Corsec Security, Inc.

CorSSL

Software Version: 1.1.1zd.001

FIPS 140-3 Non-Proprietary Security Policy

FIPS Security Level: 1
Document Version: 1.5

Prepared by:

12600 Fair Lakes Circle Suite 210
Fairfax, VA 22033
United States of America

Phone: +1 703 267 6050
www.corsec.com

Table of Contents

List of Tables

List of Figures

1. General

1.1 Overview

1.1.1 Abstract

This is a non-proprietary Cryptographic Module Security Policy for CorSSL (version: 1.1.1zd.001) from Corsec Security, Inc. (Corsec). This Security Policy describes how CorSSL meets the security requirements of Federal Information Processing Standards (FIPS) Publication 140-3, which details the U.S. and Canadian government requirements for cryptographic modules. More information about the FIPS 140-3 standard and validation program is available on the National Institute of Standards and Technology (NIST) and the Canadian Centre for Cyber Security (CCCS) Cryptographic Module Validation Program (CMVP) website at http://csrc.nist.gov/groups/STM/cmvp.

This document also describes how to run the module in a secure Approved mode of operation. This policy was prepared as part of the Level 1 FIPS 140-3 validation of the module. CorSSL is referred to in this document as CorSSL or the module.

1.1.2 References

This document deals only with operations and capabilities of the module in the technical terms of a FIPS 140-3 cryptographic module security policy. More information is available on the module from the following sources:

- The Corsec website www.corsec.com contains information on the full line of services and solutions from Corsec.
- The search page on the CMVP website (https://csrc.nist.gov/Projects/cryptographic-module-validation-program/Validated-Modules/Search) can be used to locate and obtain vendor contact information for technical or sales-related questions about the module.

1.1.3 Document Organization

ISO/IEC 19790 Annex B uses the same section naming convention as *ISO/IEC 19790* section 7 - Security requirements. For example, Annex B section B.2.1 is named “General” and B.2.2 is named “Cryptographic module specification,” which is the same as *ISO/IEC 19790* section 7.1 and section 7.2, respectively. Therefore, the format of this Security Policy is presented in the same order as indicated in Annex B, starting with “General” and ending with “Mitigation of other attacks.” If sections are not applicable, they have been marked as such in this document.

1.2 Security Levels

CorSSL is validated at the FIPS 140-3 section levels shown in the table below.

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
	Overall Level	1

Table 1: Security Levels

2. Cryptographic Module Specification

2.1 Description

2.1.1 Purpose and Use

Corsec Security, Inc is a privately owned company dedicated to assisting organizations through the security certification and validation process. Over the past 22 years, Corsec has grown significantly, becoming a global leader in product and corporate security, offering critical guidance and expertise to meet important business challenges in product security and third-party certifications and security validations, including FIPS 140-2, FIPS 140-3, Common Criteria, and the DoDIN APL.

Corsec's certification methodology helps open doors to new markets and increase revenue for clients with products ranging from mobile phones to satellites. Corsec's broad knowledge safeguards against common pitfalls and thwarts delays, translating to a swift and seamless path to certification. Corsec has created the benchmark for providing business leaders with fast, flexible access to industry knowledge on security certifications and validations.

CorSSL™ 1.1.1zd.001 is a software library providing a C language API for use by other applications requiring cryptographic functionality. CorSSL™ 1.1.1zd.001 offers symmetric encryption/decryption, digital signature generation/verification, hashing, cryptographic key generation, random number generation, message authentication, and key establishment functions to secure data-at-rest/data-in-flight and to support industry-standard secure communications protocols (including TLS 1.2/1.3).

Corsec's CorSSL™ is built upon the OpenSSL 1.1.1 code base, providing engineering teams with a completely compatible cryptographic/protocol engine, allowing quick "drop-in" replacement into any existing OpenSSL 1.1.1-based architecture. CorSSL™ (which includes both the libcrypto crypto library and the libssl protocol library) does not modify the OpenSSL interface, maintaining complete compatibility, and eliminating engineering development time to meet FIPS 140-3 requirements.

2.1.2 Module Type

CorSSL 1.1.1zd.001 is a Software module.

2.1.3 Module Embodiment

CorSSL has a Multi-Chip Standalone embodiment.

2.1.4 Cryptographic Boundary

The cryptographic boundary is the contiguous perimeter that surrounds all memory-mapped functionality provided by the module when loaded and stored in the host platform's memory. The module's cryptographic boundary consists of all functionalities contained within the module's compiled source code. This comprises:

- libcrypto (cryptographic primitives library file)
- libssl (TLS protocol library file)
- libcrypto.hmac (an HMAC digest file for libcrypto integrity checks)
- libssl.hmac (an HMAC digest file for libssl integrity checks)

Figure 1 below is a block diagram of the module executing in memory and its interactions with surrounding software components, as well as the module's cryptographic boundary and Tested Operational Environment's Physical Perimeter (TOEPP). The module utilizes PAA/PAI, which is denoted in the block diagram by including the CPU as part of the cryptographic boundary.

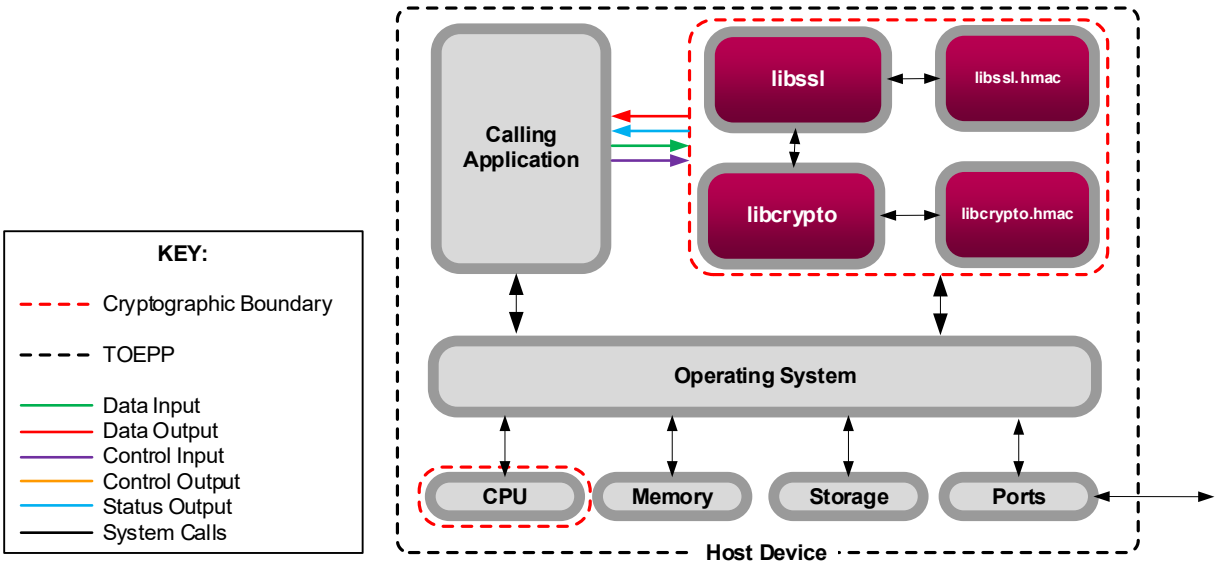

Figure 1: Module Block Diagram (with Cryptographic Boundary)

The module is entirely contained within the physical perimeter.

2.1.5 Tested Operational Environment's Physical Perimeter (TOEPP)

As a software cryptographic module, the TOEPP of the cryptographic module is defined by each host platform on which the module is installed. Figure 2 illustrates a block diagram of a typical GPC (the black dotted line represents the module's physical perimeter).

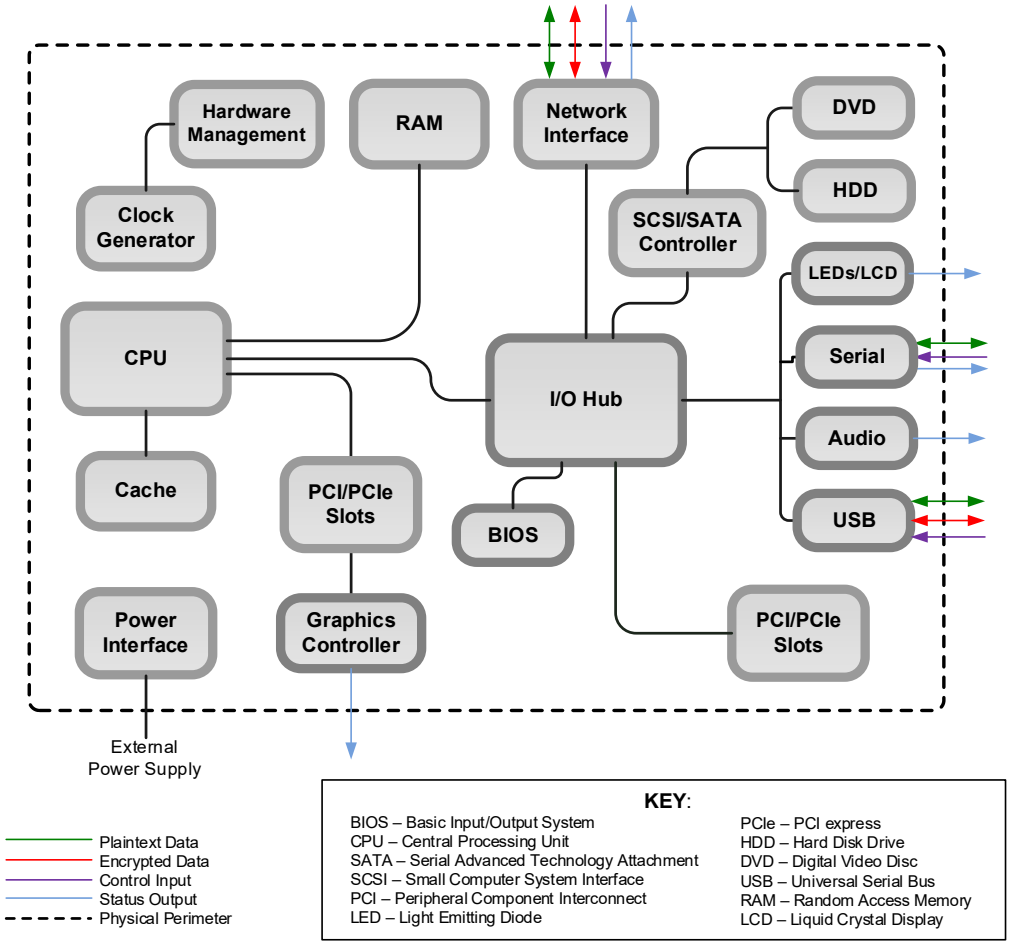

Figure 2: GPC Block Diagram

2.2 Tested and Vendor Affirmed Module Version and Identification

2.2.1 Tested Module Identification – Hardware

This section is only applicable for hardware modules.

2.2.2 Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets)

The table below lists the executable code sets of the module.

