Infinidat, Ltd.

Infinidat Cryptographic Module

Software Version: 1.1.1

FIPS 140-3 Non-Proprietary Security Policy

FIPS Security Level: 1
Document Version: 1.1

Prepared for:

Infinidat, Ltd.
500 Totten Pond Road
Waltham, MA, 02451
United States of America

Phone: +1 855 900 4634
www.infinidat.com

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

Table of Contents

List of Tables

List of Figures

1. General

1.1 Overview

This is a non-proprietary Cryptographic Module Security Policy for the Infinidat Cryptographic Module (version: 1.1.1) from Infinidat, Ltd. (Infinidat). This Security Policy describes how the Infinidat Cryptographic Module meets the security requirements of Federal Information Processing Standards (FIPS) Publication 140-3, which details the U.S. and Canadiaen 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. The Infinidat Cryptographic Module is referred to in this document as Infinidat Cryptographic Module or the module.

1.1.1 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 Infinidat website www.infinidat.com contains information on the full line of services and solutions from Infinidat.
- 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.

1.1.2 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

The Infinidat Cryptographic Module 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

Section	Title	Security Level
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

The module has an overall security level of 1.

2. Cryptographic Module Specification

2.1 Description

2.1.1 Purpose and Use

The Infinidat Cryptographic Module is a set of cryptographic libraries that implement TLS[1], symmetric key generation and encryption/decryption, and SED[2] authentication key derivation for the InfuzeOS™[3], which is the core component of the InfiniBox and InfiniBox SSA appliances. The Infinidat Cryptographic Module comes pre-installed on each InfiniBox node.

2.1.2 Module Type

The Infinidat Cryptographic Module is a Software module.

2.1.3 Module Embodiment

The Infinidat Cryptographic Module has a Multi-Chip Standalone embodiment.

The module is designed to utilize the following processor algorithm acceleration (PAA) instruction sets for its AES and SHA implementations:

- AES-NI instruction set, when executing on the InfuzeOS™ 8 operational environment

The module was tested and found to be compliant with *FIPS 140-3* requirements on the environments listed in section 2.2.4 of this Security Policy.

2.1.4 Module Characteristics

The module does not have any additional characteristics.

2.1.5 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. The module's software component comprises two shared library files and two digest files for testing integrity. All hardware and software components are contained within the host platform's physical enclosure.

[1] TLS – Transport Layer Security
[2] SED – Self-Encrypting Drive
[3] OS – Operating System

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 shows the logical 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 supports PAA. Thus, the diagram includes the CPU as a component within the cryptographic boundary.

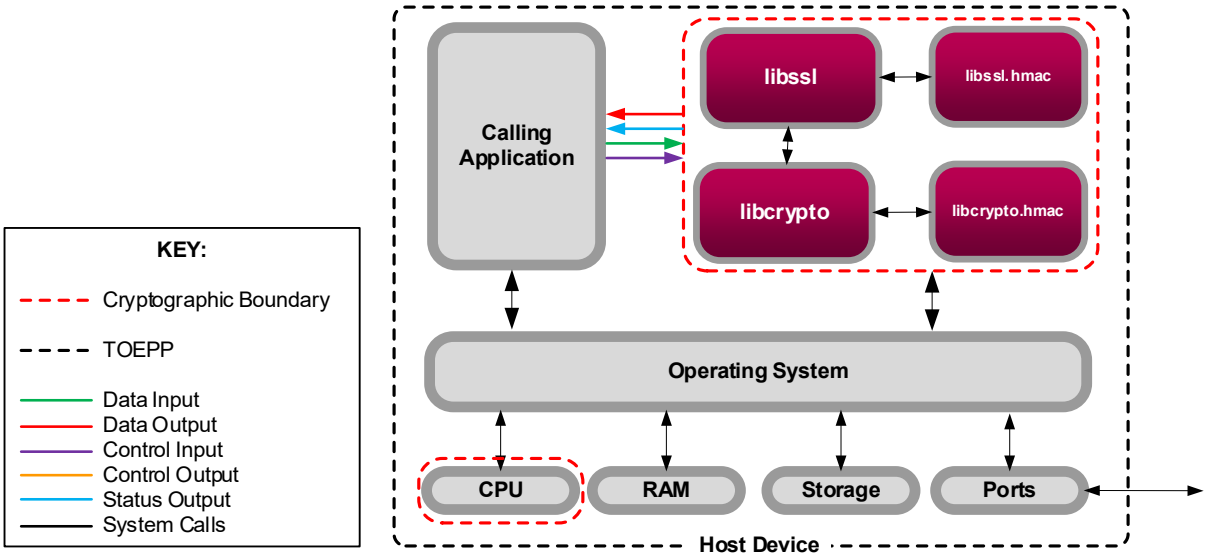

Figure 1: Module Block Diagram (with Cryptographic Boundary)

The module is entirely contained within the TOEPP.

2.1.6 Tested Operational Environment's Physical Perimeter (TOEPP)

As a software cryptographic module, the TOEPP of the cryptographic module is defined by the host platform on which the module is installed.

2.2 Tested and Vendor Affirmed Module Version and Identification

2.2.1 Tested Module Identification - Hardware

This section is only applicable to 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
libcrypto.so libssl.so libcrypto.hmac libssl.hmac	1.1.1		Yes

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

The module count is one (1).

2.2.3 Tested Module Identification – Hybrid Disjoint Hardware

This section is only applicable for 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)
InfuzeOS 8	HPE ProLiant DL345 Gen 11	AMD EPYC (Zen 4)	Yes		1.1.1
InfuzeOS 8	HPE ProLiant DL345 Gen 11	AMD EPYC (Zen 4)	No		1.1.1

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.

The cryptographic module maintains validation compliance when operating on any general-purpose computer (GPC) provided that the GPC uses any single-user operating system/mode specified on the validation certificate, or another compatible single-user 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.

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

The vendor does not affirm any operational environments.

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 table below lists the modes of operation for the module.

Mode Name	Description	Type	Status Indicator
Approved	Mode allows the use of cryptographic operations	Approved	The module implements separate indicators for approved security services and follows IG 2.3.A Example Scenario 1. However, some indicators are shared between approved security services and fall under IG 2.3.A Example Scenario 3. The *_get_service_indicator() APIs return 1 if Approved. Please refer to Appendix B. Approved Service Indicators for the specific indicators per service.
Non-Approved	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.	Non-Approved	

Table 4: Modes List and Description

2.5 Algorithms

2.5.1 Approved Algorithms

The module employs cryptographic algorithm implementations from the following sources:

- Infinidat Cryptographic Module (libcrypto) 1.0 (Cert. A5477)
- Infinidat Cryptographic Module (libssl) 1.0 (Cert. A5478)

The module implements the Approved algorithms listed below.

