Masimo Corporation

Masimo Cryptographic Module

Software Version: 1.0

FIPS 140-3 Non-Proprietary Security Policy

FIPS Security Level: 1
Document Version: 0.2

Prepared for:

Masimo Corporation
52 Discovery
Irvine, CA 92618
United States of America

Phone: +1 800 326 4890
www.masimo.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

Abstract

This is a non-proprietary Cryptographic Module Security Policy for the Masimo Cryptographic Module (software version: 1.0) from Masimo Corporation (Masimo). This Security Policy describes how the Masimo Cryptographic Module meets the security requirements of Federal Information Processing Standards (FIPS) Publication 140-3, which details the U.S. and Canadian government requirements for cryptographic modules. More information about the FIPS 140-3 standard and validation program is available on the Cryptographic Module Validation Program (CMVP) website, which is maintained by the National Institute of Standards and Technology (NIST) and the Canadian Centre for Cyber Security (CCCS).

This document also describes how to run the module in an Approved mode of operation. This policy was prepared as part of the Level 1 FIPS 140-3 validation of the module. The Masimo Cryptographic Module is referred to in this document as Masimo Crypto Module or the module.

References

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

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

Document Organization

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

Table of Contents

List of Tables

List of Figures

1. General

Masimo Corporation is a global medical technology company that develops and produces a wide array of industry-leading monitoring technologies, including innovative measurements, sensors, patient monitors, and automation and connectivity solutions. Our mission is to improve patient outcomes and reduce the cost of care.

Masimo's Root® Patient Monitoring and Connectivity Platform was built from the ground up to be as flexible and expandable as possible to facilitate the addition of other Masimo and third-party monitoring technologies. When connected to Masimo's Radical-7 Pulse CO-Oximeter®, Root provides continuous monitoring using industry-leading Masimo SET® Measure-through Motion and Low Perfusion™ pulse oximetry. In addition, the platform can be upgraded to provide Masimo rainbow SET® technology, allowing clinicians to non-invasively monitor multiple additional physiologic parameters.

The Masimo Cryptographic Module v1.0 is a software library providing a C language API[1] for use by Masimo products requiring cryptographic functionality. The Masimo Cryptographic Module v1.0 includes symmetric encryption/decryption, digital signature generation/verification, hashing, cryptographic key generation, random number generation, message authentication, and SSP establishment functions to secure data-at-rest/data-in-flight and offers cryptographic support for secure communications protocols (including TLS[2] 1.2/1.3).

The Masimo Cryptographic Module is validated at the FIPS 140-3 section levels shown in Table 1.

Table 1 – Security Levels

ISO/IEC 24579 Section 6. [Number Below]	FIPS 140-3 Section Title	Security Level
1	General	1
2	Cryptographic Module Specification	1
3	Cryptographic Module Interfaces	1
4	Roles, Services, and Authentication	1
5	Software/Firmware Security	1
6	Operational Environment	1
7	Physical Security	N/A
8	Non-Invasive Security	N/A
9	Sensitive Security Parameter Management	1
10	Self-tests	1
11	Life-Cycle Assurance	1
12	Mitigation of Other Attacks	N/A

The module has an overall security level of 1.

[1] API – Application Programming Interface
[2] TLS – Transport Layer Security

2. Cryptographic Module Specification

The Masimo Cryptographic Module v1.0 is a software module with a multi-chip standalone embodiment. The module is designed to operate within a modifiable operational environment.

2.1 Operational Environments

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

Table 2 – Tested Operational Environments

#	Operating System	Hardware Platform	Processor	PAA/Acceleration
1	Custom Linux OS with Linux kernel 2.6.38	Masimo Radical-7	ARM Cortex-A8 (ARMv7-A)	Without PAA
2	Custom Linux OS with Linux kernel 4.9.43	Masimo Root	ARM Cortex-A8 (ARMv7-A)	Without PAA

The vendor affirms the module's continued validation compliance when operating on the environments listed in Table 3.

Table 3 – Vendor-Affirmed Operational Environments

#	Operating System	Hardware Platform
1	Red Hat Enterprise Linux 8	Masimo Patient SafetyNet
2	Red Hat Enterprise Linux 8	Masimo Iris Gateway
3	Windows 10 Pro on VMware Workstation 15.x	Masimo Patient SafetyNet View Station
4	Custom Linux OS with Linux 4.14.78	Masimo Rad-97
5	Custom Linux OS with Linux 4.14.78	Masimo Rad-67
6	Custom Linux OS with Linux 4.14.78	Masimo Radius VSM
7	Custom Linux OS with Linux 4.16.7	Masimo Radius-7
8	Custom Linux OS using Yocto standard	Masimo iSirona Connectivity Hub
9	Android 6.0.1	Masimo Uniview Media Hub
10	Android 7.0	Masimo Uniview 60 Tablet
11	Android 12.0	Masimo Zebra TC51-HC Phone
12	Custom Linux OS with Linux kernel 4.14.78	Masimo Radical-7 w/ ARM Cortex-A9 (ARMv7-A)
13	Custom Linux OS with Linux kernel 4.14.78	Masimo Root w/ ARM Cortex-A9 (ARMv7-A)

The cryptographic module maintains compliance when operating on a general-purpose computer (GPC) with any of the following supported bare metal and virtual environments:

- Red Hat Enterprise Linux 8 on VMware ESXi 5.x and 6.x
- Red Hat Enterprise Linux 8 on Linux KVM 7
- Windows Server 2019 on VMware ESXi 6.x
- Windows 10 Pro on VMware Workstation 15.x
- Android 5.x – 12.x

The cryptographic module also maintains validation compliance when operating on any 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 Algorithm Implementations

Validation certificates for each Approved security function are listed in Table 4. Note that there are algorithms, modes, and key/moduli sizes that have been CAVP-tested but are not used by any Approved service of the module. Only the algorithms, modes/methods, and key lengths/curves/moduli shown in Table 4 are used by an Approved service of the module.

Table 4 – Approved Algorithms

CAVP Certificate	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A3595	**AES** *FIPS PUB[3] 197* *NIST SP 800-38A*	CBC[4], CFB1[5], CFB8, CFB128, CTR[6], ECB[7], OFB[8]	128, 192, 256	Encryption/decryption
A3595	**AES** *NIST SP 800-38B*	CMAC[9]	128, 192, 256	MAC generation/verification
A3595	**AES** *NIST SP 800-38C*	CCM[10]	128, 192, 256	Encryption/decryption
A3595	**AES** *NIST SP 80- 38D*	GCM[11] (internal IV)	128, 192, 256	Encryption/decryption
A3595	**AES** *NIST SP 80- 38D*	GMAC[12]	128, 192, 256	Encryption/decryption
A3595	**AES** *NIST SP 800-38E*	XTS[13,14,15]	128, 256	Encryption/decryption

[3] PUB – Publication
[4] CBC – Cipher Block Chaining
[5] CFB – Cipher Feedback
[6] CTR – Counter
[7] ECB – Electronic Code Book
[8] OFB – Output Feedback
[9] CMAC – Cipher-Based Message Authentication Code
[10] CCM – Counter with Cipher Block Chaining - Message Authentication Code
[11] GCM – Galois Counter Mode
[12] GMAC – Galois Message Authentication Code
[13] XOR – Exclusive OR
[14] XEX – XOR Encrypt XOR
[15] XTS – XEX-Based Tweaked-Codebook Mode with Ciphertext Stealing

