Corsec Security, Inc.

CorSSL™

Software Version: 1.1.1s.006

FIPS 140-3 Non-Proprietary Security Policy

FIPS Security Level: 1
Document Version: 0.1

Prepared by:

Corsec Security, Inc.
12600 Fair Lakes Circle, Suite 210
Fairfax, VA 22033
United States of America

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

Abstract

This is a non-proprietary Cryptographic Module Security Policy for CorSSL™ (software version: 1.1.1s.006) 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 Cryptographic Module Validation Program (CMVP) website, which is maintained by the National Institute of Standards and Technology (NIST) and the Canadian Centre for Cyber Security (CCCS).

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 "module".

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 Security, Inc..
- 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.

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.

Table of Contents

List of Tables

List of Figures

1. General

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[1] APL[2].

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 version 1.1.1s.006 is a software library providing a C language API[3] for use by other applications requiring cryptographic functionality. CorSSL 1.1.1s.006 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[4] 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.

CorSSL is validated at the FIPS 140-3 section levels shown in Table 1.

Table 1 – Security Levels

ISO/IEC 24579 Section 6. [Number Below]	FIPS 140-3 Section Title	Security Level
1	General	1
2	Cryptographic Module Specification	1
3	Cryptographic Module Interfaces	1
4	Roles, Services, and Authentication	1
5	Software/Firmware Security	1
6	Operational Environment	1
7	Physical Security	N/A
8	Non-Invasive Security	N/A

[1] DoDIN – Department of Defense Information Network
[2] APL – Approved Product List
[3] API – Application Programming Interface
[4] TLS – Transport Layer Security

ISO/IEC 24579 Section 6. [Number Below]	FIPS 140-3 Section Title	Security Level
9	Sensitive Security Parameter Management	1
10	Self-Tests	1
11	Life-Cycle Assurance	1
12	Mitigation of Other Attacks	N/A

The module has an overall security level of 1.

2. Cryptographic Module Specification

CorSSL is a software module with a multi-chip standalone embodiment. The module is designed to operate within a modifiable operational environment. Additionally, the module is designed to utilize the AES-NI[5] extended instruction set when available by the host platform's CPU for processor algorithm acceleration (PAA) of its AES implementation.

2.1 Operational Environments

The module was tested and found to be compliant with FIPS 140-3 requirements on the operational environments (OE) listed in Table 2.

Table 2 – Tested Operational Environments

#	Operating System	Hardware Platform	Processor	PAA/Acceleration
1	Debian 9	Dell PowerEdge R440	Intel® Xeon Silver 4214R	With (AES-NI)
2	Debian 9	Dell PowerEdge R440	Intel® Xeon Silver 4214R	Without

There are no vendor-affirmed operational environments claimed.

Module operators may perform post-validation porting of the module and affirm the module's continued validation compliance. The cryptographic module will remain compliant with the FIPS 140-3 validation on any general-purpose platform/processor that supports the specified operating system listed on the validation entry, or another compatible operating system. The CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when ported to an operational environment not listed on the validation certificate.

2.2 Algorithm Implementations

Validation certificates for each Approved security function are listed in Table 3. Note that there are algorithms, modes, and key/moduli sizes that have been CAVP-tested but are not used by any Approved service of the module. Only the algorithms, modes/methods, and key lengths/curves/moduli shown in Table 3 are used by an Approved service of the module.

[5] AES-NI – Advanced Encryption Algorithm New Instructions

Table 3 – Approved Algorithms

CAVP Certificate[6]	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A4978	**AES** *FIPS PUB[7] 197* *NIST SP 800-38A*	CBC[8], CFB1[9], CFB8, CFB128, CTR[10], ECB[11], OFB[12]	128, 192, 256	Encryption/decryption
A4978	**AES** *NIST SP 800-38B*	CMAC[13]	128, 192, 256	MAC generation/verification
A4978	**AES** *NIST SP 800-38C*	CCM[14]	128, 192, 256	Encryption/decryption
A4978	**AES** *NIST SP 800-38D*	GCM[15] (internal IV)	128, 192, 256	Encryption/decryption
A4978	**AES** *NIST SP 800-38D*	GMAC[16]	128, 192, 256	MAC eneration/verification
A4978	**AES** *NIST SP 800-38E*	XTS[17,18,19]	128, 256	Encryption/decryption
A4978	**AES** *NIST SP 800-38F*	KW[20], KWP[21]	128, 192, 256	Encryption/decryption
Vendor Affirmed	**CKG**[22] *NIST SP 800-133rev2*	-	-	Cryptographic key generation
A4978	**CVL**[23] *NIST SP 800-135rev1*	KDF (SSH, TLS[24] v1.0/1.1, v1.2)	-	Key derivation *No parts of the SSH or TLS protocols, other than the KDFs, have been tested by the CAVP and CMVP.*
A4978	**CVL** *RFC[25] 7627*	KDF (TLS v1.2)	-	Key derivation *No part of the TLS v1.2 protocol, other than the KDF, has been tested by the CAVP and CMVP.*

[6] This table includes vendor-affirmed algorithms that are approved but CAVP testing is not yet available.
[7] PUB – Publication
[8] CBC – Cipher Block Chaining
[9] CFB – Cipher Feedback
[10] CTR – Counter
[11] ECB – Electronic Code Book
[12] OFB – Output Feedback
[13] CMAC – Cipher-Based Message Authentication Code
[14] CCM – Counter with Cipher Block Chaining - Message Authentication Code
[15] GCM – Galois Counter Mode
[16] GMAC – Galois Message Authentication Code
[17] XOR – Exclusive OR
[18] XEX – XOR Encrypt XOR
[19] XTS – XEX-Based Tweaked-Codebook Mode with Ciphertext Stealing
[20] KW – Key Wrap
[21] KWP – Key Wrap with Padding
[22] CKG – Cryptographic Key Generation
[23] CVL – Component Validation List
[24] TLS – Transport Layer Security
[25] RFC – Request for Comments

CAVP Certificate[6]	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A4979	**CVL** *RFC 8446*	KDF (TLS v1.3)	-	Key derivation *No part of the TLS v1.3 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A4978	**DRBG**[26] *NIST SP 800-90Arev1*	Counter-based	128, 192, 256-bit AES-CTR	Deterministic random bit generation
A4978	**DSA**[27] *FIPS PUB 186-4*	KeyGen	2048/224, 2048/256, 3072/256	Key pair generation
		PQGGen	2048/224, 2048/256, 3072/256 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Domain parameter generation
		PQGVer	2048/224, 2048/256, 3072/256 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Domain parameter verification
		SigGen	2048/224, 2048/256, 3072/256 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
		SigVer	2048/224, 2048/256, 3072/256 (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification
A4978	**ECDSA**[28] *FIPS PUB 186-4*	KeyGen Secret generation mode: Testing candidates	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Key pair generation
		KeyVer	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	Public key validation
		SigGen	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
		SigVer	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 (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification

[26] DRBG – Deterministic Random Bit Generator
[27] DSA – Digital Signature Algorithm
[28] ECDSA – Elliptic Curve Digital Signature Algorithm

CAVP Certificate[6]	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A4978	**HMAC** *FIPS PUB 198-1*	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	112 (minimum)	Message authentication
A4978	**KAS-ECC-SSC**[29] *NIST SP 800-56Arev3*	ephemeralUnified	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Shared secret computation
A4978	**KAS-FFC-SSC**[30] *NIST SP 800-56Arev3*	dhEphem	2048/224 (FB), 2048/256 (FC)	
A4978	**KDA**[31] *NIST SP 800-56Crev2*	HKDF	SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512)	Key derivation
A4978	**KTS**[32] *NIST SP 800-38C*	AES-CCM	128, 192, 256	Key wrap/unwrap (authenticated encryption)[33] *Key establishment methodology provides between 128 and 256 bits of encryption strength*
A4978	**KTS** *NIST SP 800-38D*	AES-GCM	128, 192, 256	Key wrap/unwrap (authenticated encryption)[34] *Key establishment methodology provides between 128 and 256 bits of encryption strength*
A4978	**KTS** *NIST SP 800-38F*	AES-KW, AES-KWP	128, 192, 256	Key wrap/unwrap *Key establishment methodology provides between 128 and 256 bits of encryption strength*
A4978	**KTS** *FIPS PUB 197* *NIST SP 800-38B*	AES-CMAC	128, 192, 256	Key wrap/unwrap (encryption with message authentication)[35] *Key establishment methodology provides between 128 and 256 bits of encryption strength*
A4978	**KTS** *FIPS PUB 197* *FIPS PUB 198-1*	AES-ECB with HMAC	128, 192, 256	Key wrap/unwrap (encryption with message authentication)[36] *Key establishment methodology provides between 128 and 256 bits of encryption strength*

[29] KAS-ECC-SSC – Key Agreement Scheme - Elliptic Curve Cryptography - Shared Secret Computation

[30] KAS-SSC – Key Agreement Scheme - Finite Field Cryptography - Shared Secret Computation

[31] KDA – Key Derivation Algorithm

[32] KTS – Key Transport Scheme

[33] Per FIPS 140-3 Implementation Guidance D.G, AES-CCM is an Approved key transport technique.

[34] Per FIPS 140-3 Implementation Guidance D.G, AES-GCM is an Approved key transport technique.

[35] Per FIPS 140-3 Implementation Guidance D.G, AES with CMAC is an Approved key transport technique.

[36] Per FIPS 140-3 Implementation Guidance D.G, AES (in any Approved mode) with HMAC is an Approved key transport technique.

CAVP Certificate[6]	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A4978	**PBKDF2**[37] *NIST SP 800-132*	Section 5.4, option 1a	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Password-based key derivation
A4978	**RSA**[38] *FIPS PUB 186-4, Appendix B.3.3*	Key generation mode: B.3.3	2048, 3072, 4096	Key pair generation
A4978	**RSA** *FIPS PUB 186-4*	X9.31	2048, 3072, 4096 (SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
			1024, 2048, 3072, 4096 (SHA-1, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification
		PKCS#1 v1.5	2048, 3072, 4096 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
			1024, 2048, 3072, 4096 (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification
		PSS[39]	2048, 3072, 4096 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
			1024, 2048, 3072, 4096 (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification
A4978	**SHA-3** *FIPS PUB 202*	SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE[40]-128, SHAKE-256	-	Message digest
A4978	**SHS**[41] *FIPS PUB 180-4*	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	-	Message digest
A4978	**Triple-DES** *NIST SP 800-67* *NIST SP 800-38A*	CBC, CFB1, CFB8, CFB64, ECB, OFB	168	Decryption
A4978	**Triple-DES** *NIST SP 800-67* *NIST SP 800-38B*	CMAC	112, 168	MAC verification

The vendor affirms the following cryptographic security methods:

- Cryptographic key generation – In compliance with section 6.1 of *NIST SP 800-133rev2*, the module uses its Approved DRBG to generate random bits and seeds used for asymmetric key generation. The generated

[37] PBKDF – Password-based Key Derivation Function
[38] RSA – Rivest Shamir Adleman
[39] PSS – Probabilistic Signature Scheme
[40] SHAKE – Secure Hash Algorithm KECCAK
[41] SHS – Secure Hash Standard

seed is an unmodified output from the DRBG. The cryptographic 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.

The calling application and its entropy sources are located within the operational environment inside the module's physical perimeter but outside the cryptographic boundary. Thus, there is no assurance of the minimum strength of the generated keys.

The module implements the non-Approved but allowed algorithms shown in Table 4 below.

Table 4 – Non-Approved Algorithms Allowed in the Approved Mode of Operation

Algorithm	Caveat	Use / Function
AES	Cert. A4978; key unwrapping; Per IG D.G.	Symmetric key unwrapping
RSA	Cert. A4978; key unencapsulation: Per IG D.G.	Asymmetric key unencapsulation
SHA-1	Cert. A4978; secure hashing	Digital signature generation in TLS v1.0/1.1[42]
Triple-DES	Cert. A4978; key unwrapping; Per IG D.G.	Symmetric key unwrapping

The module does not implement any non-Approved algorithms in the Approved mode of operation with no security claimed.

The module employs the non-Approved algorithms shown in Table 5 below. These algorithms shall not be used in the module's Approved mode of operation.

Table 5 – Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

Algorithm	Use / Function
AES-GCM (non-compliant when used with external IV)	Authenticated encryption/decryption
AES-OCB[43]	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, CAST5	Encryption/decryption
ChaCha20	Encryption/decryption
DES	Encryption/decryption
DH (non-compliant with key sizes below 2048 bits)	Key agreement

[42] Per NIST SP 800-52, SHA-1 is allowed for generating digital signatures on ephemeral parameters within TLS v1.0/1.1.
[43] OCB – Offset Codebook

Algorithm	Use / Function
DSA (non-compliant)	Digital signature generation
ECDSA (non-compliant)	Digital signature generation
RSA (non-compliant when used with SHA-1 outside the TLS protocol)	Digital signature generation
DSA (non-compliant with key sizes below the minimums Approved for Approved mode)	Key pair generation, digital signature verification
ECDH (non-compliant with curves P-192, K-163, B-163, and non-NIST curves)	Key agreement
ECDSA (non-compliant with curves P-192, K-163, B-163, and non-NIST curves)	Key pair generation, digital signature verification
EdDSA[44]	Key pair generation, digital signature generation, digital signature verification
IDEA	Encryption/decryption
MD2, MD4, MD5	Message digest
Poly1305	Message authentication code
RC2[45], RC4, RC5	Encryption/decryption
RIPEMD	Message digest
RMD160	Message digest
RSA (non-compliant with non-approved/untested key sizes, and functions)	Key pair generation; digital signature generation; digital signature verification; key transport
SEED	Encryption/decryption
SM2, SM3	Message digest
SM4	Encryption/decryption
Triple-DES (non-compliant)	Encryption; MAC generation; key wrap
Whirlpool	Message digest

2.3 Cryptographic Boundary

As a software cryptographic module, the module has no physical components. The physical perimeter of the cryptographic module is defined by each host platform on which the module is installed. Figure 1 below illustrates a block diagram of a typical GPC and the module's physical perimeter.