Infinidat Cryptographic Module (libcrypto)

Algorithm	CAVP Cert	Properties	Reference
AES-CBC	A5477	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CCM	A5477	Key Length - 128, 192, 256	SP 800-38C
AES-CFB1	A5477	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CFB128	A5477	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CFB8	A5477	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CMAC	A5477	Direction - Generation, Verification Key Length - 128, 192, 256	SP 800-38B
AES-CTR	A5477	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-ECB	A5477	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-GCM	A5477	Direction - Decrypt, Encrypt IV Generation - Internal IV Generation Mode - 8.2.1 Key Length - 128, 192, 256	SP 800-38D
AES-GMAC	A5477	Direction - Decrypt, Encrypt IV Generation - Internal IV Generation Mode - 8.2.1 Key Length - 128, 192, 256	SP 800-38D
AES-KW	A5477	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38F
AES-KWP	A5477	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38F
AES-OFB	A5477	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-XTS Testing Revision 2.0	A5477	Direction - Decrypt, Encrypt Key Length - 128, 256	SP 800-38E
Counter DRBG	A5477	Prediction Resistance - No, Yes Mode - AES-128, AES-192, AES-256 Derivation Function Enabled - Yes	SP 800-90A Rev. 1
ECDSA KeyGen (FIPS186-5)	A5477	Curve - P-224, P-256, P-384, P-521 Secret Generation Mode - testing candidates	FIPS 186-5
ECDSA KeyVer (FIPS186-5)	A5477	Curve - P-224, P-256, P-384, P-521	FIPS 186-5
ECDSA SigGen (FIPS186-5)	A5477	Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-5
ECDSA SigVer (FIPS186-5)	A5477	Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-5
HMAC-SHA-1	A5477	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-224	A5477	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-256	A5477	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-384	A5477	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-512	A5477	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-224	A5477	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-256	A5477	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-384	A5477	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-512	A5477	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
KAS-ECC-SSC Sp800-56Ar3	A5477	Domain Parameter Generation Methods - P-224, P-256, P-384, P-521 Scheme - ephemeralUnified - KAS Role - initiator, responder	SP 800-56A Rev. 3

Algorithm	CAVP Cert	Properties	Reference
KDF SP800-108	A5477	KDF Mode - Counter Supported Lengths - Supported Lengths: 8-4096 Increment 8	SP 800-108 Rev. 1
PBKDF	A5477	Iteration Count - Iteration Count: 10-10000 Increment 1 Password Length - Password Length: 8-128 Increment 1	SP 800-132
RSA KeyGen (FIPS186-5)	A5477	Key Generation Mode - probable Modulo - 2048, 3072, 4096 Primality Tests - 2powSecStr Private Key Format - standard	FIPS 186-5
RSA SigGen (FIPS186-5)	A5477	Modulo - 2048, 3072, 4096 Signature Type - pkcs1v1.5, pss	FIPS 186-5
RSA SigVer (FIPS186-5)	A5477	Modulo - 2048, 3072, 4096 Signature Type - pkcs1v1.5, pss	FIPS 186-5
SHA-1	A5477	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA2-224	A5477	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA2-256	A5477	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA2-384	A5477	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA2-512	A5477	Message Length - Message Length: 0-65528 Increment 8	FIPS 180-4
SHA3-224	A5477	Message Length - Message Length: 0-65528 Increment 8	FIPS 202
SHA3-256	A5477	Message Length - Message Length: 0-65528 Increment 8	FIPS 202
SHA3-384	A5477	Message Length - Message Length: 0-65528 Increment 8	FIPS 202
SHA3-512	A5477	Message Length - Message Length: 0-65528 Increment 8	FIPS 202
SHAKE-128	A5477	Output Length - Output Length: 16-1024 Increment 8	FIPS 202
SHAKE-256	A5477	Output Length - Output Length: 16-1024 Increment 8	FIPS 202
TLS v1.2 KDF RFC7627 (CVL)	A5477	Hash Algorithm - SHA2-256, SHA2-384, SHA2-512	SP 800-135 Rev. 1

Table 5: Approved Algorithms - Infinidat Cryptographic Module (libcrypto)

Legacy

Algorithm	CAVP Cert	Properties	Reference
DSA SigVer (FIPS186-4)	A5477	L - 1024, 2048, 3072 N - 160, 224, 256 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
RSA SigVer (FIPS186-4)	A5477	Signature Type - ANSI X9.31, PKCS 1.5, PKCSPSS Modulo - 1024, 2048, 3072, 4096	FIPS 186-4
TDES-CBC	A5477	Direction - Decrypt	SP 800-67 Rev. 2
TDES-CFB1	A5477	Direction - Decrypt	SP 800-67 Rev. 2
TDES-CFB64	A5477	Direction - Decrypt	SP 800-67 Rev. 2
TDES-CFB8	A5477	Direction - Decrypt	SP 800-67 Rev. 2
TDES-CMAC	A5477	Direction - Verification	SP 800-67 Rev. 2
TDES-ECB	A5477	Direction - Decrypt	SP 800-67 Rev. 2
TDES-OFB	A5477	Direction - Decrypt	SP 800-67 Rev. 2

Table 6: Approved Algorithms - Legacy

Infinidat Cryptographic Module (libssl)

Algorithm	CAVP Cert	Properties	Reference
TLS v1.3 KDF (CVL)	A5478	HMAC Algorithm - SHA2-256, SHA2-384 KDF Running Modes - DHE, PSK, PSK-DHE	SP 800-135 Rev. 1

Table 7: Approved Algorithms - Infinidat Cryptographic Module (libssl)

The Triple-DES for decryption/unwrapping, DSA for signature verification, and RSA specific to *FIPS 186-4* for signature verification (ANSI X9.31 and 1024-bit modulo capabilities) algorithms are Approved for legacy usage only. These legacy algorithms can only be used on data generated before the Legacy Date specified in *FIPS 140-3 IG* C.M.

2.5.2 Vendor Affirmed Algorithms

The vendor affirms the following cryptographic security methods:

Name	Properties	Implementation	Reference
CKG	Key Type:Asymmetric	Infinidat Cryptographic Module (libcrypto)	SP 800-133 Rev. 2 Section 4 Example 1

Table 8: 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.

