CANONICAL

ubuntu 20.04 OpenSSL Cryptographic Module

version 3.1

FIPS 140-2 Non-Proprietary Security Policy

Version 1.2

Last update: 2022-03-01

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of Contents

Copyrights and Trademarks

Ubuntu and Canonical are registered trademarks of Canonical Ltd.

Linux is a registered trademark of Linus Torvalds.

1. Cryptographic Module Specification

This document is the non-proprietary FIPS 140-2 Security Policy for version 3.1 of the Ubuntu 20.04 OpenSSL Cryptographic Module. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS PUB 140-2 (Federal Information Processing Standards Publication 140-2) for a Security Level 1 software module.

The following sections describe the cryptographic module and how it conforms to the FIPS 140-2 specification in each of the required areas.

1.1. Module Overview

The Ubuntu 20.04 OpenSSL Cryptographic Module (hereafter referred to as “the module”) is a set of software libraries implementing the Transport Layer Security (TLS) protocol v1.0, v1.1, v1.2 and v1.3 and Datagram Transport Layer Security (DTLS) protocol v.1.0, v1.2 and v1.3, as well as general purpose cryptographic algorithms. The module provides cryptographic services to applications running in the user space of the underlying Ubuntu operating system through a C language Application Program Interface (API). The module utilizes processor instructions to optimize and increase performance. The module can act as a TLS server or client, and interacts with other entities via TLS/DTLS network protocols.

For the purpose of the FIPS 140-2 validation, the module is a software-only, multi-chip standalone cryptographic module validated at overall security level 1. The table below shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard.

FIPS 140-2 Section		Security 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	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	1
Overall Level		1

Table 1 - Security Levels

The cryptographic logical boundary consists of all shared libraries and the integrity check files used for Integrity Tests. The following table enumerates the files that comprise the module.

Component	Description
libssl.so.1.1	Shared library for TLS/DTLS network protocols.
libcrypto.so.1.1	Shared library for cryptographic implementations.
.libssl.so.1.1.hmac	Integrity check signature for libssl shared library.
.libcrypto.so.1.1.hmac	Integrity check signature for libcrypto shared library.

Table 2 - Cryptographic Module Components

The software block diagram below shows the module, its interfaces with the operational environment and the delimitation of its logical boundary, comprised of all the components within the **BLUE** box.