[44] EdDSA – Edwards-curve Digital Signature Algorithm
[45] RC – Rivest Cipher

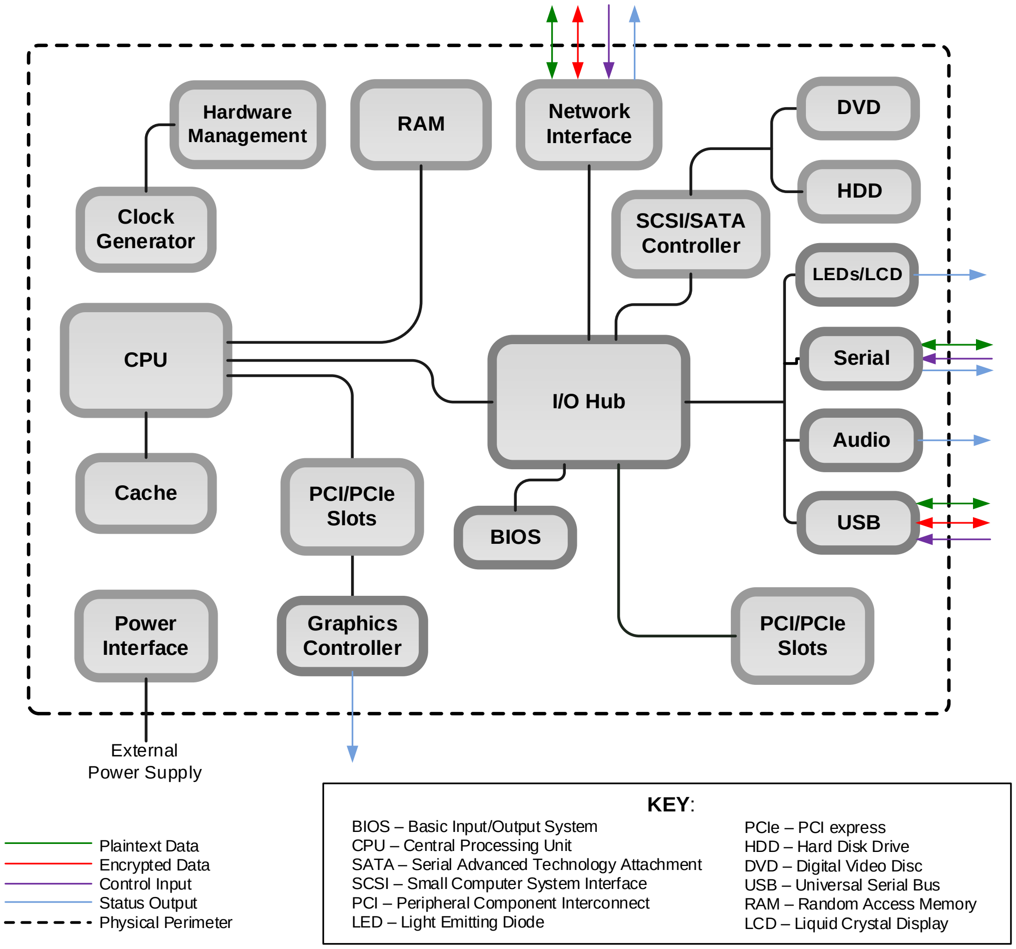

Figure 1 – GPC Block Diagram

The module's cryptographic boundary consists of all functionalities contained within the module's compiled source code and comprises the following components:

- 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)

The cryptographic boundary is the contiguous perimeter that surrounds all memory-mapped functionality provided by the module when loaded and stored in the host device's memory. The module is entirely contained within the physical perimeter.

Figure 2 shows the logical block diagram of the module executing in memory and its interactions with surrounding software components, as well as the module's physical perimeter and cryptographic boundary.

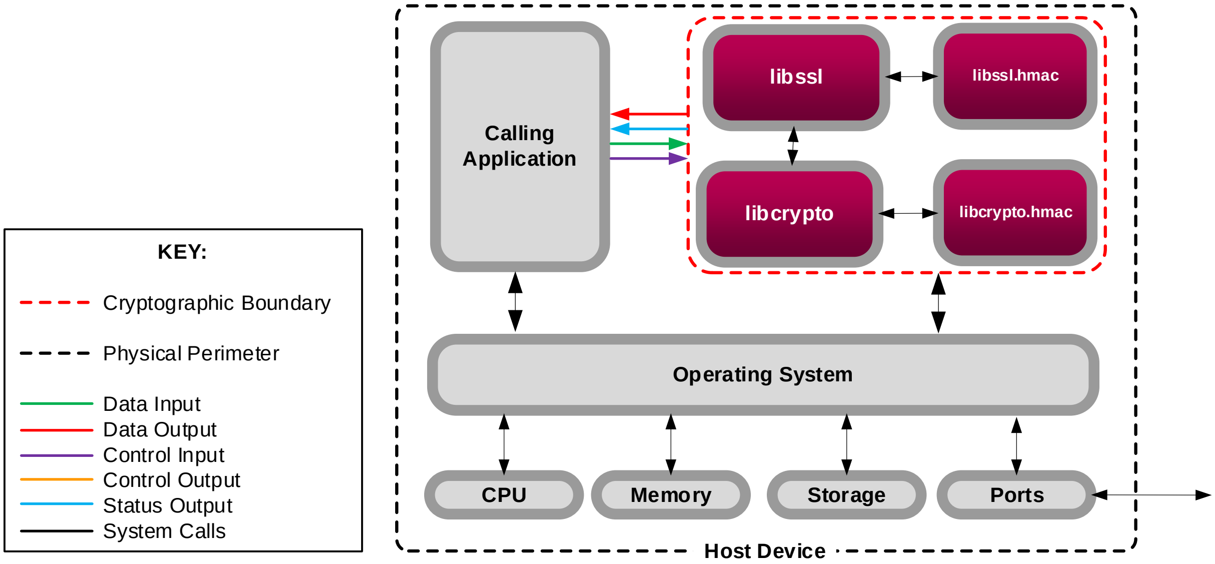

Figure 2 – Module Block Diagram (with Cryptographic Boundary)

2.4 Modes of Operation

The module supports two modes of operation: Approved and non-Approved. The module will be in Approved mode when all pre-operational self-tests have completed successfully, and only Approved services are invoked. Table 3 and Table 4 above list the Approved and allowed algorithms; Table 8 provides descriptions of the Approved services.