Name	Properties	Implementation	Reference
AES (any approved mode of AES)	Use:Key Unwrap	Infinidat Cryptographic Module (libcrypto)	IG D.G

Table 9: Non-Approved, Allowed Algorithms

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 with external IV)	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
DH	Key generation, key verification, domain parameter generation, domain parameter verification, key agreement, shared secret computation
DSA (non-compliant)	Key generation, domain parameter generation, domain parameter verification, digital signature generation, digital signature verification
ECDH (non-compliant)	Key agreement, shared secret computation
ECDSA (non-compliant)	Key pair generation; digital signature generation; digital signature verification
EdDSA	Key pair generation; digital signature generation; digital signature verification

Name	Use and Function
IDEA	Encryption/decryption
Hash_DRBG (non-compliant)	Random bit generation
HKDF (non-compliant)	Key derivation
HMAC DRBG (non-compliant)	Random bit generation
KBKDF (non-compliant)	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 (non-compliant)	Key pair generation; digital signature generation; digital signature verification; key transport
SEED	Encryption/decryption
SM2	Message digest
SM3	Message digest
SM4	Encryption/decryption
TDES	Encryption and key wrapping
TDES-CMAC	MAC generation
TLS v1.0/v1.1 KDF (non-compliant)	Key derivation
Whirlpool	Message digest

Table 10: 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/Decryption	BC-UnAuthEncrypt BC-UnAuthDecrypt	Block cipher unauthenticated.	Publication:SP 800-38A	AES-CBC: (A5477) AES-CFB1: (A5477) AES-CFB8: (A5477) AES-CFB128: (A5477) AES-CTR: (A5477) AES-ECB: (A5477) AES-OFB: (A5477)
AES-CMAC for Message Authentication	MAC	Message authentication.	Publication:SP 800-38B	AES-CMAC: (A5477)
AES-GMAC for Message Authentication	MAC	Message authentication.	Publication:SP 800-38D	AES-GMAC: (A5477)
AES-CCM for Authenticated Symmetric Encryption/Decryption	BC-AuthEncrypt BC-AuthDecrypt	Block cipher authenticated.	Publication:SP 800-38C	AES-CCM: (A5477) AES-ECB: (A5477)
AES-GCM for Authenticated Symmetric Encryption/Decryption	BC-AuthEncrypt BC-AuthDecrypt	Block cipher authenticated.	Publication:SP 800-38D	AES-GCM: (A5477) AES-CBC: (A5477)
AES-XTS for Symmetric Encryption/Decryption	BC-UnAuthEncrypt BC-UnAuthDecrypt	Block cipher unauthenticated.	Publication:SP 800-38E	AES-XTS Testing Revision 2.0: (A5477) AES-ECB: (A5477)

Name	Type	Description	Properties	Algorithms
DRBG	DRBG	Deterministic random bit generator.	Publication:SP 800-90A Rev. 1	Counter DRBG: (A5477)
DSA for Signature Verification	DigSig-SigVer	Digital signature verification.	Publication:FIPS 186-4	DSA SigVer (FIPS186-4): (A5477) SHA-1: (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477)
ECDSA for Key Generation	AsymKeyPair-KeyGen	Used to generate the ECDSA private key and ECDSA public key, which are used for digital signature generation and verification, respectively	Publication:FIPS 186-5	ECDSA KeyGen (FIPS186-5): (A5477) Counter DRBG: (A5477) CKG: ()
ECDSA for Key Verification	AsymKeyPair-KeyVer	Asymmetric key-pair verification.	Publication:FIPS 186-5	ECDSA KeyVer (FIPS186-5): (A5477)
ECDSA for Signature Generation	DigSig-SigGen	Uses the ECDSA private key to generate digital signatures.	Publication:FIPS 186-5	ECDSA SigGen (FIPS186-5): (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477) Counter DRBG: (A5477)
ECDSA for Signature Verification	DigSig-SigVer	Uses the ECDSA public key to verify digital signatures.	Publication:FIPS 186-5	ECDSA SigVer (FIPS186-5): (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477)
HMAC for Message Authentication	MAC	Message authentication.	Publication:FIPS 198-1	HMAC-SHA-1: (A5477) HMAC-SHA2-224: (A5477) HMAC-SHA2-256: (A5477) HMAC-SHA2-384: (A5477) HMAC-SHA2-512: (A5477) HMAC-SHA3-224: (A5477) HMAC-SHA3-256: (A5477) HMAC-SHA3-384: (A5477) HMAC-SHA3-512: (A5477) SHA-1: (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477) SHA3-224: (A5477) SHA3-256: (A5477) SHA3-384: (A5477) SHA3-512: (A5477)

Name	Type	Description	Properties	Algorithms
ECDH Shared Secret Computation	KAS-SSC	ECDH shared secret computation.	Publication:SP 800-56A Rev. 3	KAS-ECC-SSC Sp800-56Ar3: (A5477) ECDSA KeyGen (FIPS186-5): (A5477) ECDSA KeyVer (FIPS186-5): (A5477) Counter DRBG: (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477)
AES+MAC for Key Wrapping/Unwrapping	KTS-Wrap	Any Approved mode of AES with CMAC, GMAC, or HMAC used for key wrapping/key unwrapping.	Publication:Per FIPS 140-3 Implementation Guidance D.G, AES with CMAC, GMAC, or HMAC is an Approved key transport technique. Key Strength:Key establishment methodology provides between 128 and 256 bits of encryption strength.	AES-CBC: (A5477) AES-CFB1: (A5477) AES-CFB8: (A5477) AES-CFB128: (A5477) AES-CTR: (A5477) AES-ECB: (A5477) AES-OFB: (A5477) AES-CMAC: (A5477) AES-GMAC: (A5477) HMAC-SHA-1: (A5477) HMAC-SHA2-224: (A5477) HMAC-SHA2-256: (A5477) HMAC-SHA2-384: (A5477) HMAC-SHA2-512: (A5477) HMAC-SHA3-224: (A5477) HMAC-SHA3-256: (A5477) HMAC-SHA3-384: (A5477) HMAC-SHA3-512: (A5477) SHA-1: (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477) SHA3-224: (A5477) SHA3-256: (A5477) SHA3-384: (A5477) SHA3-512: (A5477)
AES-CCM for Key Wrapping/Unwrapping	KTS-Wrap	AES-CCM used for key wrapping/key unwrapping.	Publication:Per FIPS 140-3 Implementation Guidance D.G, AES-CCM is an Approved key transport technique. Key Strength:Key Strength: Key establishment methodology provides between 128 and 256 bits of encryption strength.	AES-CCM: (A5477) AES-CBC: (A5477)

