Infinidat, Ltd.

Infinidat Cryptographic Module

Firmware Version: 1.0.1

FIPS 140-2 Non-Proprietary Security Policy

FIPS Security Level: 1
Document Version: 1.12

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.
13921 Park Center Road, Suite 460
Herndon, VA 20171
United States of America

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

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1 Purpose

This is a non-proprietary Cryptographic Module Security Policy for the Infinidat Cryptographic Module (ICM) from Infinidat, Ltd. This Security Policy describes how ICM meets the security requirements of Federal Information Processing Standards (FIPS) Publication 140-2, which details the U.S.[1] and Canadian government requirements for cryptographic modules. More information about the FIPS 140-2 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 ensure that the module is configured and operating as validated. This policy was prepared as part of the Level 1 FIPS 140-2 validation of the module. The Infinidat Cryptographic Module is referred to in this document as ICM or the module.

1.2 References

This document deals only with operations and capabilities of the module in the technical terms of a FIPS 140-2 cryptographic module security policy. More information is available on the module from the following sources:

- The Infinidat website (http://www.infinidat.com) contains information on the full line of products 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.

1.3 Document Organization

The Security Policy document is one document in a FIPS 140-2 Submission Package. In addition to this document, the Submission Package contains:

- Vendor Evidence
- Finite State Model
- Submission Summary
- Other supporting documentation as additional references

This Security Policy and the other validation submission documentation were produced by Corsec Security, Inc. under contract to Infinidat. With the exception of this Non-Proprietary Security Policy, the FIPS 140-2 Submission Package is proprietary to Infinidat and is releasable only under appropriate non-disclosure agreements. For access to these documents, please contact Infinidat.

[1] U.S. – United States

2. Infinidat Cryptographic Module

2.1 Overview

Infinidat, Ltd.'s enterprise storage solutions are based upon the unique and patented Infinidat Storage Architecture™ (ISA). The Infinidat Storage Architecture is a fully-abstracted set of software-driven storage functions layered on top of very low-cost commodity hardware. The result is multi-petabyte capacity in a single rack, mainframe-class reliability with a 99.99999% uptime, and over 750K IOPS[2] of performance. Automated provisioning, management, and application integration provide a system that is efficient and simple to use. By separating the storage innovation from the hardware, Infinidat allows for the rapid adoption of the latest and most cost-effective hardware. In addition, by shipping the software with a highly-tested hardware reference platform, Infinidat delivers true enterprise-class software-defined storage.

The InfiniBox™ product line includes the F1000, F2000, F4000, and F6000 appliances, each built on a 42 U[3] rack (shown in Figure 1). The F1000 scales up to 115 TB[4] of usable capacity with up to 384 GB of memory. The InfiniBox F2000 is a midrange unified storage array, starting with 250 TB of usable capacity. The InfiniBox F4000 provides additional flexibility and scalability starting at 680 TB of usable capacity. The InfiniBox F6000 provides high-end performance with multi-petabyte capacity. Each model supports various block (SAN[5]) and file (NAS[6]) storage protocols on a single unified platform optimized for virtualization, database storage, and backup and recovery. The solution is managed via a single HTML5[7] GUI[8].

[2] IOPS – Input/Output Operations per Second
[3] U – Rack Unit
[4] TB – Terabyte
[5] SAN – Storage Area Network
[6] NAS – Network Attached Storage
[7] HTML5 – Hypertext Markup Language version 5
[8] GUI – Graphical User Interface.

Figure 1 – Infinidat InfiniBox Enclosure

The Infinidat InfiniGuard™ (including the B1000, B2000, B4000, and B6000 appliances) is a purpose-built backup appliance that offers a highly-available, multi-protocol, data protection solution with all of the performance, capacity, and efficiency to support modern workloads. InfiniGuard is designed with a self-healing architecture and redundant hardware components. Self-encrypting media ensures complete data security. A single InfiniGuard supports up to 1 PB[9] of usable capacity and over 20 PB of effective capacity in a single standard 42 U rack. Configurations start as low as 500 TB with instant capacity on-demand available in increments of 50 TB. IBA is also managed via a single HTML5 GUI.

[9] PB – Petabyte

Figure 2 – Infinidat InfiniGuard Enclosure

At the core of the solution is the InfiniBox node, which is a storage controller appliance (Dell PowerEdge R730xd) running InfiniBox OS, which is a modified version of CentOS 7.4. Each independent node contains a server, DRAM, and Flash cache. The InfiniBox node is pictured in Figure 3 below.

Figure 3 – InfiniBox Node

The Infinidat Cryptographic Module (ICM) is a set of cryptographic libraries that implement TLS[10] v1.2, symmetric key generation and encryption/decryption, and SED[11] authentication key derivation for the InfiniBox OS[12], which is the core component of the InfiniBox B-Series and F-Series appliances. The ICM comes pre-installed on each InfiniBox node.

The module is validated at the FIPS 140-2 Section levels shown in Table 1.

[10] TLS – Transport Layer Security
[11] SED – Self-Encrypting Drive
[12] OS – Operating System

Table 1 – Security Level per FIPS 140-2 Section

Section	Section Title	Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services, and Authentication	1
4	Finite State Model	1
5	Physical Security	1
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC[13]	1
9	Self-tests	1
10	Design Assurance	2
11	Mitigation of Other Attacks	N/A

2.2 Module Specification

The module is a firmware-hybrid module[14] with a multiple-chip standalone embodiment. The overall security level of the module is 1.