The module can also alternate service-by-service between Approved and non-Approved modes of operation. The module will switch to the non-Approved mode upon execution of a non-Approved service. The module will switch back to the Approved mode upon execution of an Approved service. Table 5 lists the non-Approved algorithms implemented by the module; Table 9 below lists the services that constitute the non-Approved mode.

When following the guidance in this document, CSPs are not shared between Approved and non-Approved services and modes of operation.

3. Cryptographic Module Interfaces

FIPS 140-3 defines the following logical interfaces for cryptographic modules:

- Data Input
- Data Output
- Control Input
- Control Output
- 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 interfaces and the module's ports and interfaces at the physical and logical boundaries can be found in Table 6. Note that the module does not output control information, and thus has no specified control output interface.

Table 6 – Ports and Interfaces

Physical Port	Logical Interface	Data That Passes Over Port/Interface
Physical data input port(s) of the tested platforms	Data Input • API input arguments that provide input data for processing	• Data to be encrypted, decrypted, signed, verified, or hashed • Keys to be used in cryptographic services • Random seed material for the module's DRBG • Keying material to be used as input to key establishment services
Physical data output port(s) of the tested platforms	Data Output • API output arguments that return generated or processed data back to the caller	• Data that has been encrypted, decrypted, or verified • Digital signatures • Hashes • Random values generated by the module's DRBG • Keys established using module's key establishment methods
Physical control input port(s) of the tested platforms	Control Input • API input arguments that are used to initialize and control the operation of the module	• API commands invoking cryptographic services • Modes, key sizes, etc. used with cryptographic services
Physical status output port(s) of the tested platforms	Status Output • API call return values	• Status information regarding the module • Status information regarding the invoked service/operation

4. Roles, Services, and Authentication

The sections below describe the module's authorized roles, services, and operator authentication methods.

4.1 Authorized Roles

The module supports a Crypto Officer (CO) that authorized operators can assume. The CO role performs cryptographic initialization or management functions and general security services.

The module also supports the following role(s):

- User – The User role performs general security services, including cryptographic operations and other approved security functions.

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

Table 7 below lists the supported roles, along with the services (including input and output) available to each role.

Table 7 – Roles, Service Commands, Input and Output

Role	Service	Input	Output
CO	Show Status	API call parameters	Current operational status
CO	Perform self-tests on-demand	Re-instantiate module; API call parameters	Status
CO	Zeroize	Restart calling application; reboot or power-cycle host platform	None
CO	Show versioning information	API call parameters	Module name, version
User	Perform symmetric encryption	API call parameters, key, plaintext	Status, ciphertext
User	Perform symmetric decryption	API call parameters, key, ciphertext	Status, plaintext
User	Generate symmetric digest	API call parameters, key, plaintext	Status, digest
User	Verify symmetric digest	API call parameters, digest	Status
User	Perform authenticated symmetric encryption	API call parameters, key, plaintext	Status, ciphertext
User	Perform authenticated symmetric decryption	API call parameters, key, ciphertext	Status, plaintext
User	Generate random number	API call parameters	Status, random bits
User	Perform keyed hash operations	API call parameters, key, message	Status, MAC[46]
User	Perform hash operation	API call parameters, message	Status, hash
User	Generate DSA domain parameters	API call parameters	Status, domain parameters
User	Verify DSA domain parameters	API call parameters	Status, domain parameters
User	Generate asymmetric key pair	API call parameters	Status, key pair
User	Verify ECDSA public key	API call parameters, key	Status
User	Generate digital signature	API call parameters, key, message	Status, signature

[46] MAC – Message Authentication Code

Role	Service	Input	Output
User	Verify digital signature	API call parameters, key, signature, message	Status
User	Perform key wrap	API call parameters, encryption key, key	Status, encrypted key
User	Perform key unwrap	API call parameters, decryption key, encrypted key	Status, decrypted key
User	Compute shared secret	API call parameters	Status, shared secret
User	Derive SSH keys	API call parameters, SSH master secret	Status, SSH keys
User	Derive TLS keys	API call parameters, TLS pre-master secret	Status, TLS keys
User	Derive key via HKDF	API call parameters	Status, key
User	Derive key via PBKDF2	API call parameters, passphrase	Status, key
User	Generate symmetric digest (CMAC)	API call parameters, key, message	Status, MAC

4.2 Authentication Methods

The module does not support authentication mechanisms; roles are implicitly selected based on the service invoked. Refer to Table 7 above for a listing of the services associated with each authorized role.

4.3 Services

Descriptions of the services available to the authorized roles are provided in Table 8 below.

This module is a software library that provides cryptographic functionality to calling applications. As such, the security functions provided via the module’s APIs are considered security services, and the module provides indicators for Approved security services as required by FIPS 140-3 IG 2.4.C.

When invoking an API for an offered security service, the calling application provides inputs (keys, key sizes, modes, etc.) that the module combines into a single, internal structure, or “context”, that drives the execution of the cryptographic service. Each security service invocation will determine if the invoked service is an Approved security service. Upon completion of the service, that context is first updated with the results of the service as well as the Approved security service indicator, and then returned to the calling application.

To access the indicator value from the context, the calling application must pass the resultant context to the indicator API associated with that security function (note the indicator check must be performed before 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. Additional details on the APIs used for the Approved service indicators are provided in Appendix B below.

Please note that the keys and Sensitive Security Parameters (SSPs) listed in the table indicate the type of access required 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.