Name	Type	Description	Properties	Algorithms
AES-GCM for Key Wrapping/Unwrapping	KTS-Wrap	AES-GCM used for key wrapping/key unwrapping.	Publication:Per FIPS 140-3 Implementation Guidance D.G, AES-GCM is an Approved key transport technique Key Strength:Key establishment methodology provides between 128 and 256 bits of encryption strength.	AES-GCM: (A5477) AES-CBC: (A5477)
AES-KW/KWP for Key Wrapping/Unwrapping	KTS-Wrap	AES-KW/KWP used for key wrapping/key unwrapping.	Publication:SP 800-38F Key Strength: Key establishment methodology provides between 128 and 256 bits of encryption strength.	AES-KW: (A5477) AES-KWP: (A5477)
KBKDF	KBKDF	Key-based key derivation.	Publication:SP 800-108 Rev. 1	KDF SP800-108: (A5477) HMAC-SHA2-256: (A5477) HMAC-SHA2-512: (A5477) SHA2-256: (A5477) SHA2-512: (A5477)
PBKDF	PBKDF	Password-based key derivation.	Publication:SP 800-132	PBKDF: (A5477) SHA-1: (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477) SHA3-224: (A5477) SHA3-256: (A5477) SHA3-384: (A5477) SHA3-512: (A5477) HMAC-SHA-1: (A5477) HMAC-SHA2-224: (A5477) HMAC-SHA2-256: (A5477) HMAC-SHA2-384: (A5477) HMAC-SHA2-512: (A5477) HMAC-SHA3-224: (A5477) HMAC-SHA3-256: (A5477) HMAC-SHA3-384: (A5477) HMAC-SHA3-512: (A5477)
RSA for Key Generation	AsymKeyPair-KeyGen	Key generation.	Publication:FIPS 186-5	RSA KeyGen (FIPS186-5): (A5477) Counter DRBG: (A5477) CKG: ()

Name	Type	Description	Properties	Algorithms
RSA for Signature Generation	DigSig-SigGen	Signature generation.	Publication:FIPS 186-5	RSA SigGen (FIPS186-5): (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477) Counter DRBG: (A5477)
RSA for Signature Verification	DigSig-SigVer	Signature verification.	Publication:FIPS 186-5	RSA SigVer (FIPS186-5): (A5477) RSA SigVer (FIPS186-4): (A5477) SHA-1: (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477)
SHA for Message Digest	SHA	Message digest.	Publication:FIPS 180-4	SHA-1: (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477)
SHA3 for Message Digest	SHA	Message digest.	Publication:FIPS 202	SHA3-224: (A5477) SHA3-256: (A5477) SHA3-384: (A5477) SHA3-512: (A5477)
SHAKE for Extendable Output Function	XOF	Extendable output function.	Publication:FIPS 202	SHAKE-128: (A5477) SHAKE-256: (A5477)
TDES for Symmetric Decryption	BC-UnAuth	Symmetric decryption.	Publication:SP 800-67 Rev. 2	TDES-ECB: (A5477) TDES-CBC: (A5477) TDES-OFB: (A5477) TDES-CFB64: (A5477) TDES-CFB8: (A5477) TDES-CFB1: (A5477)
TDES-CMAC for Message Authentication	MAC	Message authentication (verification).	Publication:SP 800-67 Rev. 2	TDES-CMAC: (A5477)
TLS v1.2 Key Agreement (ECDH)	KAS-Full	Key agreement.	Caveat:No part of the TLS v1.2 protocol, other than the KDF, has been tested by the CAVP and CMVP Key Strength:Key establishment methodology provides between 128 and 256 bits of encryption strength. Publication:SP 800-56A Rev. 3	TLS v1.2 KDF RFC7627: (A5477) KAS-ECC-SSC Sp800-56Ar3: (A5477) ECDSA KeyGen (FIPS186-5): (A5477) ECDSA KeyVer (FIPS186-5): (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477) Counter DRBG: (A5477)
TLS v1.2 Authentication	DigSig-SigVer	ECDSA or RSA signature verification used for authentication during TLS v1.3 session negotiation.	Publication:FIPS 186-5	ECDSA SigVer (FIPS186-5): (A5477) RSA SigVer (FIPS186-5): (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477)

Name	Type	Description	Properties	Algorithms
TLS v1.2 Data Encryption/Decryption	BC-Auth BC-UnAuth MAC	TLS v1.2 encryption, decryption, and authentication of TLS session packets, which are used by the TLS Session Key and TLS Authentication Key.	Publication:SP 800-38A, SP 800-38D, FIPS 198-1	AES-GCM: (A5477) Key Length: 128, 256 AES-CBC: (A5477) Key Length: 128, 256 Counter DRBG: (A5477) HMAC-SHA2-256: (A5477) HMAC-SHA2-384: (A5477) SHA2-256: (A5477) SHA2-384: (A5477)
TLS v1.2 Key Derivation	KAS-135KDF	TLS v1.2 key derivation.	Caveat:No part of the TLS v1.2 protocol, other than the KDF, has been tested by the CAVP and CMVP. Publication:SP 800-135 Rev. 1	TLS v1.2 KDF RFC7627: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477)
TLS v1.3 Key Agreement (ECDH)	KAS-Full	TLS v1.3 ECDH key agreement.	Caveat:No part of the TLS v1.3 protocol, other than the KDF, has been tested by the CAVP and CMVP. Key Strength:Key establishment methodology provides between 128 and 256 bits of encryption strength. Publication:SP 800-56Ar3	TLS v1.3 KDF: (A5478) KAS-ECC-SSC Sp800-56Ar3: (A5477) ECDSA KeyGen (FIPS186-5): (A5477) ECDSA KeyVer (FIPS186-5): (A5477) HMAC-SHA2-256: (A5477) HMAC-SHA2-384: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) Counter DRBG: (A5477)
TLS v1.3 Authentication	DigSig-SigVer	TLS v1.3 signature verification.	Publication:FIPS 186-5	ECDSA SigVer (FIPS186-5): (A5477) RSA SigVer (FIPS186-5): (A5477) SHA2-224: (A5477) SHA2-256: (A5477) SHA2-384: (A5477) SHA2-512: (A5477)
TLS v1.3 Data Encryption/Decryption	BC-Auth BC-UnAuth MAC	TLS v1.3 encryption, decryption, and authentication of TLS session packets, which are used by the TLS Session Key and TLS Authentication Key.	Publication:SP 800-38A, SP 800-38D, FIPS 198-1	AES-GCM: (A5477) Key Length: 128, 256 AES-CBC: (A5477) Key Length: 128, 256 HMAC-SHA2-256: (A5477) HMAC-SHA2-384: (A5477) SHA2-256: (A5477) SHA2-384: (A5477)
TLS v1.3 Key Derivation	KAS-135KDF	TLS v1.3 key derivation.	Caveat:No part of the TLS v1.3 protocol, other than the KDF, has been tested by the CAVP and CMVP. Publication:SP 800-135 Rev. 1	TLS v1.3 KDF: (A5478) HMAC-SHA2-256: (A5477) HMAC-SHA2-384: (A5477) SHA2-256: (A5477) SHA2-384: (A5477)

Table 11: Security Function Implementations

2.7 Algorithm Specific Information

The information below provides algorithm information of references to specifications.

2.7.1 AES-GCM IV

As per *IG* C.H Key/IV Pair Uniqueness Requirements from *SP 800-38D*, AES GCM IV implements scenario 1 only.

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 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.