Package or File Name	Software/ Firmware Version	Features	Integrity Test
libssl.so.1.1 libcrypto.so.1.1	1.1.1zd.001		Yes
libcrypto-1_1-x64.dll libssl-1_1-x64.dll	1.1.1zd.001		Yes

Table 2: Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets)

2.2.3 Tested Module Identification – Hybrid Disjoint Hardware

This section is only applicable to hybrid modules.

2.2.4 Tested Operational Environments – Software, Firmware, Hybrid

The module was tested and found to be compliant with FIPS 140-3 requirements on the environments listed in the table below.

Operating System	Hardware Platform	Processors	PAA/PAI	Hypervisor or Host OS	Version(s)
Debian 9	Dell PowerEdge R440	Intel Xeon Silver 4214R	Yes		1.1.1zd.001
Debian 9	Dell PowerEdge R440	Intel Xeon Silver 4214R	No		1.1.1zd.001
Windows Server 2022	Dell PowerEdge R760	Intel Xeon Gold 6542Y	Yes	VMware ESXi v8.0	1.1.1zd.001
Windows Server 2022	Dell PowerEdge R760	Intel Xeon Gold 6542Y	No	VMware ESXi v8.0	1.1.1zd.001

Table 3: Tested Operational Environments - Software, Firmware, Hybrid

The module is designed to utilize the AES-NI extended instruction set when available by the host platform's CPU for processor algorithm acceleration (PAA) of its AES implementation.

2.2.5 Vendor-Affirmed Operational Environments – Software, Firmware, Hybrid

There are no vendor-affirmed operational environments claimed.

2.3 Excluded Components

The module does not exclude any components from the requirements.

2.4 Modes of Operation

2.4.1 Modes List and Description

The module supports two modes of operation: Approved and non-Approved. These operational modes are described in the table below.

Mode Name	Description	Type	Status Indicator
Approved	The module is in the Approved mode when all pre-operational self-tests have successfully been completed, and only Approved Services are invoked.	Approved	
Non-approved	The module will operate in the non-Approved mode upon execution of a non-Approved service.	Non-Approved	

Table 4: Modes List and Description

Section 4.3 of this Security Policy lists the services that constitute the Approved mode of operation. Section 4.4 below lists the services that constitute the non-Approved mode. When following the guidance in section 11.3 of this document, CSPs are not shared between Approved and non-Approved services and modes of operation.

2.4.2 Mode Change Instructions and Status

The module alternates on a service-by-service basis between Approved and non-Approved modes of operation. The module will implicitly switch to the non-Approved mode upon execution of a non-Approved service. The module will implicitly switch back to the Approved mode upon execution of an Approved service.
The module does not support degraded operation.

2.5 Algorithms

2.5.1 Approved Algorithms

The module employs cryptographic algorithm implementations from the following sources:

- CorSSL (libcrypto) v1.1.1zd.001(Cert. A7572)
- CorSSL (libssl) v1.1.1zd.001 (Cert. A7573)

Validation certificates for each Approved algorithm are listed in the table below.

Algorithm	CAVP Cert	Properties	Reference
AES-CBC	A7573	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CCM	A7573	Key Length - 128, 192, 256	SP 800-38C
AES-CFB1	A7573	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CFB128	A7573	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CFB8	A7573	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CMAC	A7573	Direction - Generation, Verification Key Length - 128, 192, 256	SP 800-38B
AES-CTR	A7573	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-ECB	A7573	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-GCM	A7573	Direction - Decrypt, Encrypt IV Generation - Internal IV Generation Mode - 8.2.1 Key Length - 128, 192, 256	SP 800-38D
AES-GMAC	A7573	Direction - Decrypt, Encrypt IV Generation - Internal IV Generation Mode - 8.2.1 Key Length - 128, 192, 256	SP 800-38D
AES-KW	A7573	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38F

Algorithm	CAVP Cert	Properties	Reference
AES-KWP	A7573	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38F
AES-OFB	A7573	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-XTS Testing Revision 2.0	A7573	Direction - Decrypt, Encrypt Key Length - 128, 256	SP 800-38E
Counter DRBG	A7573	Prediction Resistance - No, Yes Mode - AES-128, AES-192, AES-256 Derivation Function Enabled - Yes	SP 800-90A Rev. 1
DSA KeyGen (FIPS186-4)	A7573	L - 2048 N - 224, 256	FIPS 186-4
DSA PQGGen (FIPS186-4)	A7573	L - 2048 N - 224, 256 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
DSA SigVer (FIPS186-4)	A7573	L - 1024, 2048, 3072 N - 160, 224, 256 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
ECDSA KeyGen (FIPS186-5)	A7573	Curve - B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 Secret Generation Mode - testing candidates	FIPS 186-5
ECDSA KeyVer (FIPS186-4)	A7573	Curve - B-163, K-163, P-192	FIPS 186-4
ECDSA KeyVer (FIPS186-5)	A7573	Curve - B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	FIPS 186-5
ECDSA SigGen (FIPS186-5)	A7573	Curve - B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-5
ECDSA SigVer (FIPS186-4)	A7573	Curve - B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
ECDSA SigVer (FIPS186-5)	A7573	Curve - B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-5
HMAC-SHA-1	A7573	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-224	A7573	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-256	A7573	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-384	A7573	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-512	A7573	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-224	A7573	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-256	A7573	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-384	A7573	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-512	A7573	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
KAS-ECC-SSC Sp800-56Ar3	A7573	Domain Parameter Generation Methods - B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 Scheme - ephemeralUnified - KAS Role - initiator, responder	SP 800-56A Rev. 3
KAS-FFC-SSC Sp800-56Ar3	A7573	Domain Parameter Generation Methods - FB, FC Scheme - dhEphem - KAS Role - initiator, responder	SP 800-56A Rev. 3
PBKDF	A7573	Iteration Count - Iteration Count: 10-10000 Increment 1 Password Length - Password Length: 8-128 Increment 1	SP 800-132

Algorithm	CAVP Cert	Properties	Reference
RSA KeyGen (FIPS186-5)	A7573	Key Generation Mode - probable Modulo - 2048, 3072, 4096 Primality Tests - 2powSecStr Private Key Format - standard	FIPS 186-5
RSA SigGen (FIPS186-5)	A7573	Modulo - 2048, 3072, 4096 Signature Type - pkcs1v1.5, pss	FIPS 186-5
RSA SigVer (FIPS186-4)	A7573	Signature Type - ANSI X9.31, PKCS 1.5, PKCSPSS Modulo - 1024, 2048, 3072, 4096	FIPS 186-4
RSA SigVer (FIPS186-5)	A7573	Modulo - 2048, 3072, 4096 Signature Type - pkcs1v1.5, pss	FIPS 186-5
SHA-1	A7573	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1	FIPS 180-4
SHA2-224	A7573	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1	FIPS 180-4
SHA2-256	A7573	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1	FIPS 180-4
SHA2-384	A7573	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1	FIPS 180-4
SHA2-512	A7573	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1	FIPS 180-4
SHA3-224	A7573	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1	FIPS 202
SHA3-256	A7573	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1	FIPS 202
SHA3-384	A7573	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1	FIPS 202
SHA3-512	A7573	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1	FIPS 202
SHAKE-128	A7573	Output Length - Output Length: 16-1024 Increment 8	FIPS 202
SHAKE-256	A7573	Output Length - Output Length: 16-1024 Increment 8	FIPS 202
TDES-CBC	A7573	Direction - Decrypt	SP 800-67 Rev. 2
TDES-CFB1	A7573	Direction - Decrypt	SP 800-67 Rev. 2
TDES-CFB64	A7573	Direction - Decrypt	SP 800-67 Rev. 2
TDES-CFB8	A7573	Direction - Decrypt	SP 800-67 Rev. 2
TDES-CMAC	A7573	Direction - Verification	SP 800-67 Rev. 2
TDES-ECB	A7573	Direction - Decrypt	SP 800-67 Rev. 2
TDES-OFB	A7573	Direction - Decrypt	SP 800-67 Rev. 2
TLS v1.2 KDF RFC7627 (CVL)	A7573	Hash Algorithm - SHA2-256, SHA2-384, SHA2-512	SP 800-135 Rev. 1
TLS v1.3 KDF (CVL)	A7572	HMAC Algorithm - SHA2-256, SHA2-384 KDF Running Modes - DHE, PSK, PSK-DHE	SP 800-135 Rev. 1

Table 5: Approved Algorithms

2.5.2 Vendor Affirmed Algorithms

The vendor affirms the following cryptographic security methods:

Name	Properties	Implementation	Reference
CKG Section 4	Key Type:Asymmetric	N/A	SP 800-133 Rev. 2 Section 4 Example 1

Table 6: Vendor-Affirmed Algorithms

2.5.3 Non-Approved, Allowed Algorithms

The table below lists the non-Approved algorithms implemented by the module that are allowed for use in the Approved mode of operation.

N/A for this module.

2.5.4 Non-Approved, Allowed Algorithms with No Security Claimed

The module does not implement any non-approved algorithms allowed in the approved mode of operation for which no security is claimed.

N/A for this module.

2.5.5 Non-Approved, Not Allowed Algorithms

The table below lists the non-Approved algorithms that are not allowed for use in the Approved mode of operation.

Name	Use and Function
AES-GCM (non-compliant)	Authenticated encryption/decryption
AES-OCB	Authenticated encryption/decryption
ANSI X9.31 RNG (with 128-bit AES core)	Random number generation
ARIA	Encryption/decryption
Blake2	Encryption/decryption
Blowfish	Encryption/decryption
Camellia	Encryption/decryption
CAST	Encryption/decryption
CAST5	Encryption/decryption
ChaCha20	Encryption/decryption
DES	Encryption/decryption
Hash DRBG	Random bit generation
HMAC DRBG	Random bit generation
DSA SigGen	Digital signature generation
DH (non-compliant)	Shared secret computation; key pair generation
DSA KeyGen (non-compliant)	Key pair generation
DSA SigVer (non-compliant)	Digital signature verification
ECDH (non-compliant)	Shared secret computation; key pair generation
ECDSA KeyGen (non-compliant)	Key pair generation
ECDSA SigGen (non-compliant)	Digital signature generation
ECDSA SigVer (non-compliant)	Digital signature verification
EdDSA KeyGen	Key pair generation
EdDSA SigGen	Digital signature generation
EdDSA SigVer	Digital signature verification
IDEA	Encryption/decryption
TLS v1.0/v1.1 KDF	Key derivation
HKDF	Key derivation

Name	Use and Function
KBKDF	Key derivation
MD2	Message digest
MD4	Message digest
MD5	Message digest
Poly1305	Message authentication code
RC2	Encryption/decryption
RC4	Encryption/decryption
RC5	Encryption/decryption
RIPEMD	Message digest
RMD160	Message digest
RSA KeyGen (non-compliant)	Key pair generation
RSA SigGen (non-compliant)	Digital signature generation
RSA SigVer (non-compliant)	Digital signature verification
RSA encrypt/decrypt	Asymmetric encryption/decryption
SEED	Encryption/decryption
SHA-1 (non-compliant)	Signature generation for TLS v1.0/v1.1
SM2	Digital signature generation; digital signature verification
SM3	Message digest
SM4	Encryption/decryption
TDES (non-compliant)	Encryption/decryption; MAC generation
Whirlpool	Message digest

Table 7: Non-Approved, Not Allowed Algorithms

2.6 Security Function Implementations

The table below lists the security function implementations for this module.