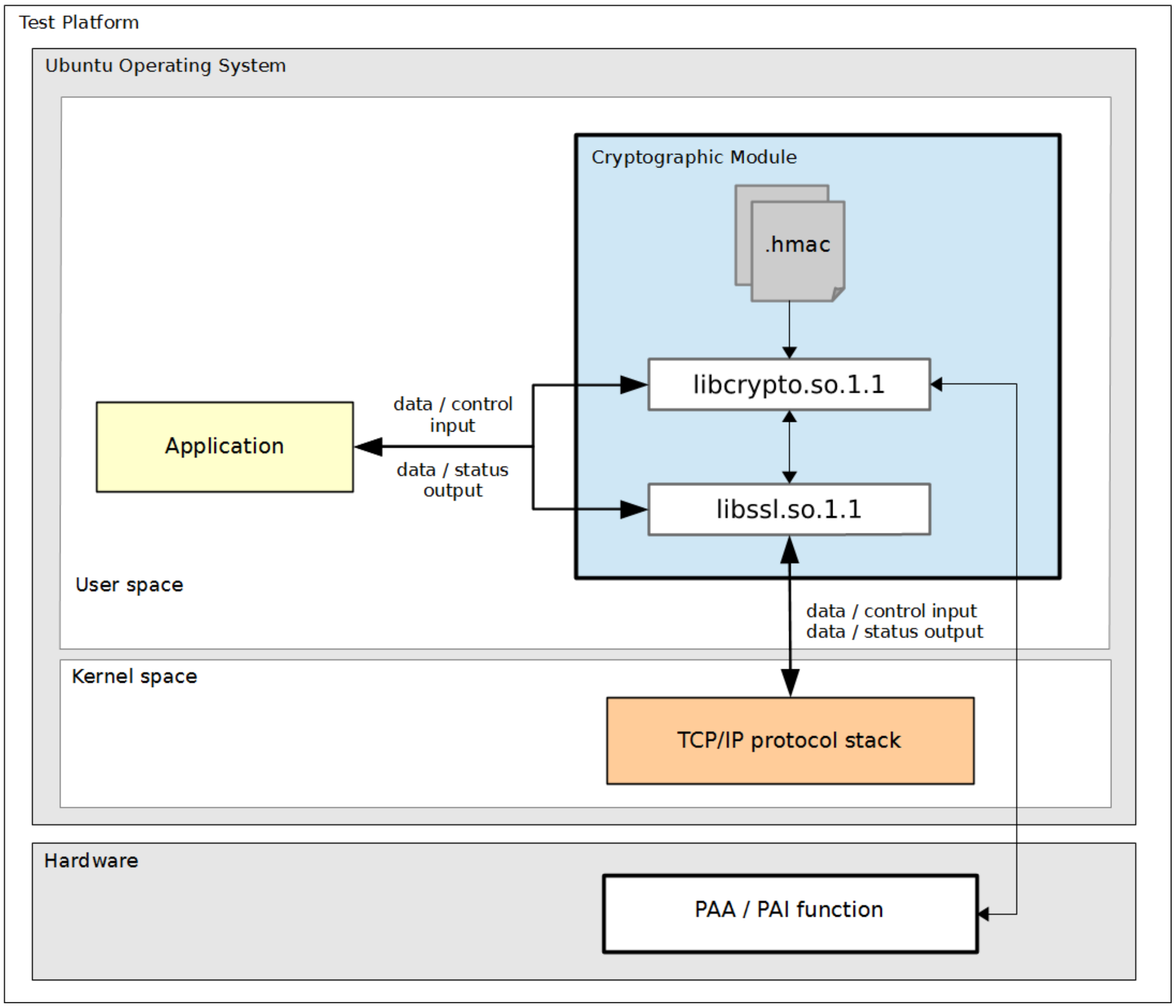

Figure 1 - Software Block Diagram

The module is aimed to run on a general purpose computer (GPC); the physical boundary of the module is the tested platforms. Figure 2 shows the major components of a GPC.