The module comprises a set of libraries that provide cryptographic services (via its well-defined APIs[15]) to calling applications running on Infinidat's InfiniBox nodes. The module provides TLS v1.2 protocol support, symmetric key generation, encryption/decryption, and SED authentication key derivation using only FIPS-Approved and allowed algorithms and cryptographic methods.

The physical cryptographic boundary is defined by the physical enclosure of the host platform. The logical cryptographic boundary includes shared library files and integrity check HMAC[16] files, as follows:

- *libcrypto.so.1.0.2k* – Cryptographic library based on CentOS OpenSSL
- *libssl.so.1.0.2k* – TLS v1.2 protocol library based on CentOS OpenSSL
- *libinfinicrypto.so* – Infinidat-developed library that implements key management and cryptographic state management functionality for InfiniBox nodes
- *.libcrypto.so.1.0.2k.hmac* – HMAC digest for *libcrypto* library
  - 0715bfe8cca1d7e5b352780689da577a31c7159017ce7b13b5dc25fd5969d055
- *.libssl.so.1.0.2k.hmac* – HMAC digest for *libssl* library
  - e69a0f7b6fbb3c4abd4ea08a6c98cfcd37dd7d2068480dd436afea04a3a9ed9c
- *.libinfinicrypto.so.hmac* – HMAC digest for *libinfinicrypto* library
  - 943080fbab0e901f71ee291f955f0d9f14499bdc31879bae61a69d55705e8ef5

[13] EMI/EMC – Electromagnetic Interference / Electromagnetic Compatibility
[14] The module relies on the AES-NI instruction set provided by the host server's Intel processor for accelerating AES operations.
[15] API – Application Programming Interface
[16] HMAC – (keyed-) Hashed Message Authentication Code

The module was tested and found compliant on the InfiniBox node hardware appliance (Dell PowerEdge R730xd) with an Intel Xeon E5-2697 processor running InfiniBox OS version 4.7.

The module implements the FIPS-Approved cryptographic algorithms listed in Table 2.

Table 2 – FIPS-Approved Algorithm Implementations[17]

CAVP[18] Cert	Algorithm	Standard	Mode/Method	Key Lengths, Curves, or Moduli	Use
5414	AES	FIPS PUB 197, NIST SP[19] 800-38A	CBC[20], ECB[21]	128, 192, 256	Data encryption/decryption
		FIPS PUB 197, NIST SP 800-38D	GCM[22]	128, 192, 256	Data encryption/decryption and authentication *Conforms to IG A.5 Scenario 1 (TLS)*
Vendor Affirmation	CKG[23]	NIST SP 800-133	-	-	Key generation
1866	CVL[24]	NIST SP 800-56Arev2	ECC CDH[25] Primitive	P-256, P-384, P-521	Shared secret computation
1868	CVL	NIST SP 800-135rev1	TLSv1.2	-	Key derivation function *No parts of the TLS protocol, other than the KDF[26], have been tested by the CAVP and CMVP.*
2109	DRBG[27]	NIST SP 800-90A	CTR[28]	128, 192, 256	Deterministic random bit generation *With and without derivation function*
1392	DSA[29]	FIPS PUB 186-4	KeyGen, SigGen, SigVer	2048, 3072-bit key sizes with SHA-224, SHA-256, SHA-384, SHA-512	Asymmetric key generation, digital signature generation, digital signature verification

[17] It is the user's responsibility to determine that the algorithms and key lengths utilized by the module are compliant with the requirements of *NIST SP 800-131A Rev1*. Refer to https://csrc.nist.gov/publications/detail/sp/800-131a/rev-1/final for additional information.
[18] CAVP – Cryptographic Algorithm Validation Program
[19] SP – Special Publication
[20] CBC – Cipher block chaining
[21] ECB – Electronic Codebook
[22] GCM – Galois Counter Mode
[23] CKG – Cryptographic Key Generation
[24] CVL – Component Validation List
[25] ECC CDH– Elliptic Curve Cryptography Cofactor Diffie-Hellman
[26] KDF – Key Derivation Function
[27] DRBG – Deterministic Random Bit Generator
[28] CTR – Counter
[29] DSA – Digital Signature Algorithm

CAVP[18] Cert	Algorithm	Standard	Mode/Method	Key Lengths, Curves, or Moduli	Use
1434	ECDSA[30]	FIPS PUB 186-4	KeyGen, SigGen, SigVer	P-256, P-384, P-521	Asymmetric key generation, digital signature generation, digital signature verification
3585	HMAC	FIPS PUB 198-1	SHA[31]-1, SHA-224, SHA-256, SHA-384, SHA-512	-	Message authentication
222	KBKDF[32]	NIST SP 800-108	Counter Mode with HMAC-SHA-256 and HMAC-SHA-512	256, 512	Key derivation function
2893	RSA[33]	FIPS PUB 186-4	KeyGen, SigGenPKCS[34]1.5, SigVerPKCS1.5	2048, 3072-bit key sizes with SHA-224, SHA-256, SHA-384, SHA-512	Asymmetric key generation, digital signature generation, digital signature verification
4344	SHS[35]	FIPS PUB 180-4	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	-	Message digest

The vendor affirms the following cryptographic security method:

- As per *NIST SP 800-133*, the module uses its FIPS-Approved counter-based DRBG to generate cryptographic keys. The resulting symmetric key or generated seed is an unmodified output from the DRBG. The module's DRBG is seeded via `/dev/random`, a non-deterministic random number generator (NDRNG) internal to the module.