- For TLS v1.3, the protocol's implementation is contained within the boundary of the module. Per scenario 1 in *FIPS 140-3 IG* C.H, the AES GCM implementation meets the *NIST SP 800-38E* collision probability requirement, as the mechanism for IV generation is compliant with *RFC 8446*. The implementations of AES GCM and all underlying algorithms have been successfully tested for compliance with their respective specifications (see CAVP Cert. A5477). The generated IV is only used in the context of the AES GCM encryption executing the provisions of the TLS 1.3 protocol.

- 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.

2.7.2 RSA

As per *FIPS 186-5 Appendix A.1*, RSA KeyGen implements the method discussed in A.1.3 Generation of Random Primes that are Probably Prime. As per section C.K. of the Implementation Guidance, Infinidat Cryptographic Module (libcrypto) is complaint with FIPS 186-5, as the *FIPS 186-4* CAVP test for RSA SigVer is mathematically identical to the FIPS 186-5 CAVP test.

2.7.3 ECDSA

As per *FIPS 186-5 Appendix A.2*, ECDSA KeyGen implements the method discussed in A.2.2 ECDSA Key Pair Generation by Rejection Sampling.

2.7.4 PBKDF2

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 password/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 $1/62^{20} = 7.044^{-35}$, which is less than 2^{-112}.

As specified in *NIST SP 800-132*, keys derived from passwords/passphrases may only be used in storage applications.

2.7.5 KAS-ECC-SSC

Per *IG* D.F, KAS-ECC-SSC claims Scenario 2 path (1). KAS-ECC-SSC with TLS v1.2 RFC 7627 or TLS v1.3 KDF claims Scenario 2 path (2). The shared secret computation and the key derivation function are CAVP-tested separately.

2.8 RBG and Entropy

The module does not have any entropy certificates.

The calling application provides the following bits of entropy per Counter DRBG mode:

- AES-128-CTR: 128 or 256 bits
- AES-192-CTR: 256, 384, or 512 bits
- AES-256-CTR: 256, 384, or 512 bits

The calling application and its entropy source are outside the module cryptographic boundary. The calling application shall use entropy sources that meet the security strength required for the Counter DRBG as shown in Table 3 of *SP 800-90Arev1*. Counter DRBG uses a derivation function. 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.

A minimum of 256 bits of entropy is required to generate SSPs with up to 256 bits of strength. AES-256-CTR is used by default to generate random values for other security functions.

2.9 Key Generation

The module supports the following key generation methods:

- CKG
- ECDSA KeyGen
- RSA KeyGen

2.10 Key Establishment

2.10.1 Key Agreement Information

The module supports the following key agreement methods:

- KAS-ECC-SSC (Elliptic-curve Diffie-Hellman) and TLS v1.2 KDF RFC7627
- KAS-ECC-SSC (Elliptic-curve Diffie-Hellman) and TLS v1.3 KDF

2.10.2 Key Transport Information

The module supports the following key transport methods:

- Any approved mode of AES with CMAC, GMAC, or HMAC
- AES-CCM
- AES-GCM
- AES-KW
- AES-KWP

2.11 Industry Protocols

The module implements the following industry protocols:

- TLS v1.2
- TLS v1.3

No parts of the TLS protocol, other than the KDF, have been tested by the CAVP or 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

The module does not support a “control output” interface.

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. The table below contains a mapping of the physical and logical interfaces of the module.

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	Status Output	Logical interface is defined as API call return values. This includes status information regarding the module or invoked service/operation.

Table 12: 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 (or set of roles) based on the service selected.

4.2 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:

- 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.

The table below lists details of the supported roles.

Name	Type	Operator Type	Authentication Methods
Crypto Officer	Role	CO	None
User	Role	User	None

Table 13: Roles

4.3 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.

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.

Please refer to the table below for descriptions of available services.

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Show Status	Return Approved mode status.	N/A	API call parameters	Current operational status	None	Crypto Officer
Perform self-tests on-demand	Perform pre-operational and conditional self-tests.	API return value	Re-instantiate module; API call parameters	Status	None	Crypto Officer

Zeroize	Zeroize and de-allocate memory containing sensitive data via OpenSSL_cleanse().	N/A	API call parameters	None	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 - ECDH private component: Z - ECDH public component : Z - Passphrase: Z - TLS pre-master secret: Z - TLS master secret: Z - TLS Session Key: Z - TLS Authentication Key: Z - DRBG entropy input: Z - DRBG seed: Z - DRBG 'V' value: Z - DRBG 'Key' value: Z

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Show versioning information	Return module versioning information.	N/A	API call parameters	Module name, version	None	Crypto Officer
Perform symmetric encryption	Encrypt plaintext data.	EVP_cipher_get_service_indicator()	API call parameters, key, plaintext	Status, ciphertext	AES for Symmetric Encryption/Decryption AES-XTS for Symmetric Encryption/Decryption	User - AES Key: W,E - AES XTS key: W,E
Perform symmetric decryption	Decrypt ciphertext data.	EVP_cipher_get_service_indicator()	API call parameters, key, ciphertext	Status, plaintext	AES for Symmetric Encryption/Decryption AES-XTS for Symmetric Encryption/Decryption TDES for Symmetric Decryption	User - AES Key: W,E - AES XTS key: W,E - TDES key: W,E
Generate symmetric digest	Generate symmetric digest.	For CMAC: CMAC_get_service_indicator() For GMAC: EVP_cipher_get_service_indicator()	API call parameters, key, plaintext	Status, digest	AES-CMAC for Message Authentication AES-GMAC for Message Authentication	User - AES CMAC key: W,E - AES GMAC key: W,E
Verify symmetric digest	Verify symmetric digest.	For CMAC: CMAC_get_service_indicator() For GMAC: EVP_cipher_get_service_indicator()	API call parameters, digest	Status	AES-CMAC for Message Authentication AES-GMAC for Message Authentication TDES-CMAC for Message Authentication	User - AES CMAC key: W,E - AES GMAC key: W,E - TDES CMAC key: W,E
Perform authenticated symmetric encryption	Encrypt plaintext using supplied AES GCM key with an internally generated IV or AES CCM key.	EVP_cipher_get_service_indicator()	API call parameters, key, plaintext	Status, ciphertext	AES-CCM for Authenticated Symmetric Encryption/Decryption AES-GCM for Authenticated Symmetric Encryption/Decryption	User - AES CCM key: W,E - AES GCM key: W,E
Perform authenticated symmetric decryption	Decrypt ciphertext using supplied AES GCM key with an internally generated IV or AES CCM key.	EVP_cipher_get_service_indicator()	API call parameters, key, ciphertext	Status, plaintext	AES-CCM for Authenticated Symmetric Encryption/Decryption AES-GCM for Authenticated Symmetric Encryption/Decryption	User - AES CCM key: W,E - AES GCM key: W,E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Generate random number	Generate 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	Compute 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	Compute a message digest.	EVP_Digest_get_service_indicator()	API call parameters, message	Status, hash	SHA for Message Digest SHA3 for Message Digest SHAKE for Extendable Output Function	User
Generate asymmetric key pair	Generate 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 public key: G,R - ECDSA private key: G,R - RSA public key: G,R - RSA private key: G,R
Verify ECDSA public key	Verify an ECDSA public key.	EC_key_get_service_indicator()	API call parameters, key	Status	ECDSA for Key Verification	User - ECDSA public key: W
Generate digital signature	Generate 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	Verify a digital signature.	EVP_Digest_get_service_indicator()	API call parameters, key, signature, message	Status	DSA for Signature Verification ECDSA for Signature Verification RSA for Signature Verification	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
Perform key wrap	Perform key wrap.	EVP_cipher_get_service_indicator()	API call parameters, encryption key, key	Status, encrypted key	AES+MAC for Key Wrapping/Unwrapping AES-CCM for Key Wrapping/Unwrapping AES-GCM for Key Wrapping/Unwrapping AES-KW/KWP for Key Wrapping/Unwrapping	User - AES Key: W,E - AES CMAC key: W,E - AES GMAC key: W,E - AES CCM key: W,E - AES GCM key: W,E - HMAC key: W,E
Perform key unwrap	Perform key unwrap.	EVP_cipher_get_service_indicator()	API call parameters, decryption key, key	Status, decrypted key	AES+MAC for Key Wrapping/Unwrapping AES-CCM for Key Wrapping/Unwrapping AES-GCM for Key Wrapping/Unwrapping AES-KW/KWP for Key Wrapping/Unwrapping	User - AES Key: W,E - AES CMAC key: W,E - AES GMAC key: W,E - AES CCM key: W,E - AES GCM key: W,E - HMAC key: W,E
Compute shared secret	Compute ECDH shared secret suitable for use as input to a TLS KDF.	EC_key_get_service_indicator()	API call parameters	Status, shared secret	ECDH Shared Secret Computation	User - ECDH public component : W,E - ECDH private component: W,E - TLS pre-master secret: G
Derive keys via TLS KDF	Derive TLS session and integrity keys using TLS v1.2 KDF 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, TLS pre-master secret	Status, TLS keys	TLS v1.2 Key Derivation TLS v1.3 Key Derivation	User - TLS pre-master secret: W,E - TLS master secret: G,E - TLS Session Key: G,R - TLS Authentication Key: G,R