CAVP Certificate	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A3595	**AES** *NIST SP 800-38F*	KW[16], KWP[17]	128, 192, 256	Encryption/decryption
A3595	**CVL**[18] *RFC[19] 7627*	TLS v1.2 KDF RCF7627	-	Key derivation *No part of the TLS v1.2 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A3596	**CVL** *RFC 8446*	TLS v1.3 KDF	-	Key derivation *No part of the TLS v1.3 protocol, other than the KDF, has been tested by the CAVP and CMVP.*
A3595	**DRBG**[20] *NIST SP 800-90Arev1*	Counter-based	AES-128, AES-192, AES-256	Deterministic random bit generation
A3595	**DSA**[21] *FIPS PUB 186-4*	-	2048/224, 2048/256, 3072/256 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Domain parameter generation
		-	1024/160, 2048/224, 2048/256, 3072/256 (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Domain parameter verification
		-	2048/224, 2048/256, 3072/256	Key pair generation
		-	2048/224, 2048/256, 3072/256 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
		-	1024/160, 2048/224, 2048/256, 3072/256 (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification
A3595	**ECDSA**[22] *FIPS PUB 186-4*	Secrets generation mode: Testing candidates	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Key pair generation
		-	B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521	Public key validation

[16] KW – Key Wrap
[17] KWP – Key Wrap with Padding
[18] CVL – Component Validation List
[19] RFC – Request for Comments
[20] DRBG – Deterministic Random Bit Generator
[21] DSA – Digital Signature Algorithm
[22] ECDSA – Elliptic Curve Digital Signature Algorithm

CAVP Certificate	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
		-	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
		-	B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521 (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification
A3595	**HMAC** *FIPS PUB 198-1*	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	-	Message authentication
A3595 A3596	**KAS**[23] *NIST SP 800-56Arev3*	KAS-ECC-SSC with KDFs (TLS 1.2 RFC7627, TLS 1.3)	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Key agreement *SSP establishment methodology provides between 112 and 256 bits of encryption strength.*
		KAS-FFC-SSC with KDFs (TLS 1.2 RFC7627, TLS 1.3)	FB, FC	Key agreement *SSP establishment methodology provides 112 bits of encryption strength.*
A3595	**KAS-ECC-SSC**[24] *NIST SP 800-56Arev3*	ephemeralUnified	B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521	Shared secret computation[25]
A3595	**KAS-FFC-SSC**[26] *NIST SP 800-56Arev3*	dhEphem	FB, FC	Shared secret computation[27]
A3595	**KTS** *NIST SP 800-38C*	AES-CCM	128, 192, 256	Key wrap/unwrap (authenticated encryption)[28] *SSP establishment methodology provides between 128 and 256 bits of encryption strength*
A3595	**KTS** *NIST SP 800-38D*	AES-GCM	128, 192, 256	Key wrap/unwrap (authenticated encryption)[29] *SSP establishment methodology provides between 128 and 256 bits of encryption strength*

[23] KAS – Key Agreement Scheme
[24] KAS-ECC-SSC – Key Agreement Scheme - Elliptic Curve Cryptography - Shared Secret Computation
[25] Key agreement method complies with *FIPS 140-3 Implementation Guidance* D.F, scenario 2(1).
[26] KAS-FFC-SSC – Key Agreement Scheme - Finite Field Cryptography - Shared Secret Computation
[27] Key agreement method complies with *FIPS 140-3 Implementation Guidance* D.F, scenario 2(1).
[28] Per *FIPS 140-3 Implementation Guidance* D.G, AES-CCM is an Approved key transport technique.
[29] Per *FIPS 140-3 Implementation Guidance* D.G, AES-GCM is an Approved key transport technique.

CAVP Certificate	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
A3595	**KTS**[30] *NIST SP 800-38F*	AES key wrap	128, 192, 256	Key wrap/unwrap *SSP establishment methodology provides between 128 and 256 bits of encryption strength*
A3595	**KTS** *FIPS PUB 197* *NIST SP 800-38B*	AES with CMAC	128, 192, 256	Key wrap/unwrap (encryption with message authentication)[31] *SSP establishment methodology provides between 128 and 256 bits of encryption strength*
A3595	**KTS** *FIPS PUB 197* *NIST SP 800-38D*	AES with GMAC	128, 192, 256	Key wrap/unwrap (encryption with message authentication)[32] *SSP establishment methodology provides between 128 and 256 bits of encryption strength*
A3595	**KTS** *FIPS PUB 197* *FIPS PUB 198-1*	AES-CBC with HMAC	128, 192, 256	Key wrap/unwrap (encryption with message authentication)[33] *SSP establishment methodology provides between 128 and 256 bits of encryption strength*
A3595	**KTS** *NIST SP 800-67rev2* *NIST SP 800-38B*	Triple-DES with CMAC	112 (KO2), 168 (KO1)	Key unwrap (encryption with message authentication)[34]
A3595	**KTS** *NIST SP 800-67rev2* *FIPS PUB 198-1*	Triple-DES with HMAC	112 (KO2), 168 (KO1)	Key unwrap (encryption with message authentication)[35] *SSP establishment methodology provides 112 or 168 bits of encryption strength*
A3595	**PBKDF2**[36] *NIST SP 800-132*	Section 5.4, option 1a	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Password-based key derivation
A3595	**RSA** *FIPS PUB 186-4*	Key generation mode: B.3.3	2048, 3072, 4096	Key pair generation
		ANSI[37] X9.31	2048, 3072, 4096 (SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
			1024, 2048, 3072, 4096 (SHA-1, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification

[30] KTS – Key Transport Scheme
[31] Per FIPS 140-3 Implementation Guidance D.G, AES with CMAC is an Approved key transport technique.
[32] Per FIPS 140-3 Implementation Guidance D.G, AES with GMAC is an Approved key transport technique.
[33] Per FIPS 140-3 Implementation Guidance D.G, AES with HMAC is an Approved key transport technique.
[34] Per FIPS 140-3 Implementation Guidance D.G, Triple-DES with CMAC is an Approved key transport technique.
[35] Per FIPS 140-3 Implementation Guidance D.G, Triple-DES with HMAC is an Approved key transport technique.
[36] PBKDF2 – Password-based Key Derivation Function 2
[37] ANSI – American National Standards Institute

CAVP Certificate	Algorithm and Standard	Mode / Method	Description / Key Size(s) / Key Strengths	Use / Function
		PKCS#1 v1.5	2048, 3072, 4096 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
			1024, 2048, 3072, 4096 (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification
		PSS[38]	2048, 3072, 4096 (SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature generation
			1024, 2048, 3072, 4096 (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)	Digital signature verification
A3595	**SHA-3** *FIPS PUB 202*	SHA3-224, SHA3-256, SHA3-384, SHA3-512	-	Message digest
A3595	**SHAKE**[39] *FIPS PUB 202*	SHAKE-128, SHAKE-256	-	Message digest
A3595	**SHS**[40] *FIPS PUB 180-4*	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	-	Message digest
A3595	**Triple-DES** *NIST SP 800-67rev2* *NIST SP 800-38A*	CBC, CFB1, CFB8, CFB64, ECB, OFB	168 (KO1)	Decryption
A3595	**Triple-DES** *NIST SP 800-67rev2* *NIST SP 800-38B*	CMAC	112 (KO2), 168 (KO1)	MAC verification

The module implements the non-Approved but allowed algorithms shown in Table 5 below.