The module implements the non-Approved but allowed algorithms shown in Table 3.

Table 3 – Allowed Algorithm Implementations

Algorithm	Caveat	Use
Diffie-Hellman (DH)	-	Shared secret computation *The module does not provide a full key agreement scheme. Rather, it provides the primitive and an Approved KDF (CVL Cert. #1868) in support of Diffie-Hellman functionality. Per FIPS Implementation Guidance D.8, such implementations are allowed in the Approved mode.*

[30] ECDSA – Elliptic Curve Digital Signature Algorithm
[31] SHA – Secure Hash Algorithm
[32] KBKDF – Key-based Key Derivation Function
[33] RSA – Rivest Shamir Adleman
[34] PKCS – Public Key Cryptography Standard
[35] SHS – Secure Hash Standard

Algorithm	Caveat	Use
NDRNG	-	The module's DRBG is seeded by a GET request to `/dev/random`, a non-deterministic random number generator (NDRNG) outside the module's logical boundary. *Each GET request returns a minimum of 256 bits of entropy.*
RSA	Key establishment methodology provides 112 or 128 bits of encryption strength	Key wrapping

As a firmware-hybrid module, the ICM has both a logical cryptographic boundary and a physical cryptographic boundary. The physical and logical boundaries are described in sections 2.2.1 and 2.2.2, respectively.

2.2.1 Physical Cryptographic Boundary

As a firmware-hybrid cryptographic module, the physical boundary of the cryptographic module is defined by the enclosure around the host server on which it runs.

The physical block diagram and cryptographic boundary for the module is depicted in Figure 4.