Table 8 – Approved Services

Service	Description	Approved Security Function(s)	Keys and/or SSPs	Roles	Access Rights to Keys and/or SSPs	Indicator
Show Status	Return mode status	None	None	CO	N/A	N/A
Perform self-tests on-demand	Perform pre-operational self-tests	None	None	CO	N/A	API return value
Zeroize	Zeroize and de-allocate memory containing sensitive data	None	All SSPs	CO	All SSPs – Z	N/A
Show versioning information	Return module versioning information	None	None	CO	N/A	N/A
Perform symmetric encryption	Encrypt plaintext data	AES (CBC, CFB1, CFB8, CFB128, CTR, ECB, OFB, KW, KWP) (Cert. A4978) XTS-AES (Cert. A4978)	AES key XTS-AES key	User	AES key – WE XTS-AES key – WE	API return value
Perform symmetric decryption	Decrypt ciphertext data	AES (CBC, CFB1, CFB8, CFB128, CTR, ECB, OFB, KW, KWP) (Cert. A4978) XTS-AES (Cert. A4978) Triple-DES (CBC, CFB1, CFB8, CFB64, ECB, OFB) (Cert. A4978)	AES key XTS-AES key Triple-DES key	User	AES key – WE XTS-AES key – WE Triple-DES key – WE	API return value
Generate symmetric digest	Generate symmetric digest	AES (CMAC)(Cert. A4978) AES (GMAC) (Cert. A4978)	AES CMAC key AES GMAC key	User	AES CMAC key – WE AES GMAC key – WE	API return value
Verify symmetric digest	Verify symmetric digest	AES (CMAC) (Cert. A4978) AES (GMAC) (Cert. A4978) Triple-DES CMAC (Cert. A4978)	AES CMAC key AES GMAC key Triple-DES CMAC key	User	AES CMAC key – WE AES GMAC key – WE Triple-DES CMAC key – WE	API return value
Perform authenticated symmetric encryption	Encrypt plaintext using supplied AES GCM key and IV	AES (GCM) (Cert. A4978)	AES GCM key AES GCM IV	User	AES GCM key – WE AES GCM IV – WE	API return value
Perform authenticated symmetric decryption	Decrypt ciphertext using supplied AES GCM key and IV	AES (GCM) (Cert. A4978)	AES GCM key AES GCM IV	User	AES GCM key – WE AES GCM IV – WE	API return value
Generate random number	Return random bits to the calling application	DRBG (Cert. A4978)	DRBG entropy input DRBG seed DRBG 'V' value DRBG 'Key' value	User	DRBG entropy input – WE DRBG seed – GE DRBG 'V' value – GE DRBG 'Key' value – GE	API return value
Perform keyed hash operations	Compute a message authentication code	HMAC (Cert. A4978) SHA (Cert. A4978)	HMAC key	User	HMAC key – WE	API return value
Perform hash operation	Compute a message digest	SHA (Cert. A4978)	None	User	N/A	API return value
Generate DSA domain parameters	Generate DSA domain parameters	DSA (Cert. A4978)	None	User	N/A	API return value
Verify DSA domain parameters	Verify DSA domain parameters	DSA (Cert. A4978)	None	User	N/A	API return value
Generate asymmetric key pair	Generate a public/private key pair	DSA (Cert. A4978) ECDSA (Cert. A4978) RSA (Cert. A4978)	DSA public key DSA private key ECDSA public key ECDSA private key RSA public key RSA private key	User	DSA public key – GR DSA private key – GR ECDSA public key – GR ECDSA private key – GR RSA public key – GR RSA private key – GR	API return value

Service	Description	Approved Security Function(s)	Keys and/or SSPs	Roles	Access Rights to Keys and/or SSPs	Indicator
Verify ECDSA public key	Verify an ECDSA public key	ECDSA (Cert. A4978)	ECDSA public key	User	ECDSA public key – W	API return value
Generate digital signature	Generate a digital signature	RSA (Cert. A4978)	RSA private key	User	RSA private key – WE	API return value
Verify digital signature	Verify a digital signature	ECDSA (Cert. A4978) RSA (Cert. A4978)	ECDSA public key RSA public key	User	ECDSA public key – WE RSA public key – WE	API return value
Perform key wrap	Perform key wrap	KTS (Cert. A4978)	AES key AES CMAC key AES GMAC key AES GCM key AES GCM IV HMAC key	User	AES key – WE AES CMAC key – WE AES GMAC key – WE AES GCM key – WE AES GCM IV – WE HMAC key – WE	API return value
Perform key unwrap	Perform key unwrap	KTS (Cert. A4978)	AES key AES CMAC key AES GMAC key AES GCM key AES GCM IV HMAC key Triple-DES key	User	AES key – WE AES CMAC key – WE AES GMAC key – WE AES GCM key – WE AES GCM IV – WE HMAC key – WE Triple-DES key – WE	API return value
Compute shared secret	Compute DH/ECDH shared secret suitable for use as input to an internal TLS KDF	KAS-ECC-SSC (Cert. A4978) KAS-FFC-SSC (Cert. A4978)	DH public component DH private component ECDH public component ECDH private component TLS pre-master secret	User	DH public component – WE DH private component – WE ECDH public component – WE ECDH private component – WE TLS pre-master secret – GE	API return value
Derive SSH keys	Derive SSH session and integrity keys	KDF (SSH) (Cert. A4978)	SSH master secret AES key HMAC key	User	SSH master secret – WE AES key – GR HMAC key – GR	API return value
Derive TLS keys	Derive TLS session and integrity keys	KDF (TLS 1.0/1.1) (Cert. A4978) KDF (TLS 1.2) (Cert. A4978) KDF (TLS 1.3) (Cert. A4979)	TLS pre-master secret TLS master secret AES key AES GCM key AES GCM IV HMAC key	User	TLS pre-master secret – WE TLS master secret – GE AES key – GR AES GCM key – GR AES GCM IV – GR HMAC key – GR	API return value
Derive key via HKDF	Derive key from HKDF	HKDF (Cert. A4978)	AES key	User	AES key – GR	API return value
Derive key via PBKDF2	Derive key from PBKDF2	PBKDF (Cert. A4978)	Passphrase AES key Triple-DES key	User	Passphrase – WE AES key – GR Triple-DES key – GR	API return value

Per FIPS 140-3 Implementation Guidance 2.4.C, the* ***Show Status*, ***Zeroize****, and* ***Show Versioning Information*** *services do not require an Approved security service indicator.*

Table 9 below lists the non-approved services available to module operators.

Table 9 – Non-Approved Services

Service	Description	Algorithm(s) Accessed	Role	Indicator
Perform data encryption (non-compliant)	Perform symmetric data encryption	ARIA, Blake2, Blowfish, Camellia, CAST, CAST5, ChaCha20, DES, IDEA, RC2, RC4, RC5, SEED, SM4, Triple-DES (non-compliant)	User	API return value
Perform data decryption (non-compliant)	Perform symmetric data decryption	ARIA, Blake2, Blowfish, Camellia, CAST, CAST5, ChaCha20, DES, IDEA, RC2, RC4, RC5, SEED, SM4	User	API return value

Service	Description	Algorithm(s) Accessed	Role	Indicator
Perform MAC operations (non-compliant)	Perform message authentication operations	Poly1305, Triple-DES/CMAC (non-compliant for MAC generation)	User	API return value
Perform hash operation (non-compliant)	Perform hash operation	MD2, MD4, MD5, RIPEMD, RMD160, SM2, SM3, Whirlpool	User	API return value
Perform digital signature functions (non-compliant)	Perform digital signature functions	DSA (non-compliant), ECDSA (non-compliant), EdDSA, RSA (non-compliant)	User	API return value
Perform key encapsulation (non-compliant)	Perform key encapsulation functions	RSA (non-compliant)	User	API return value
Perform key un-encapsulation (non-compliant)	Perform key un-encapsulation functions	RSA (non-compliant)	User	API return value
Perform key wrap (non-compliant)	Perform key wrap functions	Triple-DES/CMAC (non-compliant)	User	API return value
Perform authenticated encryption/decryption (non-compliant)	Perform authenticated encryption/decryption	AES-OCB	User	API return value
Perform random number generation (non-compliant)	Perform random number generation	ANSI X9.31 RNG (with 128-bit AES core)	User	API return value
Perform key pair generation (non-compliant)	Perform key pair generation	DSA (non-compliant), ECDSA (non-compliant), EdDSA, RSA (non-compliant)	User	API return value

5. Software/Firmware Security

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 check on 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. The CO can initiate the pre-operational tests on demand by re-instantiating the module or issuing the `FIPS_selftest()` API command.