Name	Type	Description	Properties	Algorithms
AES for Symmetric Encryption	BC-UnAuthEncrypt	The AES key is used for symmetric encryption.	Publication:SP 800-38A	AES-CBC: (A7573) AES-CFB1: (A7573) AES-CFB8: (A7573) AES-CFB128: (A7573) AES-CTR: (A7573) AES-ECB: (A7573) AES-OFB: (A7573)
AES for Symmetric Decryption	BC-UnAuthDecrypt	The AES key is used for symmetric decryption.	Publication:SP 800-38A	AES-CBC: (A7573) AES-CFB1: (A7573) AES-CFB8: (A7573) AES-CFB128: (A7573) AES-CTR: (A7573) AES-ECB: (A7573) AES-OFB: (A7573)
AES for Authenticated Symmetric Encryption	BC-AuthEncrypt	The AES key (for KW/KWP), AES CCM key, or AES GCM key is used for authenticated symmetric encryption.	Publication:SP 800-38C, SP 800-38D, SP 800-38F	AES-GCM: (A7573) AES-CCM: (A7573) AES-KW: (A7573) AES-KWP: (A7573)
AES for Authenticated Symmetric Decryption	BC-AuthDecrypt	The AES key (for KW/KWP), AES CCM key, or AES GCM key is used for authenticated symmetric decryption.	Publication:SP 800-38C, SP 800-38D, SP 800-38F	AES-GCM: (A7573) AES-CCM: (A7573) AES-KW: (A7573) AES-KWP: (A7573)

Name	Type	Description	Properties	Algorithms
AES-XTS for Symmetric Encryption	BC-UnAuthEncrypt	The AES XTS key is used for symmetric encryption.	Publication:SP 800-38E	AES-XTS Testing Revision 2.0: (A7573) AES-ECB: (A7573)
AES-XTS for Symmetric Decryption	BC-UnAuthDecrypt	The AES XTS key is used for symmetric decryption.	Publication:SP 800-38E	AES-XTS Testing Revision 2.0: (A7573) AES-ECB: (A7573)
TDES for Symmetric Decryption (legacy)	BC-UnAuthDecrypt	The TDES key is used for symmetric decryption. For legacy use only.	Publication:SP 800-67 Rev. 2, FIPS 140-3 IG C.M Legacy Algorithms	TDES-CBC: (A7573) TDES-CFB1: (A7573) TDES-CFB64: (A7573) TDES-CFB8: (A7573) TDES-ECB: (A7573) TDES-OFB: (A7573)
AES-CMAC/GMAC for Message Authentication	MAC	The AES CMAC key or AES GMAC key is used for MAC generation and verification.	Publication:SP800-38B, SP 800-38D	AES-CMAC: (A7573) AES-GMAC: (A7573) AES-ECB: (A7573) Counter DRBG: (A7573)
TDES-CMAC for Message Authentication (legacy)	MAC	The TDES CMAC key is used for MAC verification. For legacy use only.	Publication:SP 800-67 Rev. 2, FIPS 140-3 IG C.M Legacy Algorithms	TDES-CMAC: (A7573) TDES-ECB: (A7573)
DRBG	DRBG	Used for generating random bits.	Publication:SP 800-90A	Counter DRBG: (A7573) AES-CTR: (A7573)
HMAC for Message Authentication	MAC	The HMAC key is used for performing keyed hash operations.	Publication:FIPS 198-1	HMAC-SHA-1: (A7573) HMAC-SHA2-224: (A7573) HMAC-SHA2-256: (A7573) HMAC-SHA2-384: (A7573) HMAC-SHA2-512: (A7573) HMAC-SHA3-224: (A7573) HMAC-SHA3-256: (A7573) HMAC-SHA3-384: (A7573) HMAC-SHA3-512: (A7573) SHA-1: (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573) SHA3-224: (A7573) SHA3-256: (A7573) SHA3-384: (A7573) SHA3-512: (A7573)

Name	Type	Description	Properties	Algorithms
SHA/SHAKE for Message Digest	SHA XOF	Used for computing message digests.	Publication:FIPS 180-4	SHA-1: (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573) SHA3-224: (A7573) SHA3-256: (A7573) SHA3-384: (A7573) SHA3-512: (A7573) SHAKE-128: (A7573) SHAKE-256: (A7573)
DSA for Domain Parameter Generation	AsymKeyPair-DomPar	Generates domain parameters for the generation of the DH private component and DH public component.	Publication:FIPS 186-4	DSA PQGGen (FIPS186-4): (A7573) SHA-1: (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573)
DSA for Domain Parameter Verification (legacy)	AsymKeyPair-DomPar	Used to verify DSA domain parameters. For legacy use only.	Publication:FIPS 186-4, FIPS 140-3 IG C.M Legacy Algorithms	DSA PQGVer (FIPS186-4): (A7573) SHA-1: (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573)
DSA for Signature Verification (legacy)	DigSig-SigVer	The DSA public key is used for the verification of digital signatures. For legacy use only.	Publication:FIPS 186-4, FIPS 140-3 IG C.M Legacy Algorithms	DSA SigVer (FIPS186-4): (A7573) SHA-1: (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573)
ECDSA for Key Generation	AsymKeyPair-KeyGen CKG	Used for the generation of the ECDSA public key and ECDSA private key.	Publication:FIPS 186-5	ECDSA KeyGen (FIPS186-5): (A7573) Counter DRBG: (A7573) CKG Section 4: ()
ECDSA for Key Verification	AsymKeyPair-KeyVer	Used to verify the ECDSA public key.	Publication:FIPS 186-5	ECDSA KeyVer (FIPS186-5): (A7573)
ECDSA for Key Verification (legacy)	DigSig-SigVer	The ECDSA public key is used for the verification of digital signatures. For legacy use only.	Publication:FIPS 186-4, FIPS 140-3 IG C.M Legacy Algorithms	ECDSA KeyVer (FIPS186-4): (A7573)
ECDSA for Signature Generation	DigSig-SigGen	The ECDSA private key is used for the generation of digital signatures.	Publication:FIPS 186-5	ECDSA SigGen (FIPS186-5): (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573)
ECDSA for Signature Verification	DigSig-SigVer	The ECDSA public key is used for the verification of digital signatures.	Publication:FIPS 186-5	ECDSA SigVer (FIPS186-5): (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573)

Name	Type	Description	Properties	Algorithms
ECDSA for Signature Verification (legacy)	DigSig-SigVer	The ECDSA public key is used for the verification of digital signatures. For legacy use.	Publication:FIPS 186-4, FIPS 140-3 IG C.M Legacy Algorithms	ECDSA SigVer (FIPS186-4): (A7573) SHA-1: (A7573)
RSA for Key Generation	AsymKeyPair-KeyGen CKG	Used for the generation of the RSA public key and the RSA private key.	Publication:FIPS 186-5	RSA KeyGen (FIPS186-5): (A7573) Counter DRBG: (A7573) CKG Section 4: ()
RSA for Signature Generation	DigSig-SigGen	The RSA private key is used for the generation of digital signatures.	Publication:FIPS 186-5	RSA SigGen (FIPS186-5): (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573)
RSA for Signature Verification	DigSig-SigVer	The RSA public key is used for the verification of digital signatures.	Publication:FIPS 186-5	RSA SigVer (FIPS186-5): (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573)
RSA for Signature Verification (legacy)	DigSig-SigVer	The RSA public key is used for the verification of digital signatures using legacy parameters.	Publication:FIPS 186-4, FIPS 140-3 IG C.M Legacy Algorithms	RSA SigVer (FIPS186-4): (A7573) SHA-1: (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573)
PBKDF	PBKDF	The Passphrase is used for deriving keys via PBKDF.	Publication:SP 800-132	PBKDF: (A7573) SHA-1: (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573) SHA3-224: (A7573) SHA3-256: (A7573) SHA3-384: (A7573) SHA3-512: (A7573)
TLSv1.2-KDF	KAS-135KDF	The key derivation function is used to derive the session and integrity keys used during a TLS v1.2 session.	Publication:SP 800-135 Rev. 1	TLS v1.2 KDF RFC7627: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573) HMAC-SHA2-256: (A7573) HMAC-SHA2-384: (A7573) HMAC-SHA2-512: (A7573)
TLSv1.3-KDF	KAS-135KDF	The key derivation function is used to derive the session and integrity keys used during a TLS v1.3 session.	Publication:SP 800-135 Rev. 1	HMAC-SHA2-256: (A7573) HMAC-SHA2-384: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) TLS v1.3 KDF: (A7572)

Name	Type	Description	Properties	Algorithms
DH Shared Secret Computation	KAS-SSC CKG AsymKeyPair-KeyGen	Uses the DH private component and DH public component to compute the shared secret.	Publication:SP 800-56A Rev. 3	KAS-FFC-SSC Sp800-56Ar3: (A7573) DSA KeyGen (FIPS186-4): (A7573) DSA PQGGen (FIPS186-4): (A7573) Counter DRBG: (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573) CKG Section 4: ()
ECDH Shared Secret Computation	KAS-SSC CKG AsymKeyPair-KeyGen	Uses the ECDH private component and ECDH public component to compute the shared secret.	Publication:SP 800-56A Rev. 3	KAS-ECC-SSC Sp800-56Ar3: (A7573) ECDSA KeyGen (FIPS186-5): (A7573) Counter DRBG: (A7573) SHA2-224: (A7573) SHA2-256: (A7573) SHA2-384: (A7573) SHA2-512: (A7573) CKG Section 4: ()

Table 8: Security Function Implementations

2.7 Algorithm Specific Information

The following subsections provide algorithm-specific information. Additionally, algorithms designated as “Legacy” can only be used on data that was generated prior to the Legacy Date specified in *FIPS 140-3 IG C.M*.

2.7.1 AES-GCM

The module supports internal IV generation using its Approved DRBG. The IV is at least 96 bits in length per section 8.2.2 of *NIST SP 800-38D*, and the Approved DRBG generates outputs such that the (key, IV) pair collision probability is less than 2^{-32} per section 8 of *NIST SP 800-38D*.

The module also supports AES GCM encryption is used in the context of the TLS protocol versions 1.2 and 1.3. To meet the AES GCM (key/IV) pair uniqueness requirements from *NIST SP 800-38D*, the module complies with *FIPS 140-3 IG* C.H as follows:

- For TLS v1.2, the module supports acceptable AES GCM cipher suites from section 3.3.1.1 of *NIST SP 800-52rev2*.

  The mechanism for IV generation falls into scenario 1 in *FIPS 140-3 IG* C.H and is compliant with *RFC 5288*. The 64-bit counter portion of the IV is strictly increasing. The module explicitly ensures that the counter does not exhaust the maximum number of possible values of 2^{64}-1 for a given session key. If this exhaustion condition is observed, the module will return an error indication to the calling application, which will then need to either abort the connection, or trigger a handshake to establish a new encryption key. It is the responsibility of the module operator (i.e., the first party, client, or server) to trigger this handshake when this condition is encountered.