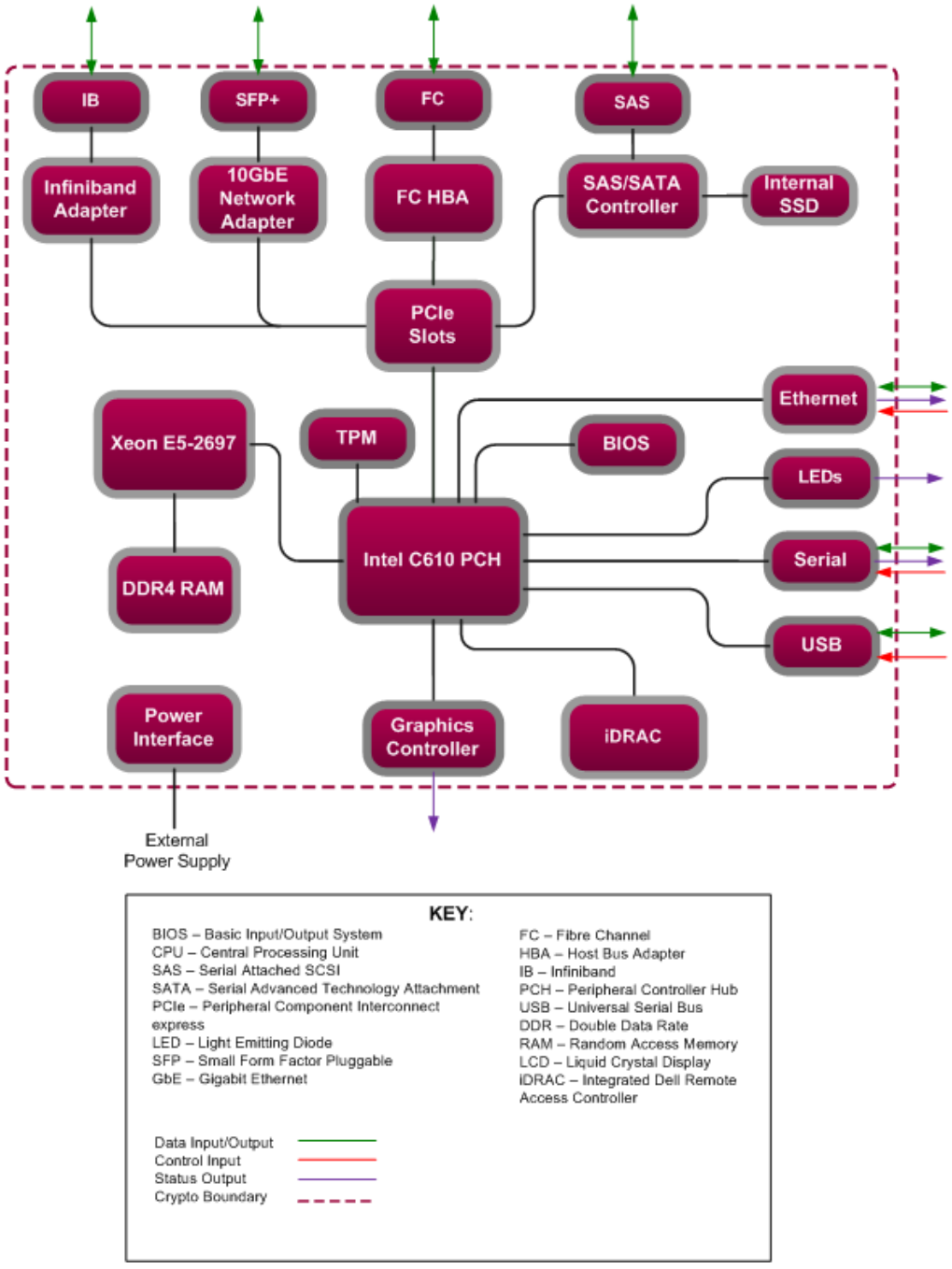

Figure 4 – InfiniBox Node Physical Block Diagram

2.2.2 Logical Cryptographic Boundary

The logical cryptographic boundary surrounds only the *libcrypto, libssl* and *libinfinicrypto* libraries (as well as their associated HMAC files). The cryptographic module is used by the calling applications (ICM Agent process, Apache Web Server, Management Service) to provide symmetric and asymmetric cipher operation, signature generation and verification, hashing, cryptographic key generation, random number generation, message authentication

functions, key agreement/key exchange protocols, and key derivation functions. The module is entirely contained within the physical cryptographic boundary described in section 2.2.1.

The logical block diagram and cryptographic boundary is depicted in Figure 5.

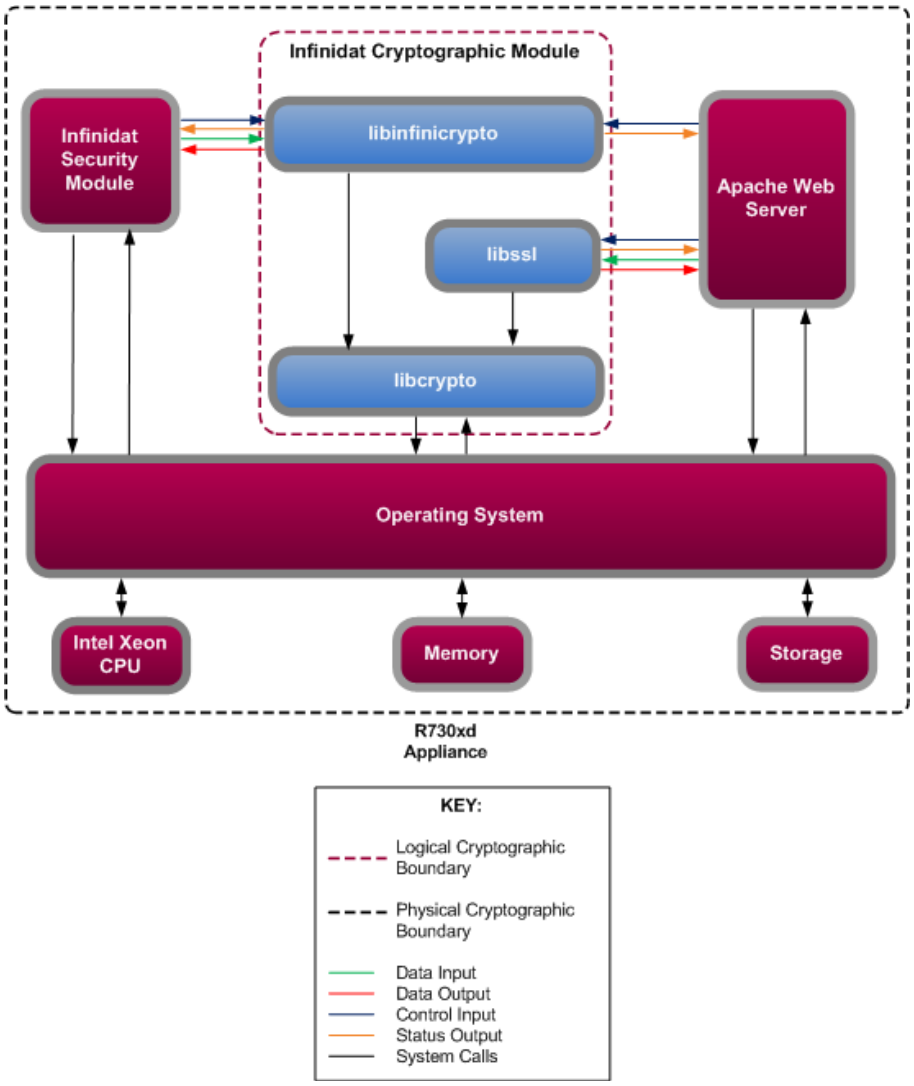

Figure 5 – ICM Logical Diagram and Cryptographic Boundary

2.3 Module Interfaces

The module isolates communications to logical interfaces that are defined in the firmware as an API. The API interface is mapped to the following four logical interfaces:

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

The module's physical boundary features the physical ports of a host server. The module's manual controls; physical indicators; and physical, logical, and electrical characteristics are those of the host server. The module's logical interfaces are at a lower level in the firmware. Data and control input through physical interfaces is translated into the logical data and control inputs for the module. A mapping of the FIPS 140-2 logical interfaces, the physical interfaces, and the module interfaces can be found in Table 4.

Table 4 – FIPS 140-2 Logical Interface Mappings

FIPS 140-2 Interface	Physical Interface	Logical Interface
Data Input	Ethernet port, serial port, USB[36] port, IB[37] port, SFP+[38] port, FC[39] port, SAS[40] port	Arguments for an API call that provide the data to be used or processed by the module
Data Output	Ethernet port, serial port, USB port, IB port, SFP+ port, FC port, SAS port	Arguments for an API call that specify where the result of the function is stored
Control Input	Ethernet port, serial port, USB port, power button	Arguments for an API call that are used to control the operation of the module
Status Output	Ethernet port, serial port, VGA[41] port, LEDs[42]	Return values from an API call
Power Input	AC Power socket	-

2.4 Roles and Services

The following sections detail the roles and services provided by the module.

