS_E) {
        printf("In core integrity hash check failure,"
               "copy above hash\n");
        printf("into verifyCore[] in fips_test.c and rebuild\n");
    }
}
```

Figure 2 – Callback Example

  - If using an informational callback function, the CO *shall* register the function by passing the function pointer to the API as follows: “wolfCrypt_SetCb_fips(myFipsCb);”.

- Prior to operational use of the Module, the CO *shall* register an entropy callback function to load entropy into the Module from an external entropy source. A portable callback is available but must be registered by the application on startup as the entropy source is external to the Module. To register the portable callback function provided with the Module, the application will call “ret = wc_SetSeed_Cb(wc_GenerateSeed);” where “ret” is an integer to capture the status return of the call and should be checked against the value 0 for success or < 0 for failure. A successful register of any entropy callback function is considered the first operational use of the Module and the DBRG CAST will run during registration of the callback.
- When working with a private key, the application *must* programmatically unlock access to private key material via the function call “wolfCrypt_SetPrivateKeyReadEnable_fips(1, WC_KEYTYPE_ALL);”. Once done working with the private key, it is recommended that the application then lock access to private key materials before resuming operations with the function call: “wolfCrypt_SetPrivateKeyReadEnable_fips(0, WC_KEYTYPE_ALL);”.
- The module will operate in the Approved mode of operation as long as only the Approved services listed in Table 7 are invoked. The module transitions to the non-Approved mode when the service in Table 8 is invoked. When using the non-Approved service, no SSPs from the Approved mode are being used.

11.1.1 Linux Installation

Operation of wolfCrypt in the FIPS 140-3 Approved Mode requires the wolfCrypt library version 5.2.0.1. To verify the fingerprint of the package, calculate the SHA2-256 sum using a FIPS 140-2 or FIPS 140-3 validated cryptographic module. The following command serves as an example:

```
$ shasum -a 256 wolfssl-5.2.0.1-commercial-fips-linuxv5.7z
746341ac6d88b0d6de02277af5b86275361ed106c9ec07559aa57508e218b3f5
```

Compare the sum to the sum provided with the package. If the sums do not match stop using the Module and contact wolfSSL.

To unpack the bundle:

```
$ 7za x wolfssl-5.7.2-commercial-fips-linuxv5.2.0.1.7z
7-Zip...
Extracting archive: wolfssl-5.7.2-commercial-fips-linuxv5.2.0.1.7z
...
Enter password (will not be echoed):
```

When prompted, enter the password. The password is provided in the distribution email.

To build and install wolfCrypt in Approved mode:

```
$ ./configure --enable-fips=v5
$ ./wolfcrypt/test/testwolfcrypt
$ sudo make install
```

If you have not received the library with FIPS 140-3 support the *./configure* step will fail. Please contact wolfSSL.

The enable and disable options required for running in the Approved Mode are automatically set by the FIPS enable option. Any encryption algorithms that are not enabled by the configuration **must not** be enabled separately. Options affecting wolfSSL usage are still allowed.

make check will verify the build and that the library is operating correctly. If *make check* fails this probably means the In Core Integrity check has failed, which is expected. To verify this do:

```
$ ./wolfcrypt/test/testwolfcrypt
```

```
...
in my Fips callback, ok = 0, err = 203 message = In Core Integrity check FIPS
error
hash = 622B4F8714276FF8845DD49DD3AA27FF68A8226C50D5651D320D914A5660B3F5
In core integrity hash check failure, copy above hash into verifyCore[] in
fips_test.c and rebuild
```

Copy the value given for the hash in the output, and replace the value of verifyCore[] in *./wolfcrypt/src/fips_test.c* with this new value. After updating verifyCore[], recompile the wolfSSL library by running *make check* again.

The In Core Integrity checksum will vary with compiler versions, runtime library versions, target hardware, and build type.

11.1.2 Windows 10 Installation

Operation of wolfCrypt in the FIPS 140-3 Approved Mode requires the wolfCrypt library version 5.2.0.1. To verify the fingerprint of the package, calculate the SHA2-256 sum using a FIPS 140-2 or FIPS 140-3 validated cryptographic module. The following command serves as an example:

```
% shasum –a 256 wolfssl-5.7.2-commercial-fips-linuxv5.2.0.1.7z
02da35d0a4d6b8e777236fe30da7a6ff93834fb16939ea16da663773f1b34cf0
```

And compare the sum to the sum provided with the package. If for some reason the sums do not match stop using the Module and contact wolfSSL.

A GUI-based 7-zip extraction may be used. To unpack the bundle from a command shell:

```
% 7za x wolfssl-5.7.2-commercial-fips-linuxv5.2.0.1.7z
7-Zip...
Extracting archive: wolfssl-5.7.2-commercial-fips-linuxv5.2.0.1.7z
...
Enter password (will not be echoed):
```

When prompted, enter the password. The password is provided in the distribution email.

To build and install wolfCrypt for use in a FIPS 140-3 approved mode:

1. In Visual Studio open IDE\WIN10\wolfssl-fips.sln.
2. Select the Release DLL and x64 as the build type and target.
3. Build the solution
4. The library should be in the directory IDE\WIN10\DLL Release\x64 as a pair of files: wolfssl-fips.lib is the linking library and wolfssl-fips.dll is the shared library proper, it can be added to your project.
5. In your application project, add a preprocessor macro for HAVE_FIPS. This will ensure the compiler is loading all the correct settings from the user_settings.h header file in the library.
6. Build the solution.
7. Run the code from the DLL Release\x64 directory, the default check failure should be output in the shell.

The enable and disable options required for Approved mode are automatically set in the user settings file mentioned in step 5 above. Any encryption algorithms that are not enabled by the Approved mode **must not** be enabled separately. Options affecting wolfSSL usage are still allowed.

The first run should indicate the In Core Integrity check has failed:

```
in my Fips callback, ok = 0, err = -203 message = In Core Integrity check
FIPS error
hash = 622B4F8714276FF8845DD49DD3AA27FF68A8226C50D5651D320D914A5660B3F5
In core integrity hash check failure, copy above hash into verifyCore[] in
fips_test.c and rebuild
```

The In Core Integrity checksum will vary with compiler versions, runtime library versions, target hardware, and build type.

11.1.3 Linux Secure Startup

The library uses an allocator method to initialize itself after loading, without programmer interaction. The library will perform its own self-test in a thread safe manner.

11.1.4 Windows 10 Secure Startup

The library uses the DllMain() function to initialize itself after loading, without programmer interaction. The library will perform its own self-test in a thread safe manner.

11.2 Administrator Guidance

The CO *shall* use the provided [UG].

11.3 Non-Administrator Guidance

The Module supports the Cryptographic Officer (CO) operator role and does not support non-administrators.

12 Mitigation of Other Attacks

N/A. The Module does not claim mitigation of other attacks.

References

- [FIPS140-3]: FIPS 140-3, *Security Requirements for Cryptographic Modules*, 3/22/2019
- [SP800-140_DTR]: NIST SP 800-140, *FIPS 140-3 Derived Test Requirements (DTR): CMVP Validation Authority Updates to ISO/IEC 24759*, 3/20/2020
- [SP800-140A]: NIST SP 800-140A, *CMVP Documentation Requirements: CMVP Validation Authority Updates to ISO/IEC 24759*, 3/20/2020
- [SP800-140B]: NIST SP 800-140B, *CMVP Security Policy Requirements: CMVP Validation Authority Updates to ISO/IEC 24759 and ISO/IEC 19790 Annex B*, 3/20/2020
- [SP800-140Cr2]: NIST SP 800-140C Rev. 2, *Cryptographic Module Validation Program (CMVP)-Approved Security Functions: CMVP Validation Authority Updates to ISO/IEC 24759*, 7/25/2023

  Supplemental Information: *SP 800-140C: Approved Security Functions*, 7/25/2023
- [SP800-140Dr2]: NIST SP 800-140D Rev. 2, *Cryptographic Module Validation Program (CMVP)-Approved Sensitive Security Parameter Generation and Establishment Methods: CMVP Validation Authority Updates to ISO/IEC 24759*, 7/25/2023

  Supplemental Information: *SP 800-140D: Approved SSP Generation and Establishment Methods*, 25-Jul-2023
- [SP800-140F]: NIST SP 800-140F, *CMVP Approved Non-Invasive Attack Mitigation Test Metrics: CMVP Validation Authority Updates to ISO/IEC 24759*, 3/20/2020
- [FIPS140-3_IG]: *Implementation Guidance for FIPS 140-3 and the Cryptographic Module Validation Program*, 8/1/2023
- [ISO19790]: ISO/IEC 19790:2012 *Information technology – Security techniques – Security requirements for cryptographic modules*, 11/1/2015
- [ISO24759]: ISO/IEC 24759:2017 *Information technology – Security techniques – Test requirements for cryptographic Modules,* 3/1/2017
- [FIPS180-4]: FIPS 180-4, *Secure Hash Standard (SHS)*, 8/4/2015
- [FIPS186-4]: FIPS 186-4, *Digital Signature Standard (DSS)*, 7/19/2013
- [FIPS186-5]: FIPS 186-5, *Digital Signature Standard (DSS)*, 2/3/2023
- [FIPS197]: FIPS 197, *Advanced Encryption Standard (AES)*, 5/09/2023
- [FIPS198-1]: FIPS 198-1, *The Keyed Hash Message Authentication Code (HMAC)*, 7/16/2008
- [FIPS202]: FIPS 202, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, 8/4/2015
- [SP800-38A]: NIST SP 800-38A, *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*, 12/1/2001
- [SP800-38B]: NIST SP 800-38B, *Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication*, 10/6/2016
- [SP800-38C]: NIST SP 800-38C, *Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality*, 7/20/2007
- [SP800-38D]: NIST SP 800-38D, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, 11/28/2007

- [SP800-52r2]: NIST SP 800-52 Rev. 2, *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*, 8/29/2019
- [SP800-56Ar3]: NIST SP 800-56A Rev. 3, *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*, 4/16/2018
- [SP800-56Br2]: NIST SP 800-56B Rev. 2, *Recommendation for Pair-Wise Key-Establishment Using Integer Factorization Cryptography*, 3/21/2019
- [SP800-56Cr2]: NIST SP 800-56C Rev. 2, *Recommendation for Key-Derivation Methods in Key-Establishment Schemes*, 8/18/2020
- [SP800-57P1r5]: NIST SP 800-57 Part 1 Rev. 5, *Recommendation for Key Management: Part 1 – General*, 5/4/2020
- [SP800-90Ar1]: NIST SP 800-90A Rev. 1, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, 6/24/2015
- [SP800-90B]: NIST SP 800-90B, *Recommendation for the Entropy Sources Used for Random Bit Generation*, 1/10/2018
- [SP800-107r1]: NIST SP 800-107 Rev. 1, *Recommendation for Applications Using Approved Hash Algorithms*, 8/24/2012
- [SP800-131Ar2]: NIST SP 800-131A Rev. 2, *Transitioning the Use of Cryptographic Algorithms and Key Lengths*, 3/21/2019
- [SP800-133r2]: NIST SP 800-133 Rev. 2, *Recommendation for Cryptographic Key Generation*, 6/4/2020
- [SP800-135r1]: NIST SP 800-135 Rev. 1, *Recommendation for Existing Application-Specific Key Derivation Functions*, 12/23/2011
- [SP800-186]: NIST SP 800-186, *Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters*, 2/3/2023
- [RFC7627]: *Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension*, 9/16/2015
- [RFC7919]: Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS), 8/10/2016
- [RFC8446]: *The Transport Layer Security (TLS) Protocol Version 1.3*, 8/10/2018
- [UG]: wolfCrypt FIPS 140-3 User Guide, 4/19/2021

Acronyms and Definitions

- AES: Advanced Encryption Standard
- AES-NI: Advanced Encryption Standard New Instructions
- API: Application Programming Interface
- CAST: Cryptographic Algorithm Self-Test
- CAVP: Cryptographic Algorithm Validation Program
- CBC: Cipher-Block Chaining
- CCM: Counter with CBC-MAC
- CMAC: Cipher-based Message Authentication Code
- CMVP: Cryptographic Module Validation Program
- CO: Cryptographic Officer
- CPU: Central Processing Unit
- CSP: Critical Security Parameter
- CTR: Counter-mode
- CVL: Component Validation List
- DH: Diffie-Hellman
- DRBG: Deterministic Random Bit Generator
- DSA: Digital Signature Algorithm
- ECB: Electronic Code Book
- ECC: Elliptic Curve Cryptography
- ECC-CDH: Elliptic Curve Cryptography Cofactor Diffie-Hellman
- ECDH: Elliptic Curve Diffie-Hellman
- ECDSA: Elliptic Curve Digital Signature Algorithm
- EMC: Electromagnetic Compatibility
- EMI: Electromagnetic Interference
- FFC: Finite Field Cryptography
- FIPS: Federal Information Processing Standard
- GCM: Galois/Counter Mode
- GMAC: Galois Message Authentication Code
- GPC: General-Purpose Computer
- HMAC: Keyed-Hash Message Authentication Code
- IG: Implementation Guidance
- IV: Initialization Vector
- KAS: Key Agreement Scheme
- KAT: Known Answer Test
- KDF: Key Derivation Function
- LTS: Long Term Support
- NIST: National Institute of Standards and Technology
- PAA: Processor Algorithm Accelerators
- PCT: Pair-wise Consistency Test
- PSP: Public Security Parameter
- RAM: Random Access Memory

- RNG: Random Number Generator
- RSA: Rivest, Shamir, and Adleman Algorithm
- RSADP: RSA Decryption Primitive
- RSAEP: RSA Encryption Primitive
- SHA: Secure Hash Algorithm
- SHS: Secure Hash Standard
- SSC: Shared Secret Computation
- SSP: Sensitive Security Parameter
- TLS: Transport Layer Security