- For TLS v1.3, the module supports acceptable AES GCM cipher suites from section 3.3.1.2 of *NIST SP 800-52rev2*. The protocol's implementation is contained within the boundary of the module, and the generated IV is only used in the context of the AES GCM encryption executing the provisions of the TLS 1.3 protocol.

  The mechanism for IV generation falls into scenario 1 in *FIPS 140-3 IG* C.H and is compliant with *RFC 8446*. Each session employs a "per-record nonce", a 64-bit sequence number (or IV) maintained separately for reading and writing records. Each sequence number is set to 0 at the beginning of a connection and whenever the key is changed (the first record transmitted under a particular traffic key uses sequence number 0), and the appropriate sequence number is incremented by one after reading or writing each record. Because the size of sequence numbers is 64 bits, they should not wrap. If a sequence number needs to wrap, it is the responsibility of the module operator to either re-key with a new key for AES-GCM or terminate the connection.

In case the module's power is lost and then restored, the calling application is responsible for ensuring that a new key for use with the AES-GCM encryption/decryption shall be established. This condition is not enforced by the module but is met implicitly. The module does not retain any state across resets or power-cycles, and AES-GCM key/IVs are not stored in non-volatile persistent memory (i.e., disk). Hence, no reconnection can occur without a fresh key establishment operation and the associated SSPs.

When a GCM IV is used for decryption, the responsibility for the IV generation lies with the party that performs the AES GCM encryption.

2.7.2 AES-XTS

The AES-XTS mode shall only be used for the cryptographic protection of data on storage devices. The AES-XTS shall not be used for other purposes, such as the encryption of data in transit.

The module implements a check to ensure that the two AES keys used in the XTS-AES algorithm are not identical.

AES-XTS keys (i.e., Key_1 and Key_2) entered into the module shall be generated and/or established independently according to NIST SP 800-133rev2, Section 6.3 for an approved use of AES-XTS.

2.7.3 DSA

The module offers DSA KeyGen and DSA PQGGen functionality compliant with *FIPS PUB 186-4*. These functions are used exclusively to support FFC key establishment when using FC and FC domain parameter generation methods. This is an Approved key agreement schemes compliant with *NIST SP 800-56A Rev. 3*. This allowance is in accordance with IG C.K resolution #3.

FIPS 140-3 IG C.K, Resolution 6 allow *FIPS 186-4* DSA SigVer for legacy use only.

2.7.4 ECDSA

ECDSA SigVer (FIPS 186-4) includes legacy parameters, which include curves with ≤ 112 bits of security strength (B-163, K-163, and P-192) and SHA-1 for the underlying hash algorithm. *FIPS 140-3 IG* C.K, Resolution 6 and Additional Comment 2 allow *FIPS 186-4* parameters to only be used for verifying ECDSA signatures generated prior to February 5, 2024.

2.7.5 KAS

The module does not establish SSPs using an Approved key agreement scheme (KAS). However, it does offer some or all the underlying KAS cryptographic functionality to be used by a calling application as part of an Approved KAS. Refer to section 2.10.1 below for additional details.

2.7.6 KTS

The module does not establish SSPs using an Approved key transport scheme (KTS). However, it does offer approved authenticated algorithms that can be used by an external operator/application as part of an approved KTS. Refer to section 2.10.2 below for additional details.

2.7.7 PBKDF

The module uses PBKDF2 option 1a from section 5.4 of *NIST SP 800-132*. In accordance to *NIST SP 800-132*, the following requirements shall be met.

- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the module's Approved DRBG.
- The iteration count shall be selected as large as possible, as long as the time required to generate the resultant key is acceptable for module operators. The minimum iteration count shall be 1000.
- The length of the password/passphrase used in the PBKDF shall be at least 20 characters, and shall consist of lower-case, upper-case, and numeric characters. The upper bound for the probability of guessing the value is estimated to be $1/62^{20} = 10^{-36}$, which is less than 2^{-112}.
- Passwords/passphrases (used as an input for the PBKDF) shall not be used as cryptographic keys.
- Keys derived from passwords/passphrases must only be used in storage applications.

2.7.8 RSA

RSA SigVer (FIPS186-4) includes legacy parameters, which include keys with ≤ 112 bits of security strength (1024 bits) and SHA-1 for the underlying hash algorithm. *FIPS 140-3 IG* C.K, Resolution 6 and Additional Comment 2 allow *FIPS 186-4* parameters to only be used for verify RSA signatures generated prior to February 5, 2024.

2.8 RBG and Entropy

The module invokes a GET command to obtain entropy for random number generation (the module requests 256 bits of entropy from the calling application per request), and then passively receives entropy from the calling application while having no knowledge of the entropy source and exercising no control over the amount or the quality of the obtained entropy. This is the logical equivalent of a LOAD command.

The calling application and its entropy sources are located outside the module's cryptographic boundary but within its TOEPP. Thus, there is no assurance of the minimum strength of the generated SSPs. The calling application is responsible for using entropy sources that meet the minimum security strength of 112 bits required for the Approved DRBGs as shown in *NIST SP 800-90Arev1,* Table 2 and Table 3. This entropy is supplied by means of callback functions. Those functions must return an error if the minimum entropy strength cannot be met.

A minimum of 256 bits of entropy is required to generate SSPs with up to 256 bits of strength. By default, the module uses the Approved Counter DRBG (AES-256-CTR; derivation function) to generate random values for other security functions.

As allowed by Additional Comment #12 of *FIPS 140-3 IG* 9.3.A, this complies with an earlier version of Resolution 2(b), which can be found at the following URL:

https://csrc.nist.gov/CSRC/media/Projects/cryptographic-module-validation-program/documents/IG%209.3.A%20Resolution%202b%5BMarch%2026%202024%5D.pdf

2.9 Key Generation

The module supports the generation of cryptographic keys as follows:

- ECDSA and RSA key pairs (per section 5.1 of *NIST SP 800-133rev2)*
- DH and ECDH key pairs (per section 5.2 of *NIST SP 800-133rev2)*

As specified in section 4 of *NIST SP 800-133rev2*, the cryptographic module uses its Approved DRBG to generate seeds used for asymmetric key generation. The generated seed is an unmodified output from the DRBG.

2.10 Key Establishment

The cryptographic module provides the cryptographic primitives necessary to support key agreement schemes and key transport methods for use by the calling application to establish keys.

2.10.1 Key Agreement Information

The module offers as a service to a calling application the CAVP-tested KAS SSC that map to *FIPS 140-3 IG* D.F Scenario 2, path (1) without establishing a key/SSP to be used by the module for cryptographic protection:

- KAS-ECC-SSC Sp800-56Ar3
- KAS-FFC-SSC Sp800-56Ar3

The module performs assurances for its key agreement schemes as specified in the following sections of *NIST SP 800-56Arev3:*

- Section 5.5.2 (for assurances of domain parameter validity)
- Section 5.6.2.1 (for assurances required by the key pair owner)

The module includes the capability to provide the required recipient assurance of ephemeral public key validity specified in section 5.6.2.2.2 of *NIST SP 800-56Arev3*. However, since public keys from other modules are not received directly by this module (those keys are received by the calling application), the module has no knowledge of when a public key is received. Invocation of the proper module services to validate another module's public key is the responsibility of the calling application.

Key confirmation is not supported by the module.

These methods are not used to establish keys into the module.

2.10.2 Key Transport Information

The module offers as a service to a calling application CAVP-tested algorithms specified in the list below without establishing a key/SSP to be used by the module for cryptographic protection.

- To support authenticated encryption/decryption as a unified service:
  - AES-CCM
  - AES-GCM
  - AES-KW
  - AES-KWP

- To support unauthenticated encryption/decryption and authentication as separate services:
  - AES + CMAC
  - AES + GMAC
  - AES + HMAC

- To support unauthenticated decryption as a legacy service:
  - AES[1]
  - Three-key TDES[2]

These methods are not used to establish keys into the module.

2.11 Industry Protocols

The module uses the following industry protocols:

- TLS 1.2
- TLS 1.3

[1] Per *FIPS 140-3 IG* D.G, key unwrapping using any Approved mode of AES is allowed in the Approved mode.
[2] Per *FIPS 140-3 IG* D.G, key unwrapping using any Approved mode of TDES is allowed in the Approved mode.

The KDFs associated with these protocols shall only be used within the context of their respective protocols. No parts of these protocols, other than the Approved cryptographic algorithms and the KDFs, have been tested by the CAVP and CMVP.

3. Cryptographic Module Interfaces

3.1 Ports and Interfaces

The module supports the following four logical interfaces:

- Data Input
- Data Output
- Control Input
- Status Output

As a software library, the cryptographic module has no direct access to any of the host platform's physical ports, as it communicates only to the calling application via its well-defined API. A mapping of the FIPS-defined interface to the module's physical ports and logical interfaces can be found in the table below.

Note that the module does not output control information, and this has no specified control output interface.

Physical Port	Logical Interface(s)	Data That Passes
N/A	Data Input	Logical interface is defined as API input arguments that provide input data for processing. This includes data to be encrypted, decrypted, signed, verified, and hashed, keys to be used in cryptographic services, random seed material for the DRBG of the module, keying material used as input to key establishment services, and intermediate data required for services.
N/A	Data Output	Logical interface is defined as API output arguments that return generated or processed data back to the caller. This includes data that has been encrypted/decrypted/verified, digital signatures, hashes, random values generated by the DRBG of the module, keys established using key establishment methods of the module, and key components/intermediate data/traffic (client and server data and messages).
N/A	Control Input	Logical interface is defined as API input arguments that are used to initialize and control the operation of the module. This includes API commands invoking cryptographic services, modes, key sizes, etc. used with cryptographic services.
N/A	Control Output	N/A, the module does not implement a Control Output interface.
N/A	Status Output	Logical interface is defined as API call return values. This includes status information regarding the module or invoked service/operation.

Table 9: Ports and Interfaces

4. Roles, Services, and Authentication

4.1 Authentication Methods

The module does not support authentication methods; operators implicitly assume an authorized role based on the service selected.

4.2 Roles

The module supports the roles listed in the table below.

Name	Type	Operator Type	Authentication Methods
Crypto Officer	Role	CO	
User	Role	User	

Table 10: Roles

The module does not support multiple concurrent operators. The calling application that loaded the module is its only operator.

4.3 Approved Services

Descriptions of the services available are provided in the table below. Access rights for each SSP are specified using the following notation:

- 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 = Zeroize: The module zeroizes the SSP.