2.4.1 Roles

There are two roles in the module that operators may assume: a Cryptographic Officer (CO) role, and a User role. When requesting a given service, module operators implicitly assume the role of both CO and User.

2.4.2 Services

Descriptions of the services available to each role as well as CSP access are detailed in Table 5 below. Please note that the keys and CSPs listed in the table indicate the type of access required and that the following notations are used:

- R – Read: The CSP is read.
- W – Write: The CSP is established, generated, or modified.

[36] USB – Universal Serial Bus
[37] IB – Infiniband
[38] SFP – Small Form Factor Pluggable
[39] FC – Fibre Channel
[40] SAS – Serial Attached SCSI
[41] VGA – Video Graphics Array
[42] LED – Light Emitting Diode

- X – Execute: The CSP is used within an Approved or Allowed security function or authentication mechanism.
- D – Delete: The CSP is deleted or zeroized.

Table 5 – Mapping of Operator Services to Inputs, Outputs, CSPs, and Type of Access

Service	Operator		Description	Input	Output	CSP and Type of Access
	CO	User				
Initialize	✓	✓	Perform initialization of the module	API call parameters	Status	None
Run self-test on demand	✓	✓	Performs power-up self-tests	Power cycle; restart process	Status	None
Show status[43]	✓	✓	Returns the current version of the module as well as the operating status	None	Status	None
Zeroize	✓	✓	Zeroizes and de-allocates memory containing sensitive data	Power cycle; unload module	None	All keys and CSPs – D
Generate random number	✓	✓	Returns the specified number of random bits to the calling application	API call parameters	Status, random bits	DRBG Seed – RWX DRBG Key value – RWX DRBG Entropy Input String – RX DRBG 'V' value – RWX
Generate message digest	✓	✓	Compute and return a message digest using SHS algorithms	API call parameters, message	Status, hash	None
Generate HMAC key	✓	✓	Generate and return the specified type of symmetric key (HMAC)	API call parameters	Status, key	DRBG Seed – RWX DRBG Key value – RWX DRBG Entropy Input String – RX DRBG 'V' value – RWX HMAC key – RW
Generate keyed hash (HMAC)	✓	✓	Compute and return a message authentication code	API call parameters, key, message	Status, hash	HMAC key – RX
Generate symmetric key	✓	✓	Generate and return the specified type of symmetric key (AES)	API call parameters	Status, key	DRBG Seed – RWX DRBG Key value – RWX DRBG Entropy Input String – RX DRBG 'V' value – RWX AES key – RW AES-GCM key – RW AES-GCM IV – RWX AES-XTS key – RW
Perform symmetric encryption	✓	✓	Encrypt plaintext using supplied key and algorithm specification (AES)	API call parameters, key, plaintext	Status, ciphertext	AES key – RX AES-GCM key – RX

[43] Running the 'show status' command on the module will, along with its operating status, return the version of the module. The value returned by the module will be version X.X. This version is equivalent to version Y.Y of the <Product Name Long>.

Service	Operator		Description	Input	Output	CSP and Type of Access
	CO	User				
Perform symmetric decryption	✓	✓	Decrypt ciphertext using supplied key and algorithm specification (AES)	API call parameters, key, ciphertext	Status, plaintext	AES key – RX AES-GCM key – RX
Generate asymmetric key pair	✓	✓	Generate and return the specified type of asymmetric key pair (RSA, DSA, or ECDSA)	API call parameters	Status, key pair	DRBG Seed – RWX DRBG Key value – RWX DRBG Entropy Input String – RX DRBG 'V' value – RWX RSA public key – RW RSA private key – RW DSA public key – RW DSA private key – RW ECDSA public key – RW ECDSA private key – RW
Establish TLS session	✓	✓	Establish a TLS session	API call parameter	Command response/ Status output	ECDSA private key – RX RSA private key – RX AES key – RX AES-GCM key – RX AES-GCM IV – RX HMAC key – RX
Generate signature	✓	✓	Generate a signature for the supplied message using the specified key and algorithm (RSA, DSA, or ECDSA)	API call parameters, key, message	Status, signature	RSA private key – RX DSA private key – RX ECDSA private key – RX
Verify signature	✓	✓	Verify the signature on the supplied message using the specified key and algorithm (RSA, DSA, or ECDSA)	API call parameters, key, signature, message	Status	RSA public key – RX DSA public key – RX ECDSA public key – RX
Generate KBKDF seed	✓	✓	Generate a seed for the SP 800-108 KBKDF using DRBG output	API call parameters, DRBG output	Status, seed	DRBG Seed – RWX DRBG Key value – RWX DRBG Entropy Input String – RX DRBG 'V' value – RWX KBKDF seed – RW
Key-based key derivation	✓	✓	Key-based key derivation function for deriving SED authentication keys, SSD authentication keys, and XTS keys.	API call, parameters, key	Status, key	KBKDF seed – RX KBKDF output – RW
Generate shared secret	✓	✓	Used to calculate an FFC[44] DH or ECC CDH shared secret for deriving TLS session keys	API call parameters	Status, shared secret	FFC DH public key – RX FFC DH private key – RX FFC DH shared secret – W ECC CDH public key – RX ECC CDH private key – RX ECC CDH shared secret – W