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Perform key agreement functions	Establish the session keys for TLS using ECDH key agreement and TLS v1.2 or v1.3 KDF.	For ECDH: EC_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, ECDH public component, ECDH private component	Status, session keys	TLS v1.2 Key Agreement (ECDH) TLS v1.3 Key Agreement (ECDH)	User - ECDH public component : W,E - ECDH private component: W,E - TLS pre-master secret: G,E - TLS master secret: G,E - TLS Session Key: G,R - TLS Authentication Key: G,R
Establish TLS connection	Establish a TLS session. Please refer to Section 11. Non-Administrator Guidance for information about the approved cipher suites.	For key exchange (ECDHE): EC_key_get_service_indicator( ) For cipher: EVP_cipher_get_service_indicator() For message authentication: HMAC_get_service_indicator() For certificate verification (ECDSA): EC_key_get_service_indicator( ) For certificate verification (RSA): RSA_key_get_service_indicator() For TLS v1.2 KDF: TLSKDF_get_service_indicator( ) For TLS v1.3 KDF: TLS1_3_get_service_indicator()	API call parameters, TLS configuration, TLS keys	Status, TLS connection information	TLS v1.2 Key Agreement (ECDH) TLS v1.2 Authentication TLS v1.2 Data Encryption/Decryption TLS v1.3 Key Agreement (ECDH) TLS v1.3 Authentication TLS v1.3 Data Encryption/Decryption	User - ECDH public component : G,E - ECDH private component: G,E - TLS pre-master secret: G,E - TLS master secret: G,E - TLS Session Key: G,E - TLS Authentication Key: G,E - ECDSA public key: R,E - RSA public key: R,E
Derive key via PBKDF2	Derive key via PBKDF2.	PBKDF_get_service_indicator()	API call parameters, password	Status, key	PBKDF	User - Passphrase: W,E - AES Key: G,R
Derive key via KBKDF	Derive symmetric key from KBKDF.	KBKDF_get_service_indicator()	API call parameters, key	Status, key	KBKDF	User - AES Key: G,R

Table 14: Approved Services

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

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	User
Perform data decryption (non-compliant)	Perform symmetric data decryption.	ARIA Blake2 Blowfish Camellia CAST CAST5 ChaCha20 DES IDEA RC2 RC4 RC5 SEED SM4	User
Perform MAC operations (non-compliant)	Perform message authentication operations.	Poly1305 TDES-CMAC	User
Perform hash operation (non-compliant)	Perform hash operation.	MD2 MD4 MD5 RIPEMD RMD160 SM2 SM3 SM4 Whirlpool	User
Perform digital signature functions (non-compliant)	Perform digital signature functions.	DSA (non-compliant) ECDSA (non-compliant) EdDSA RSA (non-compliant)	User
Perform key agreement functions (non-compliant)	Perform key agreement functions.	DH ECDH (non-compliant)	User
Perform key wrap (non-compliant)	Perform key wrap functions.	TDES TDES-CMAC	User
Perform key encapsulation (non-compliant)	Perform key encapsulation functions.	RSA (non-compliant)	User
Perform key un-encapsulation (non-compliant)	Perform key un-encapsulation functions.	RSA (non-compliant)	User

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

Table 15: Non-Approved Services

4.5 External Software/Firmware Loaded

The module does not load any external software or firmware.

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.

Infinidat Cryptographic Module is not a standalone application; it is a cryptographic toolkit intended for use 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.

5.2 Initiate on Demand

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

6. Operational Environment

6.1 Operational Environment Type and Requirements

The Infinidat Cryptographic Module comprises a software cryptographic library that executes in a Modifiable operational environment. There are no rules, settings, or restrictions for the 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.

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 multi-chip standalone software module and does not include physical security mechanisms. Therefore, per section 7.5 of the *FIPS PUB 140-3 Management Manual* this section is not applicable.

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

The table below lists sensitive security parameters (SSPs) storage areas for this module.

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

Table 16: Storage Areas

9.2 SSP Input-Output Methods

The table below lists SSP input and output methods for this module.

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

Table 17: SSP Input-Output Methods

9.3 SSP Zeroization Methods

The table below lists the SSP zeroization method for the module.

Zeroization Method	Description	Rationale	Operator Initiation
Zeroize service	The module performs the zeroization service when the calling application invokes the OpenSSL_cleanse() function, which zeroizes SSPs. Additionally, other SSPs will be zeroized by an indirect call to OpenSSL_cleanse() via object destruction APIs.	The OpenSSL_cleanse() service zeroizes SSPs instantaneously, which yields the SSPs irretrievable.	The operator calls the OpenSSL_cleanse() function. The successful completion of the procedural zeroization suffices as the implicit indicator that zeroization has completed.