CorSSL is not a standalone application; it is a cryptographic toolkit intended for use in a with a vendor's solution. The module will be linked to a host application, and the host application will be pre-installed onto a target platform by the vendor or installed onto target platforms by the end-user. The module requires no configuration steps to be performed by application developers or end-users, and no action is required from developers or end-users to initialize the module for operation. The module is designed with a default entry point (DEP) that ensures that the pre-operational tests and conditional CASTs are initiated automatically when the module is loaded.

6. Operational Environment

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 environments provide 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

The cryptographic module is a software module and does not include physical security mechanisms. Therefore, per *ISO/IEC 19790:2012(E)* section 7.7.1, requirements for physical security are not applicable.

8. Non-Invasive Security

This section is not applicable. There are currently no approved non-invasive mitigation techniques referenced in *ISO/IEC 19790:2021* Annex F.

9. Sensitive Security Parameter Management

9.1 Keys and Other SSPs

The module supports the keys and other SSPs listed in Table 10. Note that all SSP import and export is electronic and is performed within the Tested OE's Physical Perimeter (TOEPP).

Table 10 – SSPs

Key/SSP Name/Type	Strength	Security Function and Cert. Number	Generation	Import / Export	Establishment	Storage	Zeroization	Use & Related Keys
Keys								
AES key (CSP)	Between 128 and 256 bits	AES (CBC, CCM, CFB, CTR, ECB, OFB, KW, KWP modes) (Cert. A4978) KTS (Cert. A4978)	-	Imported in plaintext via API parameter Never exported	Established via TLS or SSH KDF	Not persistently stored by the module	Unload module; Remove power	Symmetric encryption, decryption
AES GCM key (CSP)	Between 128 and 256 bits	AES (GCM mode) (Cert. A4978) KTS (Cert. A4978)	-	Imported in plaintext via API parameter Never exported	Established via TLS or SSH KDF	Not persistently stored by the module	Unload module; Remove power	Authenticated symmetric encryption, decryption
XTS-AES key (CSP)	128 or 256 bits	AES (XTS mode) (Cert. A4978)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Unload module; Remove power	Symmetric encryption, decryption
AES CMAC key (CSP)	Between 128 and 256 bits	AES (CMAC mode) (Cert. A4978) KTS (Cert. A4978)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Unload module; Remove power	MAC generation, verification
AES GMAC key (CSP)	Between 128 and 256 bits	AES (GMAC mode) (Cert. A4978) KTS (Cert. A4978)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Unload module; Remove power	MAC generation, verification
Triple-DES key (CSP)	-	Triple-DES (CBC, CFB1, CFB8, CFB64, ECB, OFB modes) (Cert. A4978) KTS (Cert. A4978)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Unload module; Remove power	Symmetric decryption; key unwrapping
Triple-DES CMAC key (CSP)	-	Triple-DES (CMAC mode) (Cert. A4978)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Unload module; Remove power	MAC verification
HMAC key (CSP)	112 bits (minimum)	HMAC (Cert. A4978) KTS (Cert. A4978)	-	Imported in plaintext via API parameter Never exported	Established via TLS or SSH KDF	Not persistently stored by the module	Unload module; Remove power	Keyed hash

Key/SSP Name/Type	Strength	Security Function and Cert. Number	Generation	Import / Export	Establishment	Storage	Zeroization	Use & Related Keys
DSA private key (CSP)	112 or 128 bits	DSA (Cert. A4978)	Generated via Approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	Digital signature generation
DSA public key (PSP)	112 or 128 bits	DSA (Cert. A4978)	Generated via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	Digital signature verification
ECDSA private key (CSP)	Between 112 and 256 bits	ECDSA (Cert. A4978)	Generated via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	Digital signature generation
ECDSA public key (PSP)	Between 112 and 256 bits	ECDSA (Cert. A4978)	Generated via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	Digital signature verification
RSA private key (CSP)	Between 112 and 150 bits	RSA (Cert. A4978) KTS (Cert. A4978)	Generated via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	Digital signature generation
RSA public key (PSP)	Between 80 and 150 bits	RSA (Cert. A4978) KTS (Cert. A4978)	Generated via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	Digital signature verification
DH private component (CSP)	112 bits	KAS-SSC-FFC (Cert. A4978)	Generated via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	DH shared secret computation
DH public component (PSP)	112 bits	KAS-SSC-FFC (Cert. A4978)	Generated via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	DH shared secret computation
ECDH private component (CSP)	Between 112 and 256 bits	KAS-SSC-ECC (Cert. A4978)	Generated via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	ECDH shared secret computation
ECDH public component (PSP)	Between 112 and 256 bits	KAS-SSC-ECC (Cert. A4978)	Generated via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Unload module; Remove power	ECDH shared secret computation
Other SSPs								
Passphrase (PSP)	-	PBKDF (Cert. A4978)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Unload module; Remove power	Input to PBKDF for key derivation
AES GCM IV (CSP)	-	AES (GCM mode) (Cert. A4978)	Generated in compliance with the provisions of a peer-to-peer industry standard protocol	-	-	Not persistently stored by the module	Unload module; Remove power	Initialization vector for AES GCM