[44] FFC – Finite Field Cryptography

Service	Operator		Description	Input	Output	CSP and Type of Access
	CO	User				
Derive TLS session keys	✓	✓	Used to derive keys for securing a TLS session	API call parameters	Status, TLS keys	FFC DH shared secret – RX ECC CDH shared secret – RX Master secret – WX AES key – W AES-GCM key – W AES-GCM IV – W HMAC key – W

2.5 Physical Security

The hardware portion of the module is installed within a production grade appliance with standard integrated circuits, uniform exterior metal chassis, and standard connectors. The internal circuitry is micro-coated using industry-standard passivation techniques.

2.6 Operational Environment

The module was tested and found to be compliant with FIPS 140-2 requirements on the following operational environment:

- Dell PowerEdge R730xd with an Intel Xeon E5-2697 processor running InfiniBox OS 4.7

2.7 Cryptographic Key Management

The module supports the CSPs listed in Table 6.

Table 6 – Cryptographic Keys, Cryptographic Key Components, and CSPs

CSP	CSP Type	Generation / Input	Output	Storage	Zeroization	Use
AES key	128, 192, 256-bit key	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application OR Derived according to SP 800-135rev1 KDF	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Encryption, decryption
AES GCM key	128, 192, 256-bit key	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application OR Derived according to SP 800-135rev1 KDF	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Encryption, decryption

CSP	CSP Type	Generation / Input	Output	Storage	Zeroization	Use
AES GCM IV	128-bit value	Internally generated deterministically in compliance with TLSv1.2 GCM cipher suites as specified in RFC 5288 and Section 8.2.1 of NIST SP 800-38D. When the nonce_explicit part of the IV exhausts the maximum number of possible values for a given session key, the module will trigger a handshake to establish a new encryption key according to RFC 5246.	Never	Plaintext in volatile memory	Unload module, API call, Remove power	Initialization vector for AES-GCM
AES XTS key	256 or 512-bit key	Derived according to SP 800-108 KBKDF	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Encryption, decryption
HMAC key	160, 224, 256, 384, or 512-bit key	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application OR Derived according to SP 800-135rev1 KDF	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Message authentication with SHS
RSA private key	2048 or 3072-bit key	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Signature generation, decryption

CSP	CSP Type	Generation / Input	Output	Storage	Zeroization	Use
RSA public key	2048 or 3072-bit key	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Signature verification, encryption
DSA private key	224 or 256-bit key	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Signature generation
DSA public key	2048 or 3072-bit key	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Signature verification
ECDSA private key	NIST curves P-256, P-384, P-521	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Signature generation

CSP	CSP Type	Generation / Input	Output	Storage	Zeroization	Use
ECDSA public key	NIST curves P-256, P-384, P-521	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Signature verification
ECC CDH private key	NIST curves P-256, P-384, P-521	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Used by host application to support ECC CDH key agreement
ECC CDH public key	NIST curves P-256, P-384, P-521	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Used by host application to support ECC CDH key agreement
ECC CDH shared secret	EC DH shared secret	Established internally via ECC DDH shared secret computation OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Zeroized at service completion, Remove power	Used for deriving master secret for TLS (for ECDH-based cipher suites)
Master secret	384-bit master secret	Derived internally via TLS KDF using the ECC CDH shared secret or FFC DH shared secret	Never	Keys are not persistently stored by the module	Unload module, API call, Remove power	Used for deriving AES and HMAC keys used in TLS by the host application

CSP	CSP Type	Generation / Input	Output	Storage	Zeroization	Use
FFC DH private key	224 or 256-bit key	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Used by host application to support FFC DH key agreement
FFC DH public key	2048 or 3072-bit key	Internally generated via SP 800-133 CKG OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Used by host application to support FFC DH key agreement
FFC DH shared secret	FFC DH shared secret	Established internally via non-compliant FFC DH shared secret computation OR Electronically input in plaintext from calling application	Output in plaintext	Keys are not persistently stored by the module	Zeroized at service completion, Remove power	Used for deriving master secret for TLS (for DH-based cipher suites)
KBKDF Seed	256-bit key	Internally generated via SP 800-133 CKG	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Used for deriving SED authentication keys, SSD authentication keys, and XTS encryption keys
KBKDF Output	256-bit key	Derived using SP 800-108 KBKDF using KBKDF Seed as input	Output in plaintext	Keys are not persistently stored by the module	Unload module, API call, Remove power	Used for SED and SSD authentication and XTS encryption Note: The cryptographic operations that utilize these keys are not performed by the crypto module.

CSP	CSP Type	Generation / Input	Output	Storage	Zeroization	Use
DRBG Seed	Random data – 384 bits	Generated internally using nonce along with DRBG entropy input.	Never	Keys are not persistently stored by the module	Unload module, API call, Remove power	Seeding material for CTR_DRBG
DRBG Entropy Input String	256-bit value	Externally generated and electronically input in plaintext from calling application	Never	Plaintext in volatile memory	Unload module, API call, Remove power	Entropy material for CTR_DRBG
DRBG 'V' Value	Internal state value	Internally generated	Never	Plaintext in volatile memory	Unload module, API call, Remove power	Used for CTR_DRBG
DRBG 'Key' Value	Internal state value	Internally generated	Never	Plaintext in volatile memory	Unload module, API call, Remove power	Used for CTR_DRBG

2.8 EMI / EMC

The module was tested and found conformant to the EMI/EMC requirements specified by 47 Code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators, Digital Devices, Class A (business use).