Table 5 – Non-Approved Algorithms Allowed in the Approved Mode of Operation

Algorithm	Caveat	Use / Function
AES (Cert. A3595)	-	Key unwrapping (using any approved mode)
Triple-DES (Cert. A3595)	-	Key unwrapping (using any approved mode with two-key or three-key)

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

The module employs the non-Approved algorithms shown in Table 6 below. These algorithms shall not be used in the module's Approved mode of operation.

[38] PSS – Probabilistic Signature Scheme
[39] SHAKE – Secure Hash Algorithm KECCAK
[40] SHS – Secure Hash Standard

Table 6 – Non-Approved Algorithms Not Allowed in the Approved Mode of Operation

Algorithm / Function	Use / Function
AES-GCM (non-compliant when used with external IV)	Authenticated encryption/decryption
AES-OCB[41]	Authenticated encryption/decryption
ANSI X9.31 RNG (with 128-bit AES core)	Random number generation
ARIA	Encryption/decryption
Blake2	Encryption/decryption
Blowfish	Encryption/decryption
Camellia	Encryption/decryption
CAST, CAST5	Encryption/decryption
ChaCha20	Encryption/decryption
DES	Encryption/decryption
DH (non-compliant with untested key sizes or keys providing less than 112 bits of encryption strength)	Key agreement
DRBG (non-compliant when using Hash_DRBG and HMAC_DRBG)	Random bit generation
DSA (non-compliant with untested key sizes or keys providing less than 112 bits of encryption strength)	Key pair generation; digital signature generation; digital signature verification
DSA, ECDSA, and RSA (non-compliant when used with SHA-1 outside the TLS protocol)	Digital signature generation
ECDH (non-compliant with curves P-192, K-163, B-163, and non-NIST curves)	Key agreement
ECDSA (non-compliant with curves P-192, K-163, B-163, and non-NIST curves)	Key pair generation; digital signature generation; digital signature verification
EdDSA[42]	Key pair generation; digital signature generation; digital signature verification
IDEA	Encryption/decryption
KDF	Key derivation functions for TLS 1.0/1.1; HKDF; KBKDF
MD2, MD4, MD5	Message digest
Poly1305	Message authentication code
RC2[43], RC4, RC5	Encryption/decryption
RIPEMD	Message digest
RMD160	Message digest
RSA (non-compliant with untested key sizes or keys providing less than 112 bits of encryption strength)	Key pair generation; digital signature generation; digital signature verification
RSA (non-compliant with untested functions)	key transport
SEED	Encryption/decryption

[41] OCB – Offset Codebook
[42] EdDSA – Edwards-curve Digital Signature Algorithm
[43] RC – Rivest Cipher

Algorithm / Function	Use / Function
SHA-1 (non-compliant)	Signature generation for TLS 1.0/1.1
SM2, SM3, SM3	Message digest
SM4	Encryption/decryption
Triple-DES (non-compliant)	Encryption; MAC generation; key wrapping
Whirlpool	Message digest

2.3 Cryptographic Boundary

As a software cryptographic module, the module has no physical components. The physical perimeter of the cryptographic module is defined by each host platform on which the module is installed. Figure 1 and Figure 2 below provide hardware block diagrams of the host devices used for testing and illustrate the module's physical perimeter.

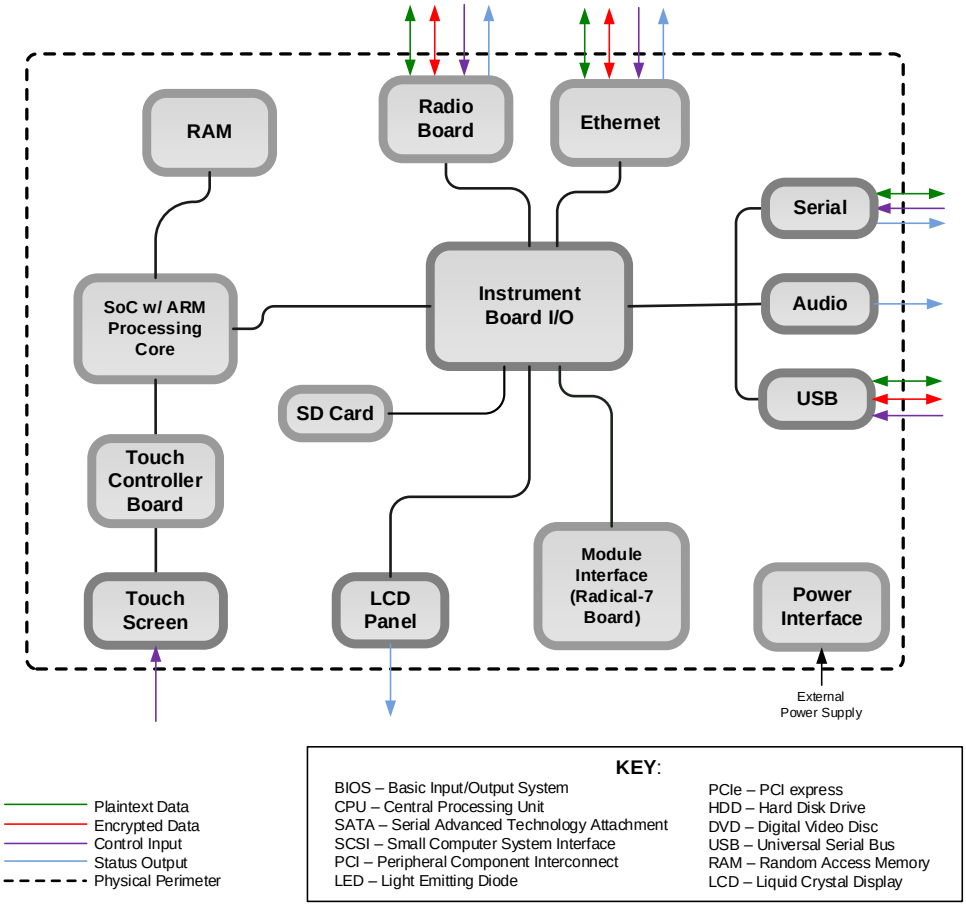

Figure 1 – Hardware Block Diagram (Root)