Key/SSP Name/Type	Strength	Security Function and Cert. Number	Generation	Import / Export	Establishment	Storage	Zeroization	Use & Related Keys
SSH shared secret (CSP)	-	KDF (SSH) (Cert. A4978)	-	Imported in plaintext via API parameter Exported in plaintext via API parameter	Established via ECC/FFC shared secret computation	Not persistently stored by the module	Unload module; Remove power	Derivation of the AES key and HMAC key used for securing SSH connections
TLS pre-master secret (CSP)	-	KDF (TLS 1.0/1.1) (Cert. A4978) KDF (TLS 1.2) (Cert. A4978) KDF (TLS 1.3) (Cert. A4979)	-	Imported in plaintext via API parameter Exported in plaintext via API parameter	Established via ECC/FFC shared secret computation	Not persistently stored by the module	Unload module; Remove power	Derivation of the TLS master secret
TLS master secret (CSP)	-	KDF (TLS 1.0/1.1) (Cert. A4978) KDF (TLS 1.2) (Cert. A4978) KDF (TLS 1.3) (Cert. A4979)	-	-	Established via TLS KDF (using imported TLS pre-master secret)	Not persistently stored by the module	Unload module; Remove power	Derivation of the AES/AES-GCM key and HMAC key used for securing TLS connections
DRBG entropy input (CSP)	-	DRBG (Cert. A4978)	-	Imported in plaintext via API parameter[47]; Never exported	-	Not persistently stored by the module	Unload module; Remove power	Entropy material for DRBG
DRBG seed (CSP)	-	DRBG (Cert. A4978)	Generated using nonce along with DRBG entropy input	-	-	Not persistently stored by the module	Unload module; Remove power	Seeding material for DRBG
DRBG 'V' value (CSP)	-	DRBG (Cert. A4978)	Generated	-	-	Not persistently stored by the module	Unload module; Remove power	State values for DRBG
DRBG 'Key' value (CSP)	-	DRBG (Cert. A4978)	Generated	-	-	Not persistently stored by the module	Unload module; Remove power	State values for DRBG

9.2 DRBGs

The module implements the following Approved DRBG:

- Counter-based DRBG

This DRBG is used to generate random values at the request of the calling application. Outputs from this DRBG are also used as seeds in the generation of asymmetric key pairs.

The module implements the following non-Approved DRBGs (which are only available in the non-Approved mode of operation):

- Hash-based DRBG (non-compliant)
- HMAC-based DRBG (non-compliant)
- ANSI X9.31 RNG (non-Approved)

[47] The module relies on entropy input received from the calling application, which is outside of the logical cryptographic boundary. As such, there is no assurance of the minimum strength of generated keys.

9.3 SSP Storage Techniques

There is no mechanism within the module's cryptographic boundary for the persistent storage of SSPs. 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.

9.4 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. For the zeroization of keys in volatile memory, module operators can unload the module from memory or reboot/power-cycle the host device.

9.5 RBG Entropy Sources

Table 11 below specifies the module's entropy sources.

Table 11 – Non-Deterministic Random Number Generation Specification

Entropy Source(s)	Minimum Number of Bits of Entropy	Details
Calling application	256	256 bits of seed material are provided to the module's DRBG by the calling application. The calling application and its entropy sources are outside the module's cryptographic boundary. The calling application shall use entropy sources that meet the security strength required for the CTR_DRBG as shown in *NIST SP 800-90Arev1*, Table 3. This entropy shall be supplied by means of a callback function. The callback function must return an error if the minimum entropy strength cannot be met.

10. Self-Tests

Both pre-operational and conditional self-tests are performed by the module. 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 following pre-operational self-test(s):

- Software integrity test for libcrypto (using an HMAC SHA2-256 digest)
- Software integrity test for libssl (using an HMAC SHA2-256 digest)

10.2 Conditional Self-Tests

The module performs the following conditional self-tests:

- Conditional cryptographic algorithm self-tests (CASTs)
  - AES GCM encrypt KAT[48] (128-bit)
  - AES GCM decrypt KAT (128-bit)
  - XTS-AES encrypt KAT (128-bit)
  - XTS-AES decrypt KAT (256-bit)
  - Triple-DES ECB decrypt KAT (3-Key)
  - Triple-DES CMAC verify KAT (3-key)
  - CRNGT[49] for the entropy input
  - CTR_DRBG KAT (AES, 256-bit, with derivation function)
  - CTR_DRBG generate/instantiate/reseed KAT (256-bit AES)
  - DSA verify KAT (P-224 and K-233 curve, SHA2-256)
  - ECDSA verify KAT (P-224 and K-233 curve, SHA2-256)
  - HKDF KAT
  - HMAC KAT (SHA2-256)
  - RSA sign KAT (2048-bit; SHA2-256; PKCS#1.5 scheme)
  - RSA verify KAT (2048-bit; SHA2-256; PKCS#1.5 scheme)
  - SHA KATs (SHA-1, SHA2-512, SHA3-256)
  - FFC DH Shared Secret “Z” Computation KAT (2048-bit)
  - ECC CDH Shared Secret “Z” Computation KAT (P-224 curve)
  - PBKDF2 KAT
  - SSH KDF KAT
  - TLS KDF KAT (1.0/1.1, 1.2)
  - TLS KDF KAT (1.3)

[48] KAT – Known Answer Test

[49] CRNGT – Continuous Random Number Generator Test

To ensure all CASTs are performed prior to the first operational use of the associated algorithm, all CASTs are performed during the module's initial power-up sequence. The SHA and HMAC KATs are performed prior to the pre-operational software integrity test; all other CASTs are executed after the successful completion of the software integrity test.

- Conditional pair-wise consistency tests (PCTs)
  - DSA sign/verify PCT[50]
  - ECDSA sign/verify PCT
  - RSA sign/verify PCT (SHA-256)
  - DH key generation PCT
  - ECDH key generation PCT

10.3 Self-Test Failure Handling

The module reaches the critical error state when any self-test fails. Upon test failure, the module will set an internal flag and enter a critical error state. In this state, the module will no longer perform cryptographic services or output data over the data output interfaces. For any subsequent request for cryptographic services, the module will return a failure indicator.

To recover, the module must be re-instantiated by the calling application. If the pre-operational self-tests complete successfully, then the module can resume normal operations. 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.

[50] PCT – Pairwise Consistency Test

11. Life-Cycle Assurance

The sections below describe how to ensure the module is operating in its validated configuration, including the following:

- Procedures for secure installation, initialization, startup, and operation of the module
- Maintenance requirements
- Administrator and non-Administrator guidance

Operating the module without following the guidance herein (including the use of undocumented services) will result in non-compliant behavior and is outside the scope of this Security Policy.

11.1 Secure Installation

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

11.2 Initialization

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.

The pre-operational integrity test and cryptographic algorithm self-tests are performed automatically via a default entry point (DEP) when the module is loaded for execution, without any specific action from the calling application or the end-user. End-users have no means to short-circuit or bypass these actions. Failure of any of the initialization actions will result in a failure of the module to load for execution.

11.3 Setup

No setup steps are required to be performed by end-users.

11.4 Administrator Guidance

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

The following list provides additional guidance for module administrators:

- 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.

- The `OpenSSL_version()` API can be used to obtain the module's versioning information. The API call will return "CorSSL 1.1.1s.006", which correlates to the module name and version information on the module's FIPS 140-3 validation certificate.

- The CO can initiate the pre-operational self-tests and CASTs on demand for periodic testing of the module by re-instantiating the module or issuing the `FIPS_selftest()` API command.