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Show Status	Returns the status of the module.	N/A	API call parameters	Current operational status		Crypto Officer
Perform self-tests on-demand	Performs pre-operational self-tests by unloading and then re-initializing the module or by invoking API.	N/A	Re-instantiate module; API call parameters	Status		Crypto Officer

Zeroize	Zeroizes and de-allocates memory containing sensitive data.	N/A	Restart calling application; reboot or power-cycle host platform	None		Crypto Officer - AES key: Z - AES CCM key: Z - AES GCM key: Z - AES XTS key: Z - AES CMAC key: Z - AES GMAC key: Z - TDES key: Z - TDES CMAC key: Z - HMAC key: Z - DSA public key: Z - ECDSA private key: Z - ECDSA public key: Z - RSA private key: Z - RSA public key: Z - DH private component: Z - DH public component: Z - DH shared secret: Z - ECDH private component: Z - ECDH public component: Z - ECDH shared secret: Z - TLS pre-master secret: Z - TLS master secret: Z -

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						Passphrase: Z - AES GCM IV: Z - DRBG entropy input: Z - DRBG seed: Z - DRBG 'V' value: Z - DRBG 'Key' value: Z
Show versioning information	Returns module versioning information.	N/A	API call parameters	Module name, version		Crypto Officer
Perform symmetric encryption	Encrypts data using an approved mode of AES.	EVP_cipher_get_service_indicator( )	API call parameters, key, plaintext	Status, ciphertext	AES for Symmetric Encryption AES-XTS for Symmetric Encryption	User - AES key: W,E - AES XTS key: W,E
Perform symmetric decryption	Decrypts data using an approved ode of AES.	EVP_cipher_get_service_indicator( )	API call parameters, key, ciphertext	Status, plaintext	AES for Symmetric Decryption AES-XTS for Symmetric Decryption TDES for Symmetric Decryption (legacy)	User - AES key: W,E - AES XTS key: W,E - TDES key: W,E
Generate symmetric digest	Generates symmetric digest.	For CMAC: CMAC_get_service_indicator() For GMAC: EVP_cipher_get_service_indicator( )	API call parameters, key, plaintext	Status, digest	AES-CMAC/GMAC for Message Authentication	User - AES CMAC key: W,E - AES GMAC key: W,E
Verify symmetric digest	Verifies symmetric digest.	For CMAC: CMAC_get_service_indicator() For GMAC: EVP_cipher_get_service_indicator( )	API call parameters, digest	Status	AES-CMAC/GMAC for Message Authentication TDES-CMAC for Message Authentication (legacy)	User - AES CMAC key: W,E - AES GMAC key: W,E - TDES CMAC key: W,E
Perform authenticated symmetric encryption	Encrypts data using AES-GCM or AES-CCM.	EVP_cipher_get_service_indicator( )	API call parameters, key, plaintext	Status, ciphertext	AES for Authenticated Symmetric Encryption	User - AES key: W,E - AES CCM key: W,E - AES GCM key: W,E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Perform authenticated symmetric decryption	Decrypts data using AES-GCM or AES-CCM.	EVP_cipher_get_service_indicator()	API call parameters, key, ciphertext	Status, plaintext	AES for Authenticated Symmetric Decryption	User - AES key: W,E - AES CCM key: W,E - AES GCM key: W,E
Generate random number	Generates random bits using DRBG.	DRBG_get_service_indicator()	API call parameters	Status, random bits	DRBG	User - DRBG entropy input: W,E - DRBG seed: G,E - DRBG 'V' value: G,E - DRBG 'Key' value: G,E
Perform keyed hash operation	Computes a message authentication code.	HMAC_get_service_indicator()	API call parameters, key, message	Status, MAC	HMAC for Message Authentication	User - HMAC key: W,E
Perform hash operation	Computes a message digest.	EVP_Digest_get_service_indicator()	API call parameters, message	Status, hash	SHA/SHAKE for Message Digest	User
Generate DSA domain parameters	Generates FIPS-186-4 domain parameters to be used for KAS-FFC-SSC.	DSA_get_service_indicator()	API call, parameters	Status, domain parameters	DSA for Domain Parameter Generation	User
Verify domain parameters	Verify DSA domain parameters.	DSA_get_service_indicator()	API call parameters	Status	DSA for Domain Parameter Verification (legacy)	User

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Generate asymmetric key pair	Generates a public/private key pair.	For RSA: RSA_key_get_service_indicator() For ECDSA: EC_key_get_service_indicator()	API call parameters	Status, key pair	ECDSA for Key Generation RSA for Key Generation	User - ECDSA private key: G - ECDSA public key: G - RSA private key: G - RSA public key: G - DRBG entropy input: W,E - DRBG seed: G,E - DRBG 'V' value: G,E - DRBG 'Key' value: G,E
Verify ECDSA public key	Verifies an ECDSA public key.	EC_key_get_service_indicator()	API call parameters, key	Status	ECDSA for Key Verification ECDSA for Key Verification (legacy)	User - ECDSA public key: W
Generate digital signature	Generates a digital signature.	EVP_Digest_get_service_indicator( )	API call parameters, key, message	Status, signature	ECDSA for Signature Generation RSA for Signature Generation	User - ECDSA private key: W,E - RSA private key: W,E
Verify digital signature	Verifies a digital signature.	EVP_Digest_get_service_indicator( )	API call parameters, key, signature, message	Status	DSA for Signature Verification (legacy) ECDSA for Signature Verification ECDSA for Signature Verification (legacy) RSA for Signature Verification RSA for Signature Verification (legacy)	User - DSA public key: W,E - ECDSA public key: W,E - RSA public key: W,E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Compute shared secret	Computes DH/ECDH shared secret.	For DH parameter & key generation: DSA_get_service_indicator() For KAS-FFC-SSC: DH_compute_key() For KAS-ECC-SSC: EC_key_get_service_indicator()	API call parameters	Status, shared secret	DH Shared Secret Computation ECDH Shared Secret Computation	User - DH private component: G,E - DH public component: G,W,E - DH shared secret: G,R - ECDH private component: G,E - ECDH public component: G,W,E - ECDH shared secret: G,R
Derive key via SP 800-135 Rev. 1 KDF	Derives keying material via TLS v1.2 or TLS v1.3 KDF.	For TLS v1.2 KDF: TLSKDF_get_service_indicator() For TLS v1.3 KDF: TLS1_3_kdf_get_service_indicator()	API call parameters	Status, derived keying material	TLSv1.2-KDF TLSv1.3-KDF	User - DH shared secret: W,E - ECDH shared secret: W,E - TLS pre-master secret: G,R - AES key: G,R - AES CCM key: G,R - AES GCM key: G,R - AES XTS key: G,R - AES CMAC key: G,R - AES GMAC key: G,R - HMAC key: G,R

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Derive key via PBKDF2	Derives keying material from PBKDF2.	PBKDF_get_service_indicator()	API call parameters, password	Status, derived keying material	PBKDF	User - Passphrase: W,E - AES key: G,R - AES CCM key: G,R - AES GCM key: G,R - AES XTS key: G,R - AES CMAC key: G,R - AES GMAC key: G,R - TDES key: G,R - TDES CMAC key: G,R - HMAC key: G,R

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Establish TLS connection	Establishes a TLS session. Please refer to Section 11.3 for a list of approved TLS cipher suites.	For KAS-ECC-SSC: EC_key_get_service_indicator() For cipher: EVP_cipher_get_service_indicator() For message authentication: HMAC_get_service_indicator() For ECDSA certificate verification: EC_key_get_service_indicator() For RSA certificate verification: RSA_key_get_service_indicator() For TLS v1.2 KDF: TLSKDF_get_service_indicator() For TLS v1.3 KDF: TLS1_3_kdf_get_service_indicator()	API call parameters, TLS configuration, TLS keys	Status, TLS connection information	ECDH Shared Secret Computation TLSv1.2-KDF TLSv1.3-KDF AES for Authenticated Symmetric Encryption AES for Authenticated Symmetric Decryption AES for Symmetric Encryption AES for Symmetric Decryption HMAC for Message Authentication ECDSA for Signature Generation ECDSA for Signature Verification ECDSA for Signature Verification (legacy) RSA for Signature Generation RSA for Signature Verification RSA for Signature Verification (legacy)	User - ECDH private component: G,E - ECDH public component: G,R,E - ECDH shared secret: G,E - TLS pre-master secret: G,E - TLS master secret: G,E - AES key: G,E - AES CCM key: G,E - AES GCM key: G,E - HMAC key: G,E - ECDSA private key: W,E - ECDSA public key: R,W,E - RSA private key: W,E - RSA public key: R,W,E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Certificate Management	Performs certificate management services.	For ECDSA certificate verification: EC_key_get_service_indicator() For RSA certificate verification: RSA_key_get_service_indicator() For cipher: EVP_cipher_get_service_indicator() For hash: EVP_Digest_get_service_indicator()	AES key, API call parameters, private keys, public keys, certificates, certificate data	Status	AES for Symmetric Encryption AES for Symmetric Decryption AES for Authenticated Symmetric Encryption AES for Authenticated Symmetric Decryption ECDSA for Signature Generation ECDSA for Signature Verification ECDSA for Signature Verification (legacy) RSA for Signature Generation RSA for Signature Verification RSA for Signature Verification (legacy)	User - AES key: W,E - AES CCM key: W,E - AES GCM key: W,E - ECDSA private key: R,W,E - ECDSA public key: R,W,E - RSA private key: R,W,E - RSA public key: R,W,E

Table 11: Approved Services

This module is a software library that provides cryptographic functionality to calling applications. As such, the security functions provided by the module are considered the module's security services. Indicators for Approved services (in the case of this module, those security functions with algorithm validation certificates and all required self-tests) are provided via API return value.

When invoking a security function, the calling application provides inputs via an internal structure, or "context". Upon each service invocation, the module will determine if the invoked security function is an Approved service. To access the resulting value, the calling application must pass the finalized context to the indicator API associated with that security function (note the indicator check must be performed prior to any context cleanup is performed). The indicator API will return "1" to indicate the usage of an Approved service. Indicators for services providing non-Approved security functions (as well as for services not requiring an indicator) will have a value other than "1", ensuring that the indicators for Approved services are unambiguous.

4.4 Non-Approved Services

The table below lists the non-Approved services available to module operators.