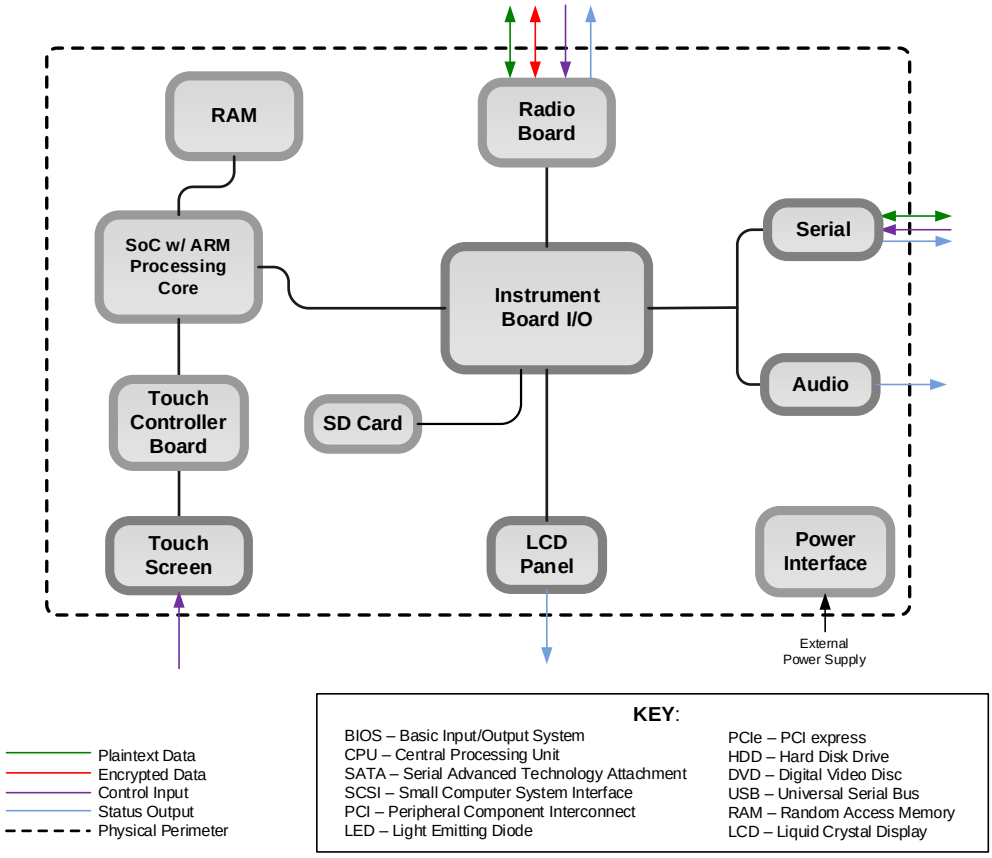

Figure 2 – Hardware Block Diagram (Radical-7)

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 (HMAC digest file for libcrypto integrity checks)
- libssl.hmac (HMAC digest file for libssl integrity checks)

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

Figure 3 shows the logical block diagram of the module executing in memory and its interactions with surrounding software components, as well as the module's physical perimeter and cryptographic boundary.

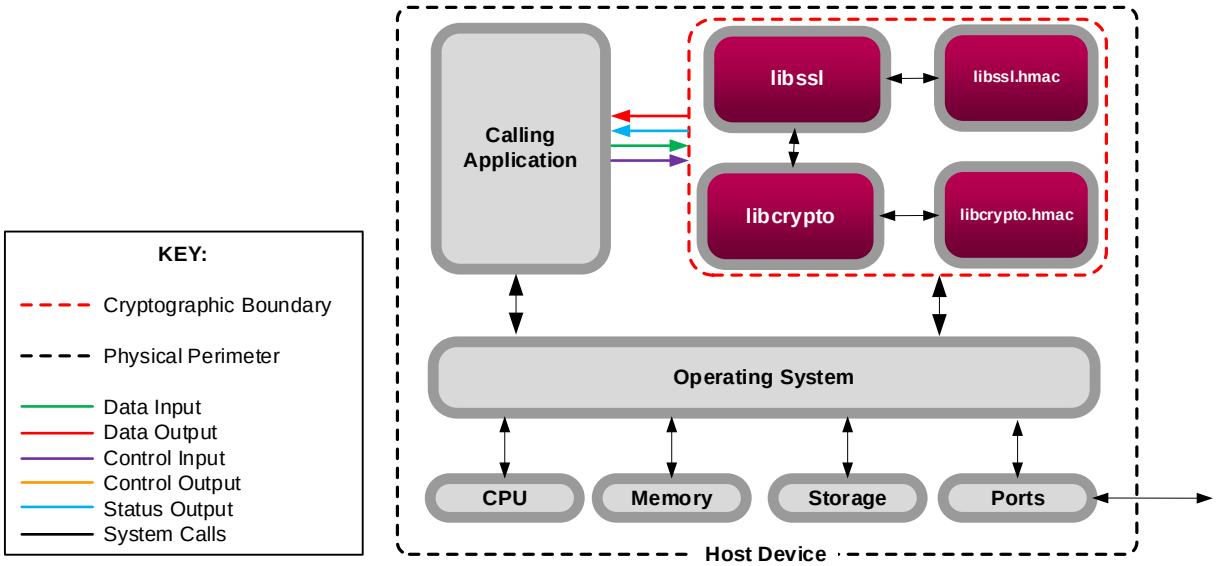

Figure 3 – Module Block Diagram (with Cryptographic Boundary)

2.4 Modes of Operation

The module supports two modes of operation: Approved and non-Approved. The module will be in its Approved mode when all pre-operational self-tests have completed successfully, and only Approved services are invoked. Table 4 and Table 5 list the Approved and allowed algorithms; Table 9 provides descriptions of the Approved services.

The module alternates on a service-by-service basis between Approved and non-Approved modes of operation. The module will switch to the non-Approved mode upon execution of a non-Approved service. The module will switch back to the Approved mode upon execution of an Approved service. Table 6 lists the non-Approved algorithms implemented by the module; Table 10 below lists the services that constitute the non-Approved mode.

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

3. Cryptographic Module Interfaces

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

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

As a software library, the cryptographic module has no direct access to any of the host platform's physical ports, as it communicates only to the calling application via its well-defined API. A mapping of the FIPS-defined interfaces and the module's ports and interfaces can be found in Table 7. Note that the module does not output control information, and thus has no specified control output interface.

Table 7 – Ports and Interfaces

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

4. Roles, Services, and Authentication

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

4.1 Authorized Roles

The module supports two roles that authorized operators can assume:

- Crypto Officer - The CO role performs cryptographic initialization or management functions and general security services.
- User – The User role performs general security services, including cryptographic operations and other approved security functions.

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

Table 8 below lists the supported roles, along with the services (including input and output) available to each role.

Table 8 – Roles, Service Commands, Input and Output

Role	Service	Input	Output
CO	Show Status	API call parameters	Current operational status
CO	Perform self-tests on-demand	Re-instantiate module; API call parameters	Status
CO	Zeroize	Restart calling application; reboot or power-cycle host device	None
CO	Show versioning information	API call parameters	Module name, version
User	Perform symmetric encryption	API call parameters, key, plaintext	Status, ciphertext
User	Perform symmetric decryption	API call parameters, key, ciphertext	Status, plaintext
User	Generate symmetric digest	API call parameters, key, plaintext	Status, digest
User	Verify symmetric digest	API call parameters, digest	Status
User	Perform authenticated symmetric encryption	API call parameters, key, plaintext	Status, ciphertext
User	Perform authenticated symmetric decryption	API call parameters, key, ciphertext	Status, plaintext
User	Generate random number	API call parameters	Status, random bits
User	Perform keyed hash operations	API call parameters, key, message	Status, MAC[44]
User	Perform hash operation	API call parameters, message	Status, hash
User	Generate DSA domain parameters	API call parameters	Status, domain parameters
User	Verify DSA domain parameters	API call parameters	Status, domain parameters
User	Generate asymmetric key pair	API call parameters	Status, key pair
User	Verify ECDSA public key	API call parameters, key	Status
User	Generate digital signature	API call parameters, key, message	Status, signature

[44] MAC – Message Authentication Code

Role	Service	Input	Output
User	Verify digital signature	API call parameters, key, signature, message	Status
User	Perform key wrap	API call parameters, encryption key, key	Status, encrypted key
User	Perform key un-encapsulation	API call parameters, decryption key, key	Status, decrypted key
User	Compute shared secret	API call parameters	Status, shared secret
User	Derive keys via TLS KDF	API call parameters, TLS pre-master secret	Status, TLS keys
User	Perform key agreement functions	API call parameters	Status, symmetric key
User	Derive key via PBKDF2	API call parameters, passphrase	Status, symmetric key

4.2 Authentication Methods

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

4.3 Services

Descriptions of the approved services available to the authorized roles are provided in Table 9 below.