Table 18: SSP Zeroization Methods

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 Tested OE's Physical Perimeter (TOEPP).

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
AES Key	Symmetric encryption, decryption; key transport	Between 128 and 256 bits - Between 128 and 256 bits	Symmetric Key - CSP		AES+MAC for Key Wrapping/Unwrapping AES-CCM for Key Wrapping/Unwrapping AES-GCM for Key Wrapping/Unwrapping AES-KW/KWP for Key Wrapping/Unwrapping KBKDF PBKDF	AES for Symmetric Encryption/Decryption AES+MAC for Key Wrapping/Unwrapping AES-KW/KWP for Key Wrapping/Unwrapping KBKDF
AES CCM key	Authenticated symmetric encryption, decryption; key transport	Between 128 and 256 bits - Between 128 and 256 bits	Symmetric Key - CSP		AES+MAC for Key Wrapping/Unwrapping AES-CCM for Key Wrapping/Unwrapping AES-GCM for Key Wrapping/Unwrapping AES-KW/KWP for Key Wrapping/Unwrapping	AES-CCM for Authenticated Symmetric Encryption/Decryption AES-CCM for Key Wrapping/Unwrapping
AES GCM key	Authenticated symmetric encryption, decryption; key transport	Between 128 and 256 bits - Between 128 and 256 bits	Symmetric Key - CSP		AES+MAC for Key Wrapping/Unwrapping AES-CCM for Key Wrapping/Unwrapping AES-GCM for Key Wrapping/Unwrapping AES-KW/KWP for Key Wrapping/Unwrapping	AES-GCM for Authenticated Symmetric Encryption/Decryption AES-GCM for Key Wrapping/Unwrapping
AES XTS key	Symmetric encryption, decryption	256 or 512 bits - 128 or 256 bits	Symmetric Key - CSP			AES-XTS for Symmetric Encryption/Decryption
AES CMAC key	MAC generation, verification	Between 128 and 256 bits - Between 128 and 256 bits	Authentication - CSP			AES-CMAC for Message Authentication AES+MAC for Key Wrapping/Unwrapping
AES GMAC key	MAC generation, verification	Between 128 and 256 bits - Between 128 and 256 bits	Authentication - CSP			AES-GMAC for Message Authentication AES+MAC for Key Wrapping/Unwrapping
TDES key	Symmetric decryption; key unwrapping	192 bits - 112 bits	Symmetric Key - CSP			TDES for Symmetric Decryption
TDES CMAC key	MAC verification	192 bits - 112 bits	Symmetric Key - CSP			TDES-CMAC for Message Authentication
HMAC key	Keyed hash	Between 160 bits and 512 bits - Between 160 bits and 512 bits	Authentication - CSP			HMAC for Message Authentication AES+MAC for Key Wrapping/Unwrapping

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
DSA public key	Used for DSA digital signature verification.	Between 1024 and 3072 bits - Between 80 and 128 bits	Public - PSP			DSA for Signature Verification
ECDSA private key	Used for ECDSA digital signature generation.	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 ECDSA digital signature verification.	Between 224 and 521 bits - Between 112 and 256 bits	Public - PSP	ECDSA for Key Generation		ECDSA for Signature Verification TLS v1.2 Authentication TLS v1.3 Authentication
RSA private key	Used for RSA digital signature generation.	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 DSA digital signature verification.	Between 2048 and 4096 bits - Between 112 and 150 bits	Public - PSP	RSA for Key Generation		RSA for Signature Verification TLS v1.2 Authentication TLS v1.3 Authentication
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 TLS v1.2 Key Agreement (ECDH) TLS v1.3 Key Agreement (ECDH)
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 TLS v1.2 Key Agreement (ECDH) TLS v1.3 Key Agreement (ECDH)
TLS Session Key	Used for the encryption and decryption of TLS session packets.	128 or 256 bits - 128 or 256 bits	Symmetric Key - CSP		TLS v1.2 Key Derivation TLS v1.3 Key Derivation	TLS v1.2 Data Encryption/Decryption TLS v1.3 Data Encryption/Decryption
TLS Authentication Key	Used for authentication for TLS session packets.	256 or 384 bits - 256 or 384 bits	Authentication - CSP		TLS v1.2 Key Derivation TLS v1.3 Key Derivation	TLS v1.2 Data Encryption/Decryption TLS v1.3 Data Encryption/Decryption
Passphrase	Used as input for PBKDF.	n/a - n/a	Passphrase - CSP			PBKDF

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
TLS pre-master secret	Derivation of the TLS master secret.	Between 224 and 521 bits - Between 224 and 521 bits	Pre-master Secret - CSP		ECDH Shared Secret Computation TLS v1.2 Key Agreement (ECDH) TLS v1.3 Key Agreement (ECDH)	TLS v1.2 Key Derivation TLS v1.3 Key Derivation
TLS master secret	Derivation of the TLS Session Key and TLS Authentication Key used for securing TLS connections.	384 bits - 384 bits	Master Secret - CSP		TLS v1.2 Key Derivation TLS v1.3 Key Derivation	TLS v1.2 Data Encryption/Decryption TLS v1.3 Data Encryption/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 19: SSP Table 1

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
AES Key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	Passphrase:Derived From AES Key:Derived From
AES CCM key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
AES GCM key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
AES XTS key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
AES CMAC key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
AES GMAC key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
TDES key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
TDES CMAC key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
HMAC key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	TLS master secret:Derived From
DSA public key	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
ECDSA private key	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	ECDSA public key:Paired With
ECDSA public key	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	ECDSA private key:Paired With
RSA private key	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	RSA public key:Paired With
RSA public key	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	RSA private key:Paired With
ECDH private component	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	ECDH public component :Paired With
ECDH public component	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	ECDH private component:Paired With
TLS Session Key		RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	TLS master secret:Derived From
TLS Authentication Key		RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	TLS master secret:Derived From
Passphrase	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
TLS pre-master secret	Imported in plaintext via API parameter Exported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
TLS master secret		RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	TLS pre-master secret:Derived From
DRBG entropy input	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
DRBG seed	Imported in plaintext via API parameter	RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
DRBG 'V' value		RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	
DRBG 'Key' value		RAM:Plaintext	Until the OpenSSL_cleanse() function is called.	Zeroize service	

Table 20: SSP Table 2

9.5 Transitions

NIST SP 800-131Arev2 states that "SHA-1 is acceptable for applications that do not require collision resistance." However, SHA-1 is disallowed for digital signature generation and is allowed for legacy use only for digital signature verification.