11.5 Non-Administrator Guidance

The following list provides additional policies for non-Administrators:

- The module uses PBKDF2 option 1a from section 5.4 of *NIST SP 800-132*.
  - 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 passphrase used in the PBKDF shall be of 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}.
  - Passphrases (used as an input for the PBKDF) shall not be used as cryptographic keys.
  - Keys derived from passphrases may only be used in storage applications.

- The length of a single data unit encrypted or decrypted with the AES-XTS shall not exceed 2^{20} AES blocks; that is, 16 MB of data per AES-XTS instance. An XTS instance is defined in section 4 of *NIST SP 800-38E*.

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

- 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 generates the IV as follows:
  - For TLS v1.2, the module supports acceptable AES GCM cipher suites from section 3.3.1 of *NIST SP 800-52rev2*. Per scenario 1 in *FIPS 140-3 IG* C.H, the mechanism for IV generation is compliant with *RFC 5288*. The counter portion of the IV is strictly increasing. When the IV exhausts the maximum number of possible values for a given session key, a failure in encryption will occur and a handshake to establish a new encryption key will be required. It is the responsibility of the module operator (i.e., the first party, client, or server) to trigger this handshake in accordance with *RFC 5246* when this condition is encountered.

  The module also supports internal IV generation using the module's Approved DRBG. The IV is at least 96 bits in length per section 8.2.2 of *NIST SP 800-38D*. Per *NIST SP 800-38D* and scenario 2 of *FIPS 140-3 IG* C.H, the DRBG generates outputs such that the (key/IV) pair collision probability is less than 2^{-32}.

In the event that power to the module is lost and subsequently restored, the calling application must ensure that any AES-GCM keys used for encryption or decryption are re-distributed.

- 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 186-4* (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 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)

  Note that several of the assurances required by the key pair owner are provided by the fact that the module itself, when acting as the key pair owner, generates the key pairs.

  The module includes the capability to provide the required recipient assurance of public key validity specified in section 5.6.2.2 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. Validation of another module's public key is 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.

- The calling application is responsible for using entropy sources that meet the minimum security strength of 112 bits required for the CTR_DRBG as shown in *NIST SP 800-90Arev1*, Table 3.

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

Table 12 – Acronyms and Abbreviations

Term	Definition
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interface
CAST	Cryptographic Algorithm Self-Test
CBC	Cipher Block Chaining
CCCS	Canadian Centre for Cyber Security
CCM	Counter with Cipher Block Chaining - Message Authentication Code
CFB	Cipher Feedback
CKG	Cryptographic Key Generation
CMAC	Cipher-Based Message Authentication Code
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
ECC	Elliptic Curve Cryptography
ECC CDH	Elliptic Curve Cryptography Cofactor Diffie-Hellman
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GMAC	Galois Message Authentication Code

Term	Definition
GPC	General-Purpose Computer
HMAC	(keyed-) Hash Message Authentication Code
KAS	Key Agreement Scheme
KAT	Known Answer Test
KDF	Key Derivation Function
KTS	Key Transport Scheme
KW	Key Wrap
KWP	Key Wrap with Padding
MD	Message Digest
NIST	National Institute of Standards and Technology
OCB	Offset Codebook
OE	Operational Environment
OFB	Output Feedback
OS	Operating System
PBKDF	Password-Based Key Derivation Function
PCT	Pairwise Consistency Test
PKCS	Public Key Cryptography Standard
PSS	Probabilistic Signature Scheme
PUB	Publication
RC	Rivest Cipher
RNG	Random Number Generator
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
SHAKE	Secure Hash Algorithm KECCAK
SHS	Secure Hash Standard
SP	Special Publication
SSC	Shared Secret Computation
SSP	Sensitive Security Parameter
TDES	Triple Data Encryption Standard
TLS	Transport Layer Security
TOEPP	Tested OE’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 security 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 security service indicator value based on the parameter(s) passed to them.

- **EVP_cipher_get_service_indicator()** is used to return the appropriate Approved security service indicator status for block ciphers like AES and Triple DES.

- **DSA_get_service_indicator()** is used to return the appropriate Approved security service indicator status for the DSA algorithm and its modes. You must include the mode you want the indicator for, which is to be specified in the DSA_MODES_t enum.

- **RSA_key_get_service_indicator()** is used to return the appropriate Approved security service indicator status for RSA algorithm and its modes.

- **PBKDF_get_service_indicator()** is used to return the appropriate Approved security service indicator status for PBKDF usage.

- **EVP_Digest_get_service_indicator()** is used to return the appropriate Approved security service indicator status for SHS algorithms like SHA-1 and SHAKE.

- **EC_key_get_service_indicator()** is used to return the appropriate Approved security service indicator status for elliptic curve algorithms like ECDSA and its modes.

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

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

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

- **TLS1_3_kdf_get_service_indicator()** is used to return the appropriate Approved security 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 security service indicator status for DRBG usage.

Return Values

Each function returns “1” when indicating the usage of approved services and “0” for non-approved security services.

Notes

When calling a I<get> function, always call it after the variables have been finalized but before they are freed or destroyed.

Examples

The code sample below provides examples of how to check the Approved security service indicators for Triple-DES (3-key, in ECB mode) encryption and decryption:

```
int 3des_indicator_test()
  {
    static EVP_CIPHER *cipher = NULL;
    static EVP_CIPHER_CTX *ctx;
    int outLen;
    unsigned char pltmp[8];
    unsigned char citmp[8];
        unsigned char key[] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
                               19,20,21,22,23,24};
        unsigned char plaintext[] = { 'e', 't', 'a', 'o', 'n', 'r', 'i', 's' };
    cipher = EVP_des_ede3_ecb();

    //Encrypt
    ctx = EVP_CIPHER_CTX_new();
    EVP_EncryptInit_ex(ctx, cipher, NULL, key, NULL);
    EVP_CIPHER_CTX_set_key_length(ctx, 24);
    EVP_EncryptUpdate(ctx, citmp, &outLen, plaintext, 8);

    // Check the indicator
    int NID = EVP_CIPHER_CTX_nid(ctx);
    fprintf(stdout,"EVP_des_ede3_ecb (NID %i) encrypt indicator = %i\n", NID, EVP_cipher_get_service_indicator(ctx));
    EVP_CIPHER_CTX_cleanup(ctx);


    //Decrypt
    ctx = EVP_CIPHER_CTX_new();
    EVP_DecryptInit_ex(ctx, cipher, NULL, key, NULL);
    EVP_CIPHER_CTX_set_key_length(ctx, 24);
    EVP_DecryptUpdate(ctx, pltmp, &outLen, citmp, 8);

    // Check the indicator
    fprintf(stdout,"EVP_des_ede3_ecb (NID %i) decrypt indicator = %i\n", NID, EVP_cipher_get_service_indicator(ctx));
    EVP_CIPHER_CTX_cleanup(ctx);
    EVP_CIPHER_CTX_free(ctx);
  }
```

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