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. Additional details on the APIs used for the Approved service indicators are provided in Appendix A below.

The keys and Sensitive Security Parameters (SSPs) listed in the table indicate the type of access required using the following notation:

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

Table 9 – Approved Services

Service	Description	Approved Security Function(s)	Keys and/or SSPs	Roles	Access Rights to Keys and/or SSPs	Indicator
Show Status	Return module mode status	None	None	CO	N/A	N/A
Perform self-tests on-demand	Perform pre-operational self-tests	HMAC (Cert. A3595) SHA2-256 (Cert. A3595)	HMAC key	CO	N/A	API return value
Zeroize	Zeroize and de-allocate memory containing sensitive data	None	All SSPs	CO	All SSPs – Z	N/A
Show versioning information	Return module versioning information	None	None	CO	N/A	N/A
Perform symmetric encryption	Encrypt plaintext data	AES (Cert. A3595) AES XTS (Cert. A3595)	AES key AES XTS key	User	AES key – WE AES XTS key – WE	API return value
Perform symmetric decryption	Decrypt ciphertext data	AES (Cert. A3595) AES XTS (Cert. A3595) Triple-DES (Cert. A3595)	AES key AES XTS key Triple-DES key	User	AES key – WE AES XTS key – WE Triple-DES key – WE	API return value
Generate symmetric digest	Generate symmetric digest	AES CMAC (Cert. A3595) AES GMAC (Cert. A3595)	AES CMAC key AES GMAC key	User	AES CMAC key – WE AES GMAC key – WE	API return value
Verify symmetric digest	Verify symmetric digest	AES CMAC (Cert. A3595) AES GMAC (Cert. A3595) Triple-DES CMAC (Cert. A3595)	AES CMAC key AES GMAC key Triple-DES CMAC key	User	AES CMAC key – WE AES GMAC key – WE Triple-DES CMAC key – WE	API return value
Perform authenticated symmetric encryption	Encrypt plaintext using supplied AES GCM key and IV	AES GCM (Cert. A3595)	AES GCM key AES GCM IV	User	AES GCM key – WE AES GCM IV – WE	API return value
Perform authenticated symmetric decryption	Decrypt ciphertext using supplied AES GCM key and IV	AES GCM (Cert. A3595)	AES GCM key AES GCM IV	User	AES GCM key – WE AES GCM IV – WE	API return value
Generate random number	Generate random bits using DRBG	CTR_DRBG (Cert. A3595)	DRBG entropy input DRBG seed DRBG 'V' value DRBG 'Key' value	User	DRBG entropy input – WE DRBG seed – GE DRBG 'V' value – GE DRBG 'Key' value – GE	API return value
Perform keyed hash operation	Compute a message authentication code	HMAC (Cert. A3595) SHA-3 (Cert. A3595) SHS (Cert. A3595)	HMAC key	User	HMAC key – WE	API return value
Perform hash operation	Compute a message digest	SHA-3 (Cert. A3595) SHAKE (Cert. A3595) SHS (Cert. A3595)	None	User	N/A	API return value
Generate DSA domain parameters	Generate DSA domain parameters	CTR_DRBG (Cert. A3595) DSA (Cert. A3595)	None	User	N/A	API return value
Verify DSA domain parameters	Verify DSA domain parameters	DSA (Cert. A3595)	None	User	N/A	API return value
Generate asymmetric key pair	Generate a public/private key pair	CTR_DRBG (Cert. A3595) DSA (Cert. A3595) ECDSA (Cert. A3595) RSA (Cert. A3595)	DSA public key DSA private key ECDSA public key ECDSA private key RSA public key RSA private key	User	DSA public key – GR DSA private key – GR ECDSA public key – GR ECDSA private key – GR RSA public key – GR RSA private key – GR	API return value
Verify ECDSA public key	Verify an ECDSA public key	ECDSA (Cert. A3595)	ECDSA public key	User	ECDSA public key – W	API return value
Generate digital signature	Generate a digital signature	DSA (Cert. A3595) ECDSA (Cert. A3595) RSA (Cert. A3595) SHS (Cert. A3595)	DSA private key ECDSA private key RSA private key	User	DSA private key – WE ECDSA private key – WE RSA private key – WE	API return value
Verify digital signature	Verify a digital signature	DSA (Cert. A3595) ECDSA (Cert. A3595) RSA (Cert. A3595) SHS (Cert. A3595)	DSA public key ECDSA public key RSA public key	User	DSA public key – WE ECDSA public key – WE RSA public key – WE	API return value

Service	Description	Approved Security Function(s)	Keys and/or SSPs	Roles	Access Rights to Keys and/or SSPs	Indicator
Perform key wrap	Perform key wrap	AES (Cert. A3595) AES-KW (Cert. A3595) AES-KWP (Cert. A3595) AES-GCM (Cert. A3595) CCM (Cert. A3595) CMAC (Cert. A3595) GMAC (Cert. A3595) HMAC (Cert. A3595)	AES key AES CMAC key AES GMAC key AES GCM key AES GCM IV HMAC key	User	AES key – WE AES CMAC key – WE AES GMAC key – WE AES GCM key – WE AES GCM IV – WE HMAC key – WE	API return value
Perform key unwrap	Perform key unwrap	AES (Cert. A3595) AES-KW (Cert. A3595) AES-KWP (Cert. A3595) AES-GCM (Cert. A3595) CCM (Cert. A3595) CMAC (Cert. A3595) GMAC (Cert. A3595) HMAC (Cert. A3595) Triple-DES (Cert. A3595)	AES key AES CMAC key AES GMAC key AES GCM key AES GCM IV HMAC key Triple-DES key	User	AES key – WE AES CMAC key – WE AES GMAC key – WE AES GCM key – WE AES GCM IV – WE HMAC key – WE Triple-DES key – WE	API return value
Compute shared secret	Compute DH/ECDH shared secret suitable for use as input to a TLS 1.2/1.3 KDF	KAS-ECC-SSC (Cert. A3595) KAS-FFC-SSC (Cert. A3595)	DH public key DH private key ECDH public key ECDH private key TLS 1.2 pre-master secret TLS 1.3 handshake secret	User	DH public key – WE DH private key – WE ECDH public key – WE ECDH private key – WE TLS 1.2 pre-master secret – G TLS 1.3 handshake secret – G	API return value
Derive keys via TLS KDF	Derive TLS 1.2/1.3 session and integrity keys	KDF (TLS 1.2 RFC7627) (Cert. A3595) KDF (TLS 1.3) (Cert. A3596)	TLS 1.2 pre-master secret TLS 1.2 master secret TLS 1.3 handshake secret TLS 1.3 handshake traffic secrets TLS 1.3 master secret TLS 1.3 application traffic secrets AES key AES GCM key AES GCM IV HMAC key	User	TLS 1.2 pre-master secret – WE TLS 1.2 master secret – GE TLS 1.3 handshake secret – GE TLS 1.3 master secret – GE TLS 1.3 handshake traffic secrets – GE TLS 1.3 master secret – GE TLS 1.3 application traffic secrets – GE AES key – G AES GCM key – G AES GCM IV – G HMAC key – G	API return value
Perform key agreement functions	Establish symmetric key using DH/ECDH key agreement	KAS-ECC-SSC (Cert. A3595) KAS-FFC-SSC (Cert. A3595) KDF (TLS 1.2) (Cert. A3595) KDF (TLS 1.3) (Cert. A3596)	DH public key DH private key ECDH public key ECDH private key TLS 1.2 pre-master secret TLS 1.2 master secret TLS 1.3 handshake secret TLS 1.3 handshake traffic secrets TLS 1.3 master secret TLS 1.3 application traffic secrets AES key AES GCM key AES GCM IV HMAC key	User	DH public key – WE DH private key – WE ECDH public key – WE ECDH private key – WE TLS 1.2 pre-master secret – GE TLS 1.2 master secret – GE TLS 1.3 handshake secret – GE TLS 1.3 master secret – GE TLS 1.3 handshake traffic secrets – GE TLS 1.3 master secret – GE TLS 1.3 application traffic secrets – GE AES key – G AES GCM key – G AES GCM IV – G HMAC key – G	API return value
Derive key via PBKDF2	Derive key from PBKDF2	PBKDF (Cert. A3595)	Passphrase	User	Passphrase– WE	API return value