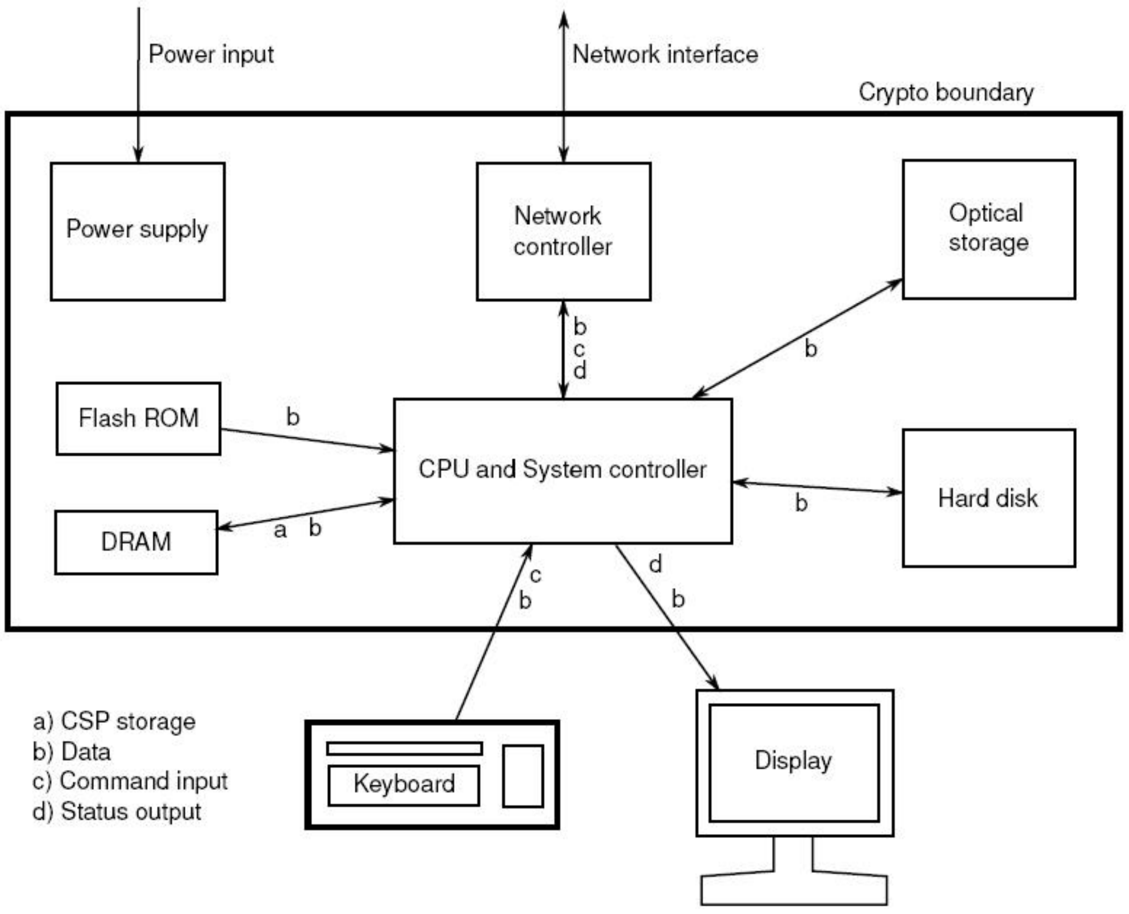

Figure 2 - Cryptographic Module Physical Boundary

The module has been tested on the test platforms shown below.

Test Platform	Processor	Test Configuration
Supermicro SYS-1019P-WTR	Intel(R) Xeon(R) Gold 6226	Ubuntu 20.04 LTS 64-bit with/without PAA
IBM z15	IBM z15	Ubuntu 20.04 LTS 64-bit with/without PAI

Table 3 - Tested Platforms

Note: Per FIPS 140-2 IG G.5, the Cryptographic Module Validation Program (CMVP) makes no statement as to the correct operation of the module or the security strengths of the generated keys when this module is ported and executed in an operational environment not listed on the validation certificate.

The platforms listed in the table below have not been tested as part of the FIPS 140-2 level 1 certification. Canonical “vendor affirms” that these platforms are equivalent to the tested and validated platforms.

Test Platform	Processor	Test Configuration
Supermicro SYS-1019P-WTR	Intel(R) Xeon(R) Platinum 8171M CPU @ 2.60GHz	Ubuntu 20.04 LTS 64-bit with/without PAA
Supermicro SYS-1019P-WTR	Intel(R) Xeon(R) CPU E5	Ubuntu 20.04 LTS 64-bit with/without PAA

Table 4 - Vendor Affirmed Platforms

1.2. Modes of Operation

The module supports two modes of operation:

- **FIPS mode** (the Approved mode of operation): only approved or allowed security functions with sufficient security strength can be used.
- **non-FIPS mode** (the non-Approved mode of operation): only non-approved security functions can be used.

The module enters FIPS mode after power-up tests succeed. Once the module is operational, the mode of operation is implicitly assumed depending on the security function invoked and the security strength of the cryptographic keys.

Critical security parameters used or stored in FIPS mode are not used in non-FIPS mode, and vice versa.

2. Cryptographic Module Ports and Interfaces

As a software-only module, the module does not have physical ports. For the purpose of the FIPS 140-2 validation, the physical ports are interpreted to be the physical ports of the hardware platform on which it runs.

The logical interfaces are the API through which applications request services, and messages sent and received from the TCP/IP protocol. The following table summarizes the four logical interfaces.

FIPS Interface	Physical Port	Logical Interface
Data Input	Ethernet ports	API input parameters, kernel I/O – network or files on file system, TLS protocol input messages.
Data Output	Ethernet ports	API output parameters, kernel I/O – network or files on file system, TLS protocol output messages.
Control Input	Keyboard, Serial port, Ethernet port, Network	API function calls, API input parameters for control.
Status Output	Serial port, Ethernet port, Network	API return codes.
Power Input	PC Power Supply Port	N/A

Table 5 - Ports and Interfaces

Note: The module is an implementation of the TLS protocol as defined in the RFC standards. The TLS protocol provides confidentiality and data integrity between communicating applications. When an application calls into the module's API, the data passed will be securely passed to the peer.

3. Roles, Services and Authentication

3.1. Roles

The module supports the following roles:

- **User role**: performs cryptographic services (in both FIPS mode and non-FIPS mode), TLS network protocol, key zeroization, get status, and on-demand self-test.
- **Crypto Officer role**: performs module installation .

The User and Crypto Officer roles are implicitly assumed by the entity accessing the module services.

3.2. Services

The module provides services to users that assume one of the available roles. All services are shown in Table 6 and Table 7, and described in detail in the user documentation (i.e., man pages) referenced in section 9.1.

The table below shows the services available in FIPS mode. For each service, the associated cryptographic algorithms, the roles to perform