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 following pre-operational self-tests:

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	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	Test for libssl. Performed automatically without operator action

Table 21: Pre-Operational Self-Tests

10.2 Conditional Self-Tests

The module performs the following conditional self-tests:

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 successful software integrity test
AES-ECB #2	128-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successful software integrity test
AES-CCM #1	192-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Encrypt	After successful software integrity test
AES-CCM #2	192-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successful software integrity test

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
AES-GCM #1	128-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Encrypt	After successful software integrity test
AES-GCM #2	128-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successful software integrity test
AES-XTS Testing Revision 2.0 #1	128-bit; 256-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Encrypt	After successful software integrity test
AES-XTS Testing Revision 2.0 #2	128-bit; 256-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Decrypt	After successful software integrity test
AES-CMAC	CBC mode, 128-bit; 192-bit; 256-bit	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Generate	After successful software integrity test
TDES-ECB	3-key	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Decrypt	After successful software integrity test
TDES-CMAC	CBC mode; 3-Key	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Verify	After successful software integrity test
Counter DRBG	AES, 256-bit, with derivation function	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Instantiate/Generate/Reseed	After successful software integrity test
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 successful software integrity test
ECDSA SigGen (FIPS186-5)	P-224; SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Sign	After successful software integrity test
ECDSA SigVer (FIPS186-5)	P-224; SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Verify	After successful software integrity test

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 successful software integrity test
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 successful software integrity test
HMAC-SHA-1	SHA-1	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hashed Message Authentication	After instantiation and before software integrity test
HMAC-SHA2-224	SHA2-224	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hashed Message Authentication	After instantiation and before software integrity test
HMAC-SHA2-256	SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hashed Message Authentication	After instantiation and before software integrity test
HMAC-SHA2-384	SHA2-384	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hashed Message Authentication	After instantiation and before software integrity test
HMAC-SHA2-512	SHA2-512	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hashed Message Authentication	After instantiation and before software integrity test
KDF SP800-108	HMAC-SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Derive	After successful software integrity test
SHA-1	-	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hash	After instantiation and before software integrity test
SHA2-224	-	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hash	After instantiation and before software integrity test
SHA2-256	-	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hash	After instantiation and before software integrity test

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
SHA2-384	-	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hash	After instantiation and before software integrity test
SHA2-512	-	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hash	After instantiation and before software integrity test
SHA3-256	-	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Hash	After instantiation and before software integrity test
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 “Z” Computation	After successful software integrity test
PBKDF	SHA2-256	KAT	CAST	An integer value indicating success (1) or failure (0) of the self-test procedure call	Derive	After successful software integrity test
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 successful software integrity test
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 successful software integrity test
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 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.
Other (ECDH)	-	PCT	PCT	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Key Generation	When the requested service requires the generation of an ECDH key pair.
AES-XTS Testing Revision 2.0	-	Duplicate Key	Critical Function	An integer value indicating success (1) or failure (0) of the self-test procedure call.	Duplicate Key Test	Executed upon initialization of AES XTS cipher with key data

Table 22: Conditional Self-Tests

10.3 Periodic Self-Test Information

The module does not perform self-tests periodically, but the operator may conduct self-tests on demand via the methods listed in Section 10.5 below. The tables below provide information regarding the pre-operational self-tests and conditional self-tests, respectively, that the operator can perform on demand:

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 23: 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-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-XTS Testing Revision 2.0 #1	KAT	CAST	On Demand	Manually
AES-XTS Testing Revision 2.0 #2	KAT	CAST	On Demand	Manually
AES-CMAC	KAT	CAST	On Demand	Manually
TDES-ECB	KAT	CAST	On Demand	Manually
TDES-CMAC	KAT	CAST	On Demand	Manually
Counter DRBG	KAT	CAST	On Demand	Manually
DSA SigVer (FIPS186-4)	KAT	CAST	On Demand	Manually
ECDSA SigGen (FIPS186-5)	KAT	CAST	On Demand	Manually
ECDSA SigVer (FIPS186-5)	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
KDF SP800-108	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
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
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
ECDSA KeyGen (FIPS186-5)	PCT	PCT	On Demand	Manually
RSA KeyGen (FIPS186-5)	PCT	PCT	On Demand	Manually

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
Other (ECDH)	PCT	PCT	On Demand	Manually
AES-XTS Testing Revision 2.0	Duplicate Key	Critical Function	On Demand	Manually

Table 24: Conditional Periodic Information

10.4 Error States

The table below describes the error states and status indicators of the module.

Name	Description	Conditions	Recovery Method	Indicator
Critical Error	Module immediately terminates the calling application. Module disables access to all cryptographic functions, SSPs, and data output services.	If the module fails pre-operational integrity tests, the SHA/HMAC pre-operational KATs (SHA, HMAC), or the conditional CASTs (DRBG, AES-ECB, AES-CCM, AES-GCM, AES-XTS, AES-CMAC, DSA Verify, ECDSA Sign/Verify, RSA Sign/Verify, KAS-ECC Shared Secret "Z" Computation, KBKDF, PBKDF, TLS v1.2 KDF, TLS v1.3 KDF).	The module must be re-instantiated by the calling application. The module operator should contact Infinidat if errors persist after re-instantiation.	Returns error code and sets an internal flag. Subsequent requests return failure indicator.
Soft Error	The module enters this state upon the failure of a PCT or self-test. The module transitions back to normal operation where the service requiring the self-test can be re-run or a new service can be performed.	If the module fails ECDSA/RSA/ECDH PCTs or the AES-XTS duplicate key test.	Module records the error and resumes normal operation.	Returns error code

Table 25: Error States

10.5 Operator Initiation of Self-Tests

The operator may initiate self-tests on demand by re-instantiating the module or by issuing the `FIPS_selftest()` API command.

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 Startup Procedures

11.1.1 Secure Installation

The module is distributed to the operator as part of the InfuzeOS when procuring the Infinibox product. No specific installation is necessary.

11.1.2 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.

11.1.3 Startup

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

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 Infinidat Customer Support should be contacted.

The following list provides additional guidance for the CO:

- The `fips_post_status()` API can be used to determine the module's operational status. If the module is in a normal operational state, `fips_post_status()` will return 1. If the module is in the Critical Error state, `fips_post_status()` will return 0.

- 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.

- The `OpenSSL_version()` API can be used to obtain the module's versioning information. This information will include the module name and version, which can be correlated with the module's validation record.

- The operator must install all module binaries in the same directory.

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 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.

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

- 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.
    - EVP_Digest_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-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 v1.3

  - TLS_AES_128_GCM_SHA256
  - TLS_AES_256_GCM_SHA384

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

Table 26: Acronyms and Abbreviations

Acronym	Definition
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
ECB	Electronic Code Book
ECC CDH	Elliptic Curve Cryptography Cofactor Diffie-Hellman
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EMI/EMC	Electromagnetic Interference /Electromagnetic Compatibility
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
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
TDES	Triple Data Encryption Standard

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);
int KBKDF_get_service(indicator(EVP_MD *evp_md);
```

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.

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