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

Table 10 below lists the non-approved services available to module operators.

Table 10 – Non-Approved Services

Service	Description	Algorithm(s) Accessed	Role	Indicator
Perform data encryption (non-compliant)	Perform symmetric data encryption	ARIA, Blake2, Blowfish, Camellia, CAST, CAST5, ChaCha20, DES, IDEA, RC2, RC4, RC5, SEED, SM4, Triple-DES (non-compliant)	User	API return value
Perform data decryption (non-compliant)	Perform symmetric data decryption	ARIA, Blake2, Blowfish, Camellia, CAST, CAST5, ChaCha20, DES, IDEA, RC2, RC4, RC5, SEED, SM4	User	API return value
Perform MAC operations (non-compliant)	Perform message authentication operations	Poly1305, Triple-DES/CMAC (non-compliant for MAC generation)	User	API return value
Perform hash operation (non-compliant)	Perform hash operation	MD2, MD4, MD5, RIPEMD, RMD160, SM2, SM3, SM4, Whirlpool	User	API return value
Perform digital signature functions (non-compliant)	Perform digital signature functions	DSA (non-compliant), ECDSA (non-compliant), RSA (non-compliant)	User	API return value
Perform key agreement functions (non-compliant)	Perform key agreement functions	DH (non-compliant), ECDH (non-compliant)	User	API return value
Perform key wrap (non-compliant)	Perform key wrap functions	Triple-DES/CMAC (non-compliant)	User	API return value
Perform key encapsulation (non-compliant)	Perform key encapsulation functions	RSA (non-compliant)	User	API return value
Perform key un-encapsulation (non-compliant)	Perform key un-encapsulation functions	RSA (non-compliant)	User	API return value
Perform key derivation functions (non-compliant)	Perform key derivation functions	HKDF (non-compliant), KBKDF (non-compliant), TLS v1.0/1.1 KDF (non-compliant)	User	API return value
Perform authenticated encryption/decryption (non-compliant)	Perform authenticated encryption/decryption	AES-OCB	User	API return value
Perform random number generation (non-compliant)	Perform random number generation	ANSI X9.31 RNG (with 128-bit AES core), Hash_DRBG (non-compliant), HMAC_DRBG (non-compliant)	User	API return value
Perform key pair generation (non-compliant)	Perform key pair generation	DSA (non-compliant), ECDSA (non-compliant), EdDSA, RSA (non-compliant)	User	API return value

5. Software/Firmware Security

All software components within the cryptographic boundary are verified using an Approved integrity technique implemented within the cryptographic module itself. The module implements independent HMAC SHA2-256 digest checks to test the integrity of each library file; failure of the integrity 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. The CO can initiate the pre-operational tests on demand by re-instantiating the module or issuing the `FIPS_selftest()` API command.

The Masimo Cryptographic Module is not delivered to end-users as a standalone offering. Rather, it is a pre-built component integrated into Masimo's application software. Masimo does not provide end-users with any mechanisms to directly access the module, its source code, its APIs, or any information sent to/from the module. Thus, end-users have no ability to independently load the module onto target platforms. No configuration steps are required to be performed by end-users, and no end-user action is required to initialize the module for operation.

6. Operational Environment

The Masimo Cryptographic Module comprises a software cryptographic library that executes in a modifiable operational environment.

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

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

7. Physical Security

This section is not applicable. Per section 7.7.1 of *ISO/IEC 19790:2021*, the requirements of this section are "applicable to hardware and firmware modules, and hardware and firmware components of hybrid modules".

8. Non-Invasive Security

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

9. Sensitive Security Parameter Management

9.1 Keys and SSPs

The module supports the keys and other SSPs listed Table 11. Note that all SSP import and export is electronic and is performed within the Tested OE's Physical Perimeter (TOEPP).

Table 11 – SSPs

Key/SSP Name/Type	Strength	Security Function and Cert. Number	Generation	Import / Export	Establishment	Storage	Zeroization	Use & Related Keys
Keys								
AES key (CSP)	Between 128 and 256 bits	AES (CBC, CCM, CFB, CTR, ECB, OFB, KW, KWP modes) (Cert. A3595) KTS (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	Derived via TLS KDFs	Not persistently stored by the module	Reboot or power-cycle the host device	Symmetric encryption, decryption; key transport
AES GCM key (CSP)	Between 128 and 256 bits	AES (GCM mode) (Cert. A3595) KTS (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	Derived via TLS KDFs	Not persistently stored by the module	Reboot or power-cycle the host device	Authenticated symmetric encryption, decryption; key transport
AES XTS key (CSP)	256 bits	AES (XTS mode) (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Reboot or power-cycle the host device	Symmetric encryption, decryption
AES CMAC key (CSP)	Between 128 and 256 bits	AES (CMAC mode) (Cert. A3595) KTS (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Reboot or power-cycle the host device	MAC generation, verification
AES GMAC key (CSP)	Between 128 and 256 bits	AES (GMAC mode) (Cert. A3595) KTS (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Reboot or power-cycle the host device	MAC generation, verification
Triple-DES key (CSP)	-	Triple-DES (Cert. A3595) KTS (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Reboot or power-cycle the host device	Symmetric decryption; key unwrapping
Triple-DES CMAC key (CSP)	-	Triple-Des (CMAC mode) (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Reboot or power-cycle the host device	MAC verification
HMAC key (CSP)	112 bits (minimum)	HMAC (Cert. A3595) KTS (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	Derived via TLS KDFs	Not persistently stored by the module	Reboot or power-cycle the host device	Keyed hash