Name	Description	Algorithms	Role
Perform data encryption (non-compliant)	Perform symmetric data encryption	ARIA Blake2 Blowfish Camellia CAST CAST5 ChaCha20 DES IDEA RC2 RC4 RC5 SEED SM4 TDES (non-compliant)	User
Perform data decryption (non-compliant)	Perform symmetric data decryption	ARIA Blake2 Blowfish Camellia CAST CAST5 ChaCha20 DES IDEA RC2 RC4 RC5 SEED SM4 TDES (non-compliant)	User
Perform MAC operations (non-compliant)	Perform message authentication operations	Poly1305 TDES (non-compliant)	User
Perform hash operation (non-compliant)	Perform hash operation	MD2 MD4 MD5 RIPEMD RMD160 SHA-1 (non-compliant) SM3 Whirlpool	User
Perform digital signature functions (non-compliant)	Perform digital signature functions	DSA SigGen DSA SigVer (non-compliant) ECDSA SigGen (non-compliant) ECDSA SigVer (non-compliant) EdDSA SigGen EdDSA SigVer RSA SigGen (non-compliant) RSA SigVer (non-compliant) SM2	User
Perform asymmetric encryption	Perform asymmetric encryption	RSA encrypt/decrypt	User
Perform asymmetric decryption	Perform asymmetric decryption functions	RSA encrypt/decrypt	User

Name	Description	Algorithms	Role
Perform key derivation functions (non-compliant)	Perform key derivation functions	TLS v1.0/v1.1 KDF HKDF KBKDF	User
Perform authenticated encryption/decryption (non-compliant)	Perform authenticated encryption/decryption	AES-GCM (non-compliant) AES-OCB	User
Perform random number generation (non-compliant)	Perform random number generation	ANSI X9.31 RNG (with 128-bit AES core) Hash DRBG HMAC DRBG	User
Perform key pair generation (non-compliant)	Perform key pair generation	DH (non-compliant) DSA KeyGen (non-compliant) ECDH (non-compliant) ECDSA KeyGen (non-compliant) EdDSA KeyGen RSA KeyGen (non-compliant)	User
Shared secret computation (non-compliant)	Perform shared secret computation	DH (non-compliant) ECDH (non-compliant)	User

Table 12: Non-Approved Services

4.5 External Software/Firmware Loaded

The module does not load any software or firmware from external sources.

5. Software/Firmware Security

5.1 Integrity Techniques

All software components within the cryptographic boundary are verified using an Approved integrity technique implemented within the cryptographic module itself. The module implements independent HMAC SHA2-256 digest checks to test the integrity of each library file; failure of the integrity test for either library file will cause the module to enter a critical error state.

The module's integrity check is performed automatically at module instantiation (i.e., when the module is loaded into memory for execution) without action from the module operator.

5.2 Initiate on Demand

The CO can initiate the pre-operational self-tests on demand by re-instantiating the module, rebooting/power-cycling the host device, or issuing the `FIPS_selftest()` API command.

6. Operational Environment

6.1 Operational Environment Type and Requirements

CorSSL comprises a software cryptographic library that executes in a Modifiable operational environment.

The cryptographic module has control over its own SSPs. The process and memory management functionality of the host device's OS prevents unauthorized access to plaintext private and secret keys, intermediate key generation values and other SSPs by external processes during module execution. The module only allows access to SSPs through its well-defined API. The operational environment provides the capability to separate individual application processes from each other by preventing uncontrolled access to CSPs and uncontrolled modifications of SSPs regardless of whether this data is in the process memory or stored on persistent storage within the operational environment. Processes that are spawned by the module are owned by the module and are not owned by external processes/operators.

Please refer to section 2.1 of this document for a list/description of the applicable operational environments.

7. Physical Security

This section is not applicable. Per section 7.7.1 of *ISO/IEC 19790:2012*, the requirements of this section are "applicable to hardware and firmware modules, and hardware and firmware components of hybrid modules".

8. Non-Invasive Security

This section is not applicable. There are currently no approved non-invasive mitigation techniques references in Annex F of *ISO/IEC 19790*.

9. Sensitive Security Parameters Management

9.1 Storage Areas

As a software cryptographic module, there is no mechanism within the module boundary for the persistent storage of SSPs. SSPs are stored in volatile RAM during module operation. The table below lists the storage areas used by the module.

Storage Area Name	Description	Persistence Type
RAM	SSPs stored in RAM.	Dynamic

Table 13: Storage Areas

The module stores DRBG state values for the lifetime of the DRBG instance. The module uses SSPs passed in on the stack by the calling application and does not store these SSPs beyond the lifetime of the API call.

9.2 SSP Input-Output Methods

The table below lists SSP input and output methods for this module. Section 9.4 below selects from the input and output methods listed and specifies the appropriate parameter in the “Inputs/Outputs” column if applicable to a specific SSP.

Name	From	To	Format Type	Distribution Type	Entry Type	SFI or Algorithm
Imported in plaintext via API parameter	External	RAM	Plaintext	Manual	Electronic	
Exported in plaintext via API parameter	RAM	External	Plaintext	Manual	Electronic	

Table 14: SSP Input-Output Methods

9.3 SSP Zeroization Methods

The table below lists SSP zeroization methods for this module. Section 9.4 below selects from the zeroization methods listed and specifies the appropriate parameter in the “Zeroization” column if applicable to a specific SSP.

Zeroization Method	Description	Rationale	Operator Initiation
OpenSSL_cleanse()	The OpenSSL_cleanse() function zeroizes SSPs.	The OpenSSL_cleanse() function zeroizes SSPs, yielding them irretrievable.	The operator calls the OpenSSL_cleanse() function. The successful completion of the procedural zeroization suffices as the implicit indicator that zeroization has completed.
Reboot/Power cycle	The host device is rebooted or power cycled, which zeroizes the SSPs.	Rebooting or power cycling the host device clears RAM, yielding the SSPs irretrievable.	The operator reboots or power cycles the host device.

Table 15: SSP Zeroization Methods

Maintenance, including protection and zeroization, of any keys and CSPs that exist outside the module's cryptographic boundary are the responsibility of the end-user.

9.4 SSPs

The module supports the keys and other SSPs listed in the table below. Note that all SSP imports and exports are electronic and performed within the TOEPP.

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
AES key	Used for symmetric encryption and decryption.	Between 128 and 256 bits - Between 128 and 256 bits	Symmetric Key - CSP	PBKDF TLSv1.2-KDF TLSv1.3-KDF		AES for Symmetric Encryption AES for Symmetric Decryption AES for Authenticated Symmetric Encryption AES for Authenticated Symmetric Decryption
AES CCM key	Used for authenticated symmetric encryption and decryption.	Between 128 and 256 bits - Between 128 and 256 bits	Symmetric Key - CSP	PBKDF TLSv1.2-KDF TLSv1.3-KDF		AES for Authenticated Symmetric Encryption AES for Authenticated Symmetric Decryption
AES GCM key	Used for authenticated symmetric encryption and decryption.	Between 128 and 256 bits - Between 128 and 256 bits	Symmetric Key - CSP	PBKDF TLSv1.2-KDF TLSv1.3-KDF		AES for Authenticated Symmetric Encryption AES for Authenticated Symmetric Decryption
AES XTS key	Used for symmetric encryption and decryption.	256 or 512 bits - 128 or 256 bits	Symmetric Key - CSP	PBKDF		AES-XTS for Symmetric Encryption AES-XTS for Symmetric Decryption
AES CMAC key	Used for MAC generation and verification.	Between 128 and 256 bits - Between 128 and 256 bits	Authentication - CSP	PBKDF TLSv1.2-KDF TLSv1.3-KDF		AES-CMAC/GMAC for Message Authentication
AES GMAC key	Used for MAC generation and verification.	Between 128 and 256 bits - Between 128 and 256 bits	Authentication - CSP	PBKDF		AES-CMAC/GMAC for Message Authentication
TDES key	Used for symmetric decryption. For legacy usage only.	192 bits - 112 bits	Symmetric Key - CSP	PBKDF		TDES for Symmetric Decryption (legacy)

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
TDES CMAC key	Used for MAC verification. For legacy usage only.	192 bits - 112 bits	Authentication - CSP	PBKDF		TDES-CMAC for Message Authentication (legacy)
HMAC key	Used for keyed hash operations.	Between 160 and 512 bits - Between 160 and 512 bits	Authentication - CSP	PBKDF TLSv1.2-KDF TLSv1.3-KDF		HMAC for Message Authentication
DSA public key	Used for verifying digital signatures. For legacy use only.	Between 1024 and 3072 bits - Between 80 and 128 bits	Public - PSP			DSA for Signature Verification (legacy)
ECDSA private key	Used for generating digital signatures.	Between 224 and 521 bits - Between 112 and 256 bits	Private - CSP	ECDSA for Key Generation		ECDSA for Signature Generation
ECDSA public key	Used for verifying digital signatures.	Between 224 and 521 bits - Between 112 and 256 bits	Public - PSP	ECDSA for Key Generation		ECDSA for Signature Verification ECDSA for Signature Verification (legacy) ECDSA for Key Verification ECDSA for Key Verification (legacy)
RSA private key	Used for generating digital signatures and un-encapsulating keys.	Between 2048 and 4096 bits - Between 112 and 150 bits	Private - CSP	RSA for Key Generation		RSA for Signature Generation
RSA public key	Used for verifying digital signatures and encapsulation keys.	Between 1024 and 4096 bits - Between 80 and 150 bits	Public - PSP	RSA for Key Generation		RSA for Signature Verification RSA for Signature Verification (legacy)
DH private component	Used for DH shared secret computation.	Between 224 and 256 bits - Between 112 and 128 bits	Private - CSP	DSA KeyGen (FIPS186-4) (A7573)		DH Shared Secret Computation
DH public component	Used for DH shared secret computation.	2048 bits - 112 bits	Public - PSP	DSA KeyGen (FIPS186-4) (A7573)		DH Shared Secret Computation
DH shared secret	Shared secret computed from the DH private component and the DH public component.	2048 bits - 112 bits	Shared Secret - CSP		DH Shared Secret Computation	TLSv1.2-KDF TLSv1.3-KDF
ECDH private component	Used for ECDH shared secret computation.	Between 224 and 521 bits - Between 112 and 256 bits	Private - CSP	ECDSA for Key Generation		ECDH Shared Secret Computation

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
ECDH public component	Used for ECDH shared secret computation.	Between 224 and 521 bits - Between 112 and 256 bits	Public - PSP	ECDSA for Key Generation		ECDH Shared Secret Computation ECDSA for Key Verification
ECDH shared secret	Shared secret computed from the ECDH private component and ECDH public component.	Between 224 and 521 bits - Between 112 and 256 bits	Shared Secret - CSP		ECDH Shared Secret Computation	TLSv1.2-KDF TLSv1.3-KDF
TLS pre-master secret	Used for the derivation of the TLS master secret.	[DH] Between [ECDH] Between 224 and 521 bits - [DH] Between [ECDH] Between 112 and 256 bits	Shared Secret - CSP		ECDH Shared Secret Computation	TLSv1.2-KDF TLSv1.3-KDF
TLS master secret	Used for the derivation of the AES key and HMAC key for TLS sessions.	384 bits - 384 bits	Shared Secret - CSP	TLSv1.2-KDF TLSv1.3-KDF		TLSv1.2-KDF TLSv1.3-KDF
Passphrase	Used as input for PBKDF2 for deriving keys.	Between 64 and 1024 bits - Between 64 and 1024 bits	Passphrase - CSP			PBKDF
AES GCM IV	Used as the initialization vector with the AES GCM key.	Between 96 and 1024 bits - Between 96 and 1024 bits	Initialization Vector - PSP			AES for Authenticated Symmetric Encryption AES for Authenticated Symmetric Decryption
DRBG entropy input	Entropy material for DRBG.	Between 128 and 512 bits - Between 128 and 512 bits	Entropy Input - CSP			DRBG
DRBG seed	Seeding material for DRBG.	Between 256 and 384 bits - Between 256 and 384 bits	Seed - CSP	DRBG		DRBG
DRBG 'V' value	State values for DRBG.	128 bits - 128 bits	State Value - CSP	DRBG		DRBG
DRBG 'Key' value	State values for DRBG.	Between 128 and 256 bits - Between 128 and 256 bits	State Value - CSP	DRBG		DRBG