2.9 Self-Tests

The module performs power-up self-tests, conditional self-tests, and critical function tests. These tests are described in the sections that follow. Power-up self-tests are performed automatically after power is applied; no further intervention is required from the operator. Conditional tests are performed when conditions require. Data output and cryptographic operations are inhibited until the module has successfully passed all the power-up self-tests.

2.9.1 Power-Up Self-Tests

The module executes the power-up self-tests automatically without operator intervention upon invocation of the module. While the self-tests are executing, all cryptographic operations and data output are inhibited.

The following self-tests are performed at power-up to verify the integrity of the module firmware and the correct operation of the FIPS-Approved algorithm implementations.

Once all power-up self-tests have completed, the module will report a status of `ICM_STATUS_OK`.

- Firmware integrity check (using HMAC SHA-256)
- Algorithm implementation tests
    - AES-ECB encrypt KAT[45]
    - AES-ECB decrypt KAT
    - AES-GCM encrypt KAT
    - AES-GCM decrypt KAT
    - RSA sign/verify KAT
    - DSA PCT[46] for sign/verify
    - ECDSA PCT for sign/verify
    - ECC CDH Primitive 'Z' KAT
    - FFC DH Primitive 'Z' KAT
    - CTR_DRBG KAT
    - HMAC SHA-1 KAT
    - HMAC-SHA-224 KAT
    - HMAC SHA-256 KAT
    - HMAC SHA-384 KAT
    - HMAC SHA-512 KAT
    - SHA-1 KAT
    - SHA-256 KAT
    - SHA-512 KAT

[45] KAT – Known Answer Test
[46] PCT – Pairwise Consistency Test

Note: HMAC KATs with SHA-224 and SHA-384 utilize (and thus test) the full functionality of the SHA-224 and SHA-384 algorithms; therefore, no independent KAT for SHA-224 and SHA-384 is required.

2.9.2 Conditional Self-Tests

The module performs the following conditional self-tests:

- Continuous RNG test on the NDRNG
- Continuous RNG test on the CTR_DRBG
- RSA PCT for sign/verify
- RSA PCT for encrypt/decrypt
- DSA PCT for sign/verify
- ECDSA PCT for sign/verify
- DRBG instantiate health test
- DRBG generate health test
- DRBG reseed health test

2.9.3 Critical Function Tests

The module performs an XTS key generation test to ensure that Key 1 ≠ Key 2 when keys are generated for use by the calling application.

2.9.4 Self-Test Failure Handling

If any of the self-tests fail (with the exception of the critical function tests), the module enters a critical error state and returns `ICM_STATUS_CRITICAL_FAILURE`. While in an error state, the module inhibits all cryptographic functions and data output. When the calling application detects that the module is in an error state, it will automatically exit and restart the process, which will trigger the re-execution of the power-up self-tests.

The error condition is considered to have been cleared if the module successfully passes all of the subsequent power-up self-tests. If the module continues to fail subsequent power-up self-tests, the module is considered to be malfunctioning or compromised, and the CO shall contact Infinidat Customer Support for repair or replacement.

The module may also return `ICM_STATUS_NON_CRITICAL_FAILURE` which will not cause the module to enter a critical error state. This error code is reported whenever a critical function test fails. In the cause of critical function test failure, the module will return to a normal operational state, and the calling application will attempt to correct the condition, which may result in a restart of the process. A return code of `ICM_STATUS_NON_CRITICAL_FAILURE` may also be reported if the module receives improperly formed input (for example, unexpected input to the key derivation function). In this case, the module will also return to a normal state where the calling application may re-attempt the operation.

2.10 Mitigation of Other Attacks

This section is not applicable. The modules do not claim to mitigate any attacks beyond the FIPS 140-2 Level 1 requirements for this validation.

3. Secure Operation

The module meets Level 1 requirements for FIPS 140-2. The sections below describe how to ensure the module is running in a secure and validated fashion.

3.1 Installation and Setup

All InfiniBox nodes are delivered with the OS and all necessary firmware pre-loaded, including the cryptographic libraries comprising the module. All calling applications that use the module's services are configured to pre-load the module as part of the installation.

3.2 Initialization

No end-user action is required to initialize the module into FIPS mode.

This module is designed to support Infinidat applications running in Infinidat-controlled operational environments, and the sole consumers of the cryptographic services provided by the module are Infinidat applications. All configuration actions are performed by Infinidat personnel prior to delivery to the end-user, and the calling applications perform all initialization actions required to place the module into FIPS mode. The initialization actions are performed automatically, without end-user intervention, and 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.

3.2.1 Verification

To verify whether the module is operating in its validated configuration, the calling application shall check that the API `icm_get_security_mode()` returns a value of "2" (`ICM_SM_FIPS1`) .

The module reports its status to the calling application via the `icm_get_failed_status()` API function. In addition, all API functions will return a status that indicates success or failure of the requested API function. The module will either return `ICM_STATUS_OK` or `ICM_STATUS_CRITICAL_FAILURE` if the request is successful or failed, respectively.