Key/SSP Name/Type	Strength	Security Function and Cert. Number	Generation	Import / Export	Establishment	Storage	Zeroization	Use & Related Keys
DSA private key (CSP)	112 or 128 bits	DSA (Cert. A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	Digital signature generation
DSA public key (PSP)	112 or 128 bits	DSA (Cert. A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	Digital signature verification
ECDSA private key (CSP)	Between 112 and 256 bits	ECDSA (Cert. A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	Digital signature generation
ECDSA public key (PSP)	Between 112 and 256 bits	ECDSA (Cert. A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	Digital signature verification
RSA private key (CSP)	Between 112 and 150 bits	RSA (Cert. A3595) KTS (Cert. A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	Digital signature generation
RSA public key (PSP)	Between 80 and 150 bits	RSA (Cert. A3595) KTS (Cert. A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	Digital signature verification
DH private key (CSP)	112 bits	KAS-SSC-FFC (Cert. A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	DH shared secret computation
DH public key (PSP)	112 bits	KAS-SSC-FFC (Cert. _A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	DH shared secret computation
ECDH private key (CSP)	Between 112 and 256 bits	KAS-SSC-ECC (Cert. A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	ECDH shared secret computation
ECDH public key (PSP)	Between 112 and 256 bits	KAS-SSC-ECC (Cert. A3595)	Generated internally via approved DRBG	Imported in plaintext via API parameter Exported in plaintext via API parameter	-	Not persistently stored by the module	Reboot or power-cycle the host device	ECDH shared secret computation
Other SSPs								

Key/SSP Name/Type	Strength	Security Function and Cert. Number	Generation	Import / Export	Establishment	Storage	Zeroization	Use & Related Keys
Passphrase (PSP)	-	PBKDF (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Reboot or power-cycle the host device	Input to PBKDF for key derivation
AES GCM IV (CSP)	-	AES (GCM mode) (Cert. A3595)	Generated internally in compliance with the provisions of a peer-to-peer industry standard protocols	-	-	Not persistently stored by the module	Reboot or power-cycle the host device	Initialization vector for AES GCM
TLS 1.2 pre-master secret (CSP)	-	KDF (TLS 1.2 RFC7627) (Cert. A3595)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Reboot or power-cycle the host device	Input to TLS 1.2 KDF for derivation of secrets and keys
TLS 1.2 master secret (CSP)	-	KDF (TLS 1.2 RFC7627) (Cert. A3595)	-	-	Derived internally via TLS 1.2 KDF with EMS[45] extension	Not persistently stored by the module	Reboot or power-cycle the host device	Derivation of keys used for securing TLS 1.2 session traffic
TLS 1.3 handshake secret (CSP)	-	KDF (TLS 1.3) (Cert. A3596)	-	Imported in plaintext via API parameter Never exported	-	Not persistently stored by the module	Reboot or power-cycle the host device	Input to TLS 1.3 KDF for derivation of secrets and keys
TLS 1.3 handshake traffic secrets (CSP)	-	KDF (TLS 1.3) (Cert. A3596)	-	-	Derived internally via TLS 1.3 KDF	Not persistently stored by the module	Reboot or power-cycle the host device	Derivation of keys used for securing TLS 1.3 handshake traffic
TLS 1.3 master secret (CSP)	-	KDF (TLS 1.3) (Cert. A3596)	-	-	Derived internally via TLS 1.3 KDF	Not persistently stored by the module	Reboot or power-cycle the host device	Derivation of TLS 1.3 application traffic secrets
TLS 1.3 application traffic secrets (CSP)	-	KDF (TLS 1.3) (Cert. A3596)	-	-	Derived internally via TLS 1.3 KDF	Not persistently stored by the module	Reboot or power-cycle the host device	Derivation of keys used for securing TLS 1.3 session traffic
DRBG entropy input (CSP)	-	DRBG (Cert. A3595)	-	Imported in plaintext via API parameter[46] Never exported	-	Not persistently stored by the module	Reboot or power-cycle the host device	Entropy material for DRBG
DRBG seed (CSP)	-	DRBG (Cert. A3595)	Generated internally using nonce along with DRBG entropy input	-	-	Not persistently stored by the module	Reboot or power-cycle the host device	Seeding material for DRBG
DRBG 'V' value (CSP)	-	DRBG (Cert. A3595)	Generated internally	-	-	Not persistently stored by the module	Reboot or power-cycle the host device	State values for DRBG
DRBG 'Key' value (CSP)	-	DRBG (Cert. A3595)	Generated internally	-	-	Not persistently stored by the module	Reboot or power-cycle the host device	State values for DRBG

[45] EMS – Extended Master Secret

[46] The module obtains entropy input from the calling application (which is outside of the cryptographic boundary) but exercises no control over the amount or the quality of the obtained entropy. As such, there is no assurance of the minimum strength of generated keys.

9.2 DRBGs

The module implements the following Approved DRBG:

- Counter-based DRBG

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

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

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

9.3 SSP Storage Techniques

There is no mechanism within the module's cryptographic boundary for the persistent storage of SSPs. The module stores DRBG state values for the lifetime of the DRBG instance. The module uses SSPs passed in on the stack by the calling application and does not store these SSPs beyond the lifetime of the API call.

9.4 SSP Zeroization Methods

Maintenance, including protection and zeroization, of any keys and CSPs that exist outside the module's cryptographic boundary are the responsibility of the end-user. For the zeroization of keys in volatile memory, module operators can reboot/power-cycle the host device.

9.5 RBG Entropy Sources

The cryptographic module's entropy scheme follows the scenario given in *FIPS 140-3 Implementation Guidance* 9.3.A, section 2(b).

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

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

10. Self-Tests

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

10.1 Pre-Operational Self-Tests

The module performs the following pre-operational self-test(s):

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

10.2 Conditional Self-Tests

The module performs the following conditional self-tests:

- Conditional cryptographic algorithm self-tests (CASTs)
  - AES ECB encrypt KAT[47] (128-bit)
  - AES ECB decrypt KAT (128-bit)
  - AES CCM encrypt KAT (192-bit)
  - AES CCM decrypt KAT (192-bit)
  - AES GCM encrypt KAT (128-bit)
  - AES GCM decrypt KAT (128-bit)
  - AES XTS encrypt KAT (128/256-bit)
  - AES XTS decrypt KAT (128/256-bit)
  - AES CMAC generate KAT (CBC mode; 128/192/256-bit)
  - AES CMAC verify KAT (CBC mode; 128/192/256-bit)
  - Triple-DES ECB encrypt KAT (3-Key)
  - Triple-DES ECB decrypt KAT (3-Key)
  - Triple-DES CMAC generate KAT (CBC mode; 3-Key)
  - Triple-DES CMAC verify KAT (CBC mode; 3-Key)
  - CTR_DRBG generate/instantiate/reseed health checks (256-bit AES)
  - DSA sign KAT (2048-bit; SHA2-256)
  - DSA verify KAT (2048-bit; SHA2-256)
  - ECDSA sign KAT (P-224 and K-233 curves; SHA2-256)
  - ECDSA verify KAT (P-224 and K-233 curves; SHA2-256)
  - RSA sign KAT (2048-bit; SHA2-256; PKCS#1.5 scheme)
  - RSA verify KAT (2048-bit; SHA2-256; PKCS#1.5 scheme)
  - HMAC KATs (SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512)
  - SHA-1 KAT

[47] KAT – Known Answer Test

- SHA-2 KATs (SHA2-224, SHA2-256, SHA2-384, SHA2-512)
- SHA-3 KAT (SHA3-256)
- FFC DH Shared Secret "Z" Computation KAT (2048-bit)
- ECC CDH Shared Secret "Z" Computation KAT (P-224 curve)
- PBKDF2 KAT (SHA2-256)
- TLS 1.2 KDF KAT
- TLS 1.3 KDF KAT

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

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

10.3 Self-Test Failure Handling

The module reaches the critical error state when any self-test fails. Upon test failure, the module immediately terminates the calling application's API call with a returned error code and sets an internal flag, signaling the error condition. For any subsequent request made by the calling application for cryptographic services, the module will return a failure indicator, thereby disabling all access to its cryptographic functions, sensitive security parameters (SSPs), and data output services while the error condition persists.