Table 16: SSP Table 1

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
AES key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	Passphrase:Derived From TLS master secret:Derived From
AES CCM key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	Passphrase:Derived From TLS master secret:Derived From
AES GCM key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	AES GCM IV:Used With Passphrase:Derived From TLS master secret:Derived From
AES XTS key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	Passphrase:Derived From
AES CMAC key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	Passphrase:Derived From TLS master secret:Derived From
AES GMAC key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	Passphrase:Derived From
TDES key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	Passphrase:Derived From
TDES CMAC key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	Passphrase:Derived From
HMAC key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	Passphrase:Derived From TLS master secret:Derived From
DSA public key	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	
ECDSA private key	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	ECDSA private key:Paired With
ECDSA public key	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	ECDSA public key:Paired With
RSA private key	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	RSA public key:Paired With

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
RSA public key	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	RSA private key:Paired With
DH private component	Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	DH public component:Paired With
DH public component	Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	DH private component:Paired With
DH shared secret	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	DH public component:Derived From DH private component:Derived From
ECDH private component	Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	ECDH public component:Paired With
ECDH public component	Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	ECDH private component:Paired With
ECDH shared secret	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	ECDH public component:Derived From ECDH private component:Derived From
TLS pre-master secret		RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	ECDH public component:Derived From ECDH private component:Derived From
TLS master secret		RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	TLS pre-master secret:Derived From
Passphrase	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	
AES GCM IV	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	AES GCM key:Used With
DRBG entropy input	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	
DRBG seed	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	DRBG entropy input:Derived From
DRBG 'V' value	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	DRBG seed:Derived From
DRBG 'Key' value	Imported in plaintext via API parameter	RAM:Plaintext	Stored in RAM until the zeroize service is called or host device is rebooted/power cycled.	OpenSSL_cleanse() Reboot/Power cycle	DRBG seed:Derived From

Table 17: SSP Table 2

9.5 Transitions

The following list specifies applicable transition periods or timeframes where an algorithm or key length transitions from Approved to non-Approved:

- RSA SigVer: The module offers support for 1024-bit RSA SigVer compliant with *FIPS PUB 186-2* and *FIPS PUB 186-4*. These publications have been superseded by *FIPS PUB 186-5*, and this implementation is now allowed for legacy use only.

- SHA-1: The module offers support for SHA-1 for hashing. This implementation will be non-Approved for all uses starting January 1, 2031.

- TDES: The module includes an implementation of TDES. This implementation supports the use of TDES encryption and TDES-CMAC verification in the Approved mode. Per FIPS 140-3 I.G C.M, these uses are for legacy purposes only. All other TDES functionality is non-Approved.

10. Self-Tests

The module performs pre-operational self-tests and conditional self-tests. Pre-operational tests are performed between the time the cryptographic module is instantiated and before the module transitions to the operational state. Conditional self-tests are performed by the module during module operation when certain conditions exist. The following sections list the self-tests performed by the module, their expected error status, and the error resolutions.

10.1 Pre-Operational Self-Tests

The module performs the pre-operational self-tests listed in the table below.

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details
HMAC-SHA2-256 #1	SHA2-256	Software Integrity Test	SW/FW Integrity	Internal flag; subsequent requests return failure indicator	Software integrity test for libcrypto. Performed automatically without operator action
HMAC-SHA2-256 #2	SHA2-256	Software Integrity Test	SW/FW Integrity	Internal flag; subsequent requests return failure indicator	Software integrity test for libssl. Performed automatically without operator action

Table 18: Pre-Operational Self-Tests

10.2 Conditional Self-Tests

The module performs the conditional self-tests listed in the table below.

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
AES-ECB #1	128-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Encrypt	After successfully completing the software integrity tests.
AES-ECB #2	128-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successfully completing the software integrity tests.
AES-XTS Testing Revision 2.0 #1	256-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Encrypt	After successfully completing the software integrity tests.
AES-XTS Testing Revision 2.0 #2	256-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successfully completing the software integrity tests.

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
AES-XTS Testing Revision 2.0 #3	512-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Encrypt	After successfully completing the software integrity tests.
AES-XTS Testing Revision 2.0 #4	512-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successfully completing the software integrity tests.
AES-CCM #1	192-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Encrypt	After successfully completing the software integrity tests.
AES-CCM #2	192-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successfully completing the software integrity tests.
AES-GCM #1	128-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Encrypt	After successfully completing the software integrity tests.
AES-GCM #2	128-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successfully completing the software integrity tests.
AES-CMAC #1	128-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Generate	After successfully completing the software integrity tests.
AES-CMAC #2	128-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Verify	After successfully completing the software integrity tests.
AES-CMAC #3	192-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Generate	After successfully completing the software integrity tests.
AES-CMAC #4	192-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Verify	After successfully completing the software integrity tests.
AES-CMAC #5	256-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Generate	After successfully completing the software integrity tests.

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
AES-CMAC #6	256-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Verify	After successfully completing the software integrity tests.
TDES-ECB	3-Key	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successfully completing the software integrity tests.
TDES-CMAC	3-Key	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Verify	After successfully completing the software integrity tests.
Counter DRBG #1	AES-128-CTR, with derivation function	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Generate/Instantiate/Reseed	After successfully completing the software integrity tests.
Counter DRBG #2	AES-192-CTR, with derivation function	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Generate/Instantiate/Reseed	After successfully completing the software integrity tests.
Counter DRBG #3	AES-256-CTR, with derivation function	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Generate/Instantiate/Reseed	After successfully completing the software integrity tests.
DSA SigVer (FIPS186-4)	2048-bit; SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Verify	After successfully completing the software integrity tests.
ECDSA SigGen (FIPS186-5) #1	P-224; SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Sign	After successfully completing the software integrity tests.
ECDSA SigVer (FIPS186-5) #1	P-224; SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Verify	After successfully completing the software integrity tests.
ECDSA SigGen (FIPS186-5) #2	K-233; SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Sign	After successfully completing the software integrity tests.
ECDSA SigVer (FIPS186-5) #2	K-233; SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Verify	After successfully completing the software integrity tests.

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
RSA SigGen (FIPS186-5)	2048-bit; SHA2-256; PKCS#1.5 scheme	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Sign	After successfully completing the software integrity tests.
RSA SigVer (FIPS186-5)	2048-bit; SHA2-256; PKCS#1.5 scheme	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Verify	After successfully completing the software integrity tests.
HMAC-SHA-1	SHA-1	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hashed Message	Before completing the software integrity tests.
HMAC-SHA2-224	SHA2-224	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hashed Message	Before completing the software integrity tests.
HMAC-SHA2-256	SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hashed Message	Before completing the software integrity tests.
HMAC-SHA2-384	SHA2-384	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hashed Message	Before completing the software integrity tests.
HMAC-SHA2-512	SHA2-512	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hashed Message	Before completing the software integrity tests.
SHA-1	SHA-1	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hash	Before completing the software integrity tests.
SHA2-224	SHA2-224	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hash	Before completing the software integrity tests.
SHA2-256	SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hash	Before completing the software integrity tests.
SHA2-384	SHA2-384	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hash	Before completing the software integrity tests.

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
SHA2-512	SHA2-512	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hash	Before completing the software integrity tests.
SHA3-256	SHA3-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Hash	Before completing the software integrity tests.
KAS-ECC-SSC Sp800-56Ar3	P-224	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Shared Secret Computation	After successfully completing the software integrity tests.
KAS-FFC-SSC Sp800-56Ar3	2048-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Shared Secret Computation	After successfully completing the software integrity tests.
PBKDF	SHA2-256; 5 iterations	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Derive	After successfully completing the software integrity tests.
TLS v1.2 KDF RFC7627	SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Derive	After successfully completing the software integrity tests.
TLS v1.3 KDF	SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Derive	After successfully completing the software integrity tests.
DSA KeyGen (FIPS186-4)	-	PCT	PCT	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Sign/Verify	When the requested service requires the generation of a DH key pair.
ECDSA KeyGen (FIPS186-5)	-	PCT	PCT	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Sign/Verify	When the requested service requires the generation of an ECDH/ECDSA key pair.
RSA KeyGen (FIPS186-5)	-	PCT	PCT	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Sign/Verify	When the requested service requires the generation of an RSA key pair.

Table 19: Conditional Self-Tests

10.3 Periodic Self-Test Information

The tables below specify the self-tests that are performed when periodic self-tests are initiated by the module operator.

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
HMAC-SHA2-256 #1	Software Integrity Test	SW/FW Integrity	On Demand	Manually
HMAC-SHA2-256 #2	Software Integrity Test	SW/FW Integrity	On Demand	Manually

Table 20: Pre-Operational Periodic Information

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
AES-ECB #1	KAT	CAST	On Demand	Manually
AES-ECB #2	KAT	CAST	On Demand	Manually
AES-XTS Testing Revision 2.0 #1	KAT	CAST	On Demand	Manually
AES-XTS Testing Revision 2.0 #2	KAT	CAST	On Demand	Manually
AES-XTS Testing Revision 2.0 #3	KAT	CAST	On Demand	Manually
AES-XTS Testing Revision 2.0 #4	KAT	CAST	On Demand	Manually
AES-CCM #1	KAT	CAST	On Demand	Manually
AES-CCM #2	KAT	CAST	On Demand	Manually
AES-GCM #1	KAT	CAST	On Demand	Manually
AES-GCM #2	KAT	CAST	On Demand	Manually
AES-CMAC #1	KAT	CAST	On Demand	Manually
AES-CMAC #2	KAT	CAST	On Demand	Manually
AES-CMAC #3	KAT	CAST	On Demand	Manually
AES-CMAC #4	KAT	CAST	On Demand	Manually
AES-CMAC #5	KAT	CAST	On Demand	Manually
AES-CMAC #6	KAT	CAST	On Demand	Manually
TDES-ECB	KAT	CAST	On Demand	Manually
TDES-CMAC	KAT	CAST	On Demand	Manually
Counter DRBG #1	KAT	CAST	On Demand	Manually
Counter DRBG #2	KAT	CAST	On Demand	Manually
Counter DRBG #3	KAT	CAST	On Demand	Manually
DSA SigVer (FIPS186-4)	KAT	CAST	On Demand	Manually
ECDSA SigGen (FIPS186-5) #1	KAT	CAST	On Demand	Manually
ECDSA SigVer (FIPS186-5) #1	KAT	CAST	On Demand	Manually
ECDSA SigGen (FIPS186-5) #2	KAT	CAST	On Demand	Manually
ECDSA SigVer (FIPS186-5) #2	KAT	CAST	On Demand	Manually
RSA SigGen (FIPS186-5)	KAT	CAST	On Demand	Manually
RSA SigVer (FIPS186-5)	KAT	CAST	On Demand	Manually
HMAC-SHA-1	KAT	CAST	On Demand	Manually
HMAC-SHA2-224	KAT	CAST	On Demand	Manually
HMAC-SHA2-256	KAT	CAST	On Demand	Manually
HMAC-SHA2-384	KAT	CAST	On Demand	Manually
HMAC-SHA2-512	KAT	CAST	On Demand	Manually
SHA-1	KAT	CAST	On Demand	Manually
SHA2-224	KAT	CAST	On Demand	Manually
SHA2-256	KAT	CAST	On Demand	Manually

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
SHA2-384	KAT	CAST	On Demand	Manually
SHA2-512	KAT	CAST	On Demand	Manually
SHA3-256	KAT	CAST	On Demand	Manually
KAS-ECC-SSC Sp800-56Ar3	KAT	CAST	On Demand	Manually
KAS-FFC-SSC Sp800-56Ar3	KAT	CAST	On Demand	Manually
PBKDF	KAT	CAST	On Demand	Manually
TLS v1.2 KDF RFC7627	KAT	CAST	On Demand	Manually
TLS v1.3 KDF	KAT	CAST	On Demand	Manually
DSA KeyGen (FIPS186-4)	PCT	PCT	On Demand	Manually
ECDSA KeyGen (FIPS186-5)	PCT	PCT	On Demand	Manually
RSA KeyGen (FIPS186-5)	PCT	PCT	On Demand	Manually

Table 21: Conditional Periodic Information

10.4 Error States

The table below describes the module's error states, error status indicators, and recovery methods.