3.3 Operator Guidance

The following sections provide guidance to module operators for the correct and secure operation of the module.

3.3.1 Crypto Officer Guidance

Infinidat's calling applications assume the role of Crypto Officer. However, from an end-user perspective, no specific management activities are required to ensure that the module runs securely; once operational, the module only executes in its validated manner. However, if any irregular activity is observed or the module is consistently reporting errors, then Infinidat Customer Support should be contacted.

3.3.2 User Guidance

Infinidat's calling applications assume the role of User. However, although the end-user does not have any ability to modify the configuration of the module, they should notify Infinidat Customer Support if any irregular activity is observed.

3.3.3 General Operator Guidance

The following provide further guidance for the general operation of the module:

- The module does not store any CSPs persistently (that is, beyond the lifetime of an invoked API call), with the exception of DRBG state values used for the module's default key generation service. All module services automatically overwrite CSPs temporarily stored in allocated memory with zeros when released by the invoked API call. The previously-allocated memory is then freed for reuse.

  Zeroization of CSPs in allocated memory can be performed on-demand by unloading the module from memory or power-cycling the module's host appliance.

- Power-up self-tests may be performed on-demand by restarting the process or power-cycling the module's host appliance.

- To determine the module's operational status, the `icm_get_security_mode()` API function must be used. A return value of "2" (`ICM_SM_FIPS1`) indicates that the module is properly configured. Any other return indicates that the module is not running in its validated configuration; in this case, the CO must contact Infinidat Customer Support for assistance.

- While in the Approved mode, root access to the InfiniBox OS (and any other access allowing modifications or replacement to the module binary) is not permitted.

- The `icm_self_test()` API is used only during the pre-loading of the module binaries and shall not be called during normal operation by any calling application.

3.4 Additional Guidance and Usage Policies

The notes below provide additional guidance and policies that module operators must follow:

- The CO shall power-cycle the module if the module has encountered a critical error and becomes non-operational. If power cycling the module does not correct the error condition, the module is considered to be compromised or malfunctioning, and to the CO shall contact Infinidat Customer Support for repair or replacement.

- As a firmware cryptographic library, the module's services are intended 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 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 calling application shall use entropy sources that meet the security strength required for the DRBG as shown in Table 3 of *NIST SP 800-90A.*

- The module requests entropy through a `GET` command. Each GET request returns a minimum of 256 bits of entropy. Responses to requests for entropy are blocked by the entropy mechanism until there is sufficient entropy to satisfy the request. If, during power-up self-tests, the minimum entropy strength cannot be met, the module will fail with exit code 51 and return the message "`ICM_EXIT_FAILED_SSL_DRNG_INIT`". The module will then enter a critical error state, causing the calling application to shut down. If rebooting the host server does not result in the successful execution of power-up self-tests, then the module will not be able to operate in its validated configuration. The CO must contact Infinidat Customer Support for assistance.

- As the module does not persistently store keys, the calling application is responsible for the storage and zeroization of keys and CSPs passed into and out of the module.

- IVs for AES-GCM are generated deterministically in accordance with *NIST SP 800-38D* Section 8.2.1. If 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.

3.5 Non-Approved Mode

When operating in its validated configuration (as verified by the guidance provided in section 3.2.1 above), the module does not support a non-Approved mode of operation.

4. Acronyms

Table 7 provides definitions for the acronyms used in this document.

Table 7 – Acronyms

Acronym	Definition
AES	Advanced Encryption Standard
API	Application Programming Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCCS	Canadian Centre for Cyber Security
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
DDR	Double Data Rate
DH	Diffie-Hellman
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FC	Fibre Channel
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standard
GbE	Gigabit Ethernet
GCM	Galois/Counter Mode
GUI	Graphical User Interface
HMAC	(keyed-) Hash Message Authentication Code

HTML5	HyperText Markup Language 5
IB	Infiniband
ICM	Infinidat Crytographic Module
iDRAC	Integrated Dell Remote Access Controller
ID	Identifier
ICM	Infinidat Cryptographic Module
IOPS	Input/Output Operations per Second
ISA	Infinidat Storage Architecture™
IV	Initialization Vector
KAT	Known Answer Test
KBKDF	Key-based KDF
KDF	Key Derivation Function
LCD	Liquid Crystal Display
LED	Light Emitting Diode
NAS	Network Attached Storage
NDRNG	Non-deterministic RNG
NIST	National Institute of Standards and Technology
OS	Operating System
PB	Petabyte
PCH	Peripheral Controller Hub
PCIe	Peripheral Component Interconnect express
PCT	Pairwise Consistency Test
PKCS	Public Key Cryptography Standard
PUB	Publication
RAM	Random Access Memory
RSA	Rivest, Shamir, Adleman
RNG	Random Number Generator
SAN	Storage Area Network
SAS	Serial Attached SCSI
SED	Self-Encrypting Drive
SFP	Small Form Factor Pluggable
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SNMP	Simple Network Management Protocol
SP	Special Publication
SSD	Solid State Drive
TB	Terabyte

TLS	Transport Layer Security
U	Rack Unit
USB	Universal Serial Bus
XEX	XOR-Encrypt-XOR
XOR	Exclusive Or
XTS	XEX-Based Tweaked-Codebook Mode with Ciphertext Stealing

Prepared by:

Corsec Security, Inc.

13921 Park Center Road, Suite 460
Herndon, VA 20171
United States of America

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