To recover, the module must be re-instantiated by the calling application. If the pre-operational self-tests complete successfully, then the module can resume normal operations. If the module continues to experience self-test failures after reinitializing, then the module will not be able to resume normal operations, and the CO should contact Masimo Corporation for assistance.

11. Life-Cycle Assurance

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

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

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

11.1 Secure Installation

The module is an integrated component of Masimo's product application software, module operators have no ability to independently load the module onto the target platform. The module and its calling application are to be installed on a platform specified in section 2.1 or one where portability is maintained. Masimo does not provide any mechanisms to directly access the module, its source code, its APIs, or any information sent between it and other Masimo applications.

11.2 Initialization

This module is designed to support Masimo 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 for module initialization.

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

11.3 Startup

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

11.4 Administrator Guidance

There are no specific management activities required of the CO role to ensure that the module runs securely. If any irregular activity is observed, or if the module is consistently reporting errors, then Masimo 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. A non-zero return value indicates that the module has passed all pre-operational self-tests and is currently in its Approved mode.

- 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.

11.5 Non-Administrator Guidance

The following list provides additional policies for non-Administrators:

- The module uses PBKDF2 option 1a from section 5.4 of *NIST SP 800-132*. The iteration count shall be selected as large as possible, as long as the time required to generate the resultant key is acceptable for module operators. The minimum iteration count shall be 1000.

  The length of the passphrase used in the PBKDF shall be of at least 20 characters, and shall consist of lower-case, upper-case, and numeric characters. The upper bound for the probability of guessing the value is estimated to be $1/62^{20} = 10^{-36}$, which is less than 2^{-112}.

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

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

  The AES-XTS mode shall only be used for the cryptographic protection of data on storage devices. The AES-XTS shall not be used for other purposes, such as the encryption of data in transit. In compliance with *FIPS 140-3 Implementation Guidance* C.I, the module implements the check to ensure that the two AES keys used in the XTS-AES algorithm are not identical.

- AES GCM encryption is used in the context of the TLS protocol versions 1.2 and 1.3. To meet the AES GCM (key/IV) pair uniqueness requirements from *NIST SP 800-38D*, the module generates the IV as follows:

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

  - For TLS v1.3, the protocol's implementation is contained within the boundary of the module. Per scenario 5 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, TLS 1.3 KDF, and all underlying algorithms, have been successfully tested for compliance with their respective specifications (see CAVP Certs. A3595 and A3596).The generated IV is only used in the context of the AES GCM encryption executing the provisions of the TLS 1.3 protocol.

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

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

- The cryptographic module's services are designed to be provided to a calling application. Excluding the use of the NIST-defined elliptic curves as trusted third-party domain parameters, all other assurances from *FIPS PUB 186-4* (including those required of the intended signatory and the signature verifier) are outside the scope of the module and are the responsibility of the calling application.

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

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

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

  The module includes the capability to provide the required recipient assurance of public key validity specified in section 5.6.2.2 of *NIST SP 800-56Arev3*. However, since public keys from other modules are not received directly by this module (those keys are received by the calling application), the module has no knowledge of when a public key is received. Validation of another module's public key is the responsibility of the calling application.

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

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

12. Mitigation of Other Attacks

The module does not claim to mitigate any attacks beyond the FIPS 140-3 Level 1 requirements for this validation. Therefore, per *ISO/IEC 19790:2021* section 7.12, requirements for this section are not applicable.

Appendix A. Approved Service Indicators

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

Synopsis

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

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

Description

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

- **EVP_cipher_get_service_indicator()** is used to return the Approved service indicator status for block ciphers like AES and Triple-DES.
- **DSA_get_service_indicator()** is used to return the 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 Approved service indicator status for RSA algorithm and its modes.
- **PBKDF_get_service_indicator()** is used to return the Approved service indicator status for PBKDF usage.
- **EVP_Digest_get_service_indicator()** is used to return the Approved service indicator status for SHS algorithms like SHA-1 and SHAKE.
- **EC_key_get_service_indicator()** is used to return the Approved service indicator status for elliptic curve algorithms like ECDSA and its modes.

- **CMAC_get_service_indicator()** is used to return the Approved service indicator status for CMAC requests that use AES or 3DES. 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 Approved service indicator status for HMAC requests and the associated SHS algorithm.

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

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

Return Values

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

Notes

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

Examples

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

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

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

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

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

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

Appendix B. Acronyms and Abbreviations

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

Table 12 – Acronyms and Abbreviations

Term	Definition
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interface
CAST	Cryptographic Algorithm Self-Test
CBC	Cipher Block Chaining
CCCS	Canadian Centre for Cyber Security
CCM	Counter with Cipher Block Chaining - Message Authentication Code
CFB	Cipher Feedback
CMAC	Cipher-Based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CO	Cryptographic Officer
CPU	Central Processing Unit
CSP	Critical Security Parameter
CTR	Counter
CVL	Component Validation List
DEP	Default Entry Point
DES	Data Encryption Standard
DH	Diffie-Hellman
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECC CDH	Elliptic Curve Cryptography Cofactor Diffie-Hellman
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EMI/EMC	Electromagnetic Interference /Electromagnetic Compatibility
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GMAC	Galois Message Authentication Code

Term	Definition
GPC	General-Purpose Computer
HMAC	(keyed-) Hash Message Authentication Code
KAS	Key Agreement Scheme
KAT	Known Answer Test
KTS	Key Transport Scheme
KW	Key Wrap
KWP	Key Wrap with Padding
MD	Message Digest
NIST	National Institute of Standards and Technology
OCB	Offset Codebook
OE	Operational Environment
OFB	Output Feedback
OS	Operating System
PBKDF	Password-Based Key Derivation Function
PCT	Pairwise Consistency Test
PKCS	Public Key Cryptography Standard
PSS	Probabilistic Signature Scheme
PUB	Publication
RC	Rivest Cipher
RNG	Random Number Generator
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
SHAKE	Secure Hash Algorithm KECCAK
SHS	Secure Hash Standard
SP	Special Publication
TLS	Transport Layer Security
TOEPP	Tested OE's Physical Perimeter
XEX	XOR Encrypt XOR
XTS	XEX-Based Tweaked-Codebook Mode with Ciphertext Stealing

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