Name	Description	Conditions	Recovery Method	Indicator
Critical Error	Upon test failure, the module immediately terminates the calling application's API call with a returned error code and sets an internal flag, signaling the error condition. For any subsequent request made by the calling application for cryptographic services, the module will return a failure indicator, thereby disabling all access to its cryptographic functions, sensitive security parameters (SSPs), and data output services while the error condition persists.	If any of the pre-operational integrity tests fail. If any of the conditional self-tests fail.	The module must be re-instantiated by the calling application. The Crypto Officer should contact Sunhillo Corporation if errors persist after re-instantiation.	Returns error code and sets an internal flag. Subsequent requests return failure indicator.

Table 22: Error States

If the module continues to experience self-test failures after reinitializing, then the module will not be able to resume normal operations, and the CO should contact Corsec Security, Inc. for assistance.

10.5 Operator Initiation of Self-Tests

The CO can initiate the pre-operational self-tests and conditional CASTs on demand for periodic testing of the module by re-instantiating the module, rebooting/power-cycling the host device, or issuing the `FIPS_selftest()` API command.

11. Life-Cycle Assurance

11.1 Installation, Initialization, and Startup Procedures

The module is distributed as a package containing the binaries and HMAC digest files that the Crypto Officer is to install onto a target platform specified in section 2.1 or one where portability is maintained.

This module is designed to support third-party vendor applications, and these applications are the sole consumers of the cryptographic services provided by the module. No end-user action is required to initialize the module for operation; the calling application performs any actions required to initialize the module.

11.2 Administrator Guidance

There are no specific management activities required of the CO role to ensure that the module runs securely. If any irregular activity is observed, or if the module is consistently reporting errors, then Corsec Customer Support should be contacted.

The following subsections provide additional guidance for approved services.

11.2.1 Show Status

The `fips_post_status()` API can be used to determine the module's operational status. A non-zero return value indicates that the module has passed all pre-operational self-tests and is currently in its Approved mode.

11.2.2 Show Versioning Information

The `OpenSSL_version()` API can be used to obtain the module's versioning information. The API call will return "CorSSL v1.1.1zd.001", which correlates to the "CorSSL 1.1.1zd.001" on the module's FIPS 140-3 validation certificate.

The "Certificate Management" Approved Service invokes multiple sub-services for certificate management. The calling application is responsible for calling the service indicators used in the negotiated cipher suite to verify each sub-service uses Approved algorithms.

For example, when generating an ECDSA based certificate, the calling application must invoke:

- `EC_key_get_service_indicator()` for the signature verification of the server certificate.
- `EVP_Digest_get_service_indicator()` for the hash of the signature.
- `EVP_cipher_get_service_indicator()` for the cipher algorithm used for the certificate.

11.3 Non-Administrator Guidance

The following list provides additional policies for the User role:

- The cryptographic module's services are designed to be provided to a calling application. Excluding the use of the NIST-defined elliptic curves as trusted third-party domain parameters, all other assurances from *FIPS PUB 186-5* (including those required of the intended signatory and the signature verifier) are outside the scope of the module and are the responsibility of the calling application.

- The calling application is responsible for ensuring that CSPs are not shared between Approved and non-Approved services and modes of operation.

11.3.1 Establish TLS Connection

The "Establish TLS connection" Approved Service invokes multiple sub-services for TLS connection establishment. The calling application is responsible for calling the service indicators used in the negotiated cipher suite to verify each sub-service uses Approved algorithms.

For example, the client and server agree upon the following cipher suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256. During the TLS handshake and session, the calling application must invoke:

- `EC_key_get_service_indicator()` for the shared secret computation component of the key exchange algorithm.
- `TLSKDF_get_service_indicator()` for the key derivation function used to derive the TLS session and authentication keys.
- `EC_key_get_service_indicator()` for the signature verification of the server certificate.
- `EVP_cipher_get_service_indicator()` for the cipher algorithm used during the TLS session for encryption and decryption.
- `HMAC_get_service_indicator()` for the HMAC algorithm used during the TLS session for message authentication (unless using an AEAD, like AES-CCM or AES-GCM).

The following cipher suites are Approved for the "Establish TLS connection" service:

TLS v1.2

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8

TLS v1.3

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_AES_128_CCM_SHA256
- TLS_AES_128_CCM_8_SHA256

Please refer to Appendix B for the detailed information about approved service indicator usage.

12. Mitigation of Other Attacks

This section is not applicable. The module does not claim to mitigate any attacks beyond the FIPS 140-3 Level 1 requirements for this validation.

Appendix A. Acronyms and Abbreviations

Table 23 provides definitions for the acronyms and abbreviations used in this document.

Table 23: Acronyms and Abbreviations

Acronym	Definition
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard – New Instructions
API	Application Programming Interface
CBC	Cipher Block Chaining
CCCS	Canadian Centre for Cyber Security
CMVP	Cryptographic Module Validation Program
CO	Cryptographic Officer
CPU	Central Processing Unit
CSP	Critical Security Parameter
CTR	Counter
CVL	Component Validation List
DEP	Default Entry Point
DES	Data Encryption Standard
DH	Diffie-Hellman
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GMAC	Galois Message Authentication Code
GPC	General-Purpose Computer
HMAC	(keyed-) Hash Message Authentication Code
KAS	Key Agreement Scheme
KAT	Known Answer Test
KTS	Key Transport Scheme
KW	Key Wrap
KWP	Key Wrap with Padding

Acronym	Definition
NIST	National Institute of Standards and Technology
OS	Operating System
PBKDF	Password-Based Key Derivation Function
PCT	Pairwise Consistency Test
PKCS	Public Key Cryptography Standard
PSS	Probabilistic Signature Scheme
RNG	Random Number Generator
RSA	Rivest, Shamir, and Adleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SP	Special Publication
TLS	Transport Layer Security
TOEPP	Tested Operational Environment's Physical Perimeter
XEX	XOR Encrypt XOR
XTS	XEX-Based Tweaked-Codebook Mode with Ciphertext Stealing

Appendix B. Approved Service Indicators

This appendix specifies the APIs that are externally accessible and return the Approved service indicators.

Synopsis

```
#include <openssl/service_indicator.h>
#include <openssl/ssl.h>

int EVP_cipher_get_service_indicator(EVP_CIPHER_CTX *ctx);
int DSA_get_service_indicator(DSA * ptr_dsa, DSA_MODES_t mode);
int RSA_key_get_service_indicator(RSA * ptr_rsa);
int PBKDF_get_service_indicator();
int EVP_Digest_get_service_indicator(EVP_MD_CTX *ctx);
int EC_key_get_service_indicator(EC_KEY *ec_key);
int CMAC_get_service_indicator(CMAC_CTX *cmac_ctx, CMAC_MODE_t mode);
int HMAC_get_service_indicator(HMAC_CTX *ctx);
int TLSKDF_get_service_indicator(EVP_PKEY_CTX *tls_ctx);
int TLS1_3_kdf_get_service_indicator(EVP_MD *md);
int TLS1_3_get_service_indicator(SSL *s);
int DRBG_get_service_indicator(RAND_DRBG *drbg);
```

Description

These APIs are high-level interfaces that return the Approved service indicator value based on the parameter(s) passed to them.

- EVP_cipher_get_service_indicator() is used to return the appropriate Approved service indicator status for block ciphers like AES and Triple DES.
- DSA_get_service_indicator() is used to return the appropriate Approved service indicator status for the DSA algorithm and its modes. You must include the mode you want the indicator for, which are specified in the DSA_MODES_t enum.
- RSA_key_get_service_indicator() is used to return the appropriate Approved service indicator status for RSA algorithm and its modes.
- PBKDF_get_service_indicator() is used to return the appropriate Approved service indicator status for PBKDF usage.
- EVP_Digest_get_service_indicator() is used to return the appropriate Approved service indicator status for SHS algorithms like SHA-1 and SHAKE.
- EC_key_get_service_indicator() is used to return the appropriate Approved service indicator status for elliptic curve algorithms like ECDSA and its modes.

- CMAC_get_service_indicator() is used to return the appropriate Approved service indicator status for CMAC requests that use AES or Triple DES. You must include the mode you want the indicator for, which are specified in the CMAC_MODE_t enum.

- HMAC_get_service_indicator() is used to return the appropriate Approved service indicator status for HMAC requests and the associated SHS algorithm.

- TLSKDF_get_service_indicator() is used to return the appropriate Approved service indicator status for TLS KDF usage excluding TLS 1.3.

- TLS1_3_kdf_get_service_indicator() is used to return the appropriate Approved service indicator status for TLS 1.3 KDF usage. This function requires the ssl.h file and is used to call the TLS1_3_get_service_indicator() function because of the SSL struct requirement. You cannot call TLS1_3_get_service_indicator() directly unless you have the SSL struct that was used.

- DRBG_get_service_indicator() is used to return the appropriate Approved service indicator status for DRBG usage.

- To confirm the approved usage of KAS-FFC-SSC key agreement, the calling application must check:

    - DSA_get_service_indicator(dsa, DSA_KEY_GEN) value greater than 0,
    - DSA_get_service_indicator(dsa, DSA_PARAM_GEN) value greater than 0, and
    - DH_compute_key() value of 224 - 256
    - Additionally, the operator must ensure that the DH key used in DH_compute_key was constructed from the same DSA key invoked in the DSA indicator.

  Since the final action occurs outside the module control or visibility, IG 2.4.C is applicable to determining the indicator for the KAS-FFC-SSC service.

Prepared by:
Corsec Security, Inc.

12600 Fair Lakes Circle, Suite 210
Fairfax, VA 22033
United States of America

Phone: +1 703 267 6050
Email: info@corsec.com
http://www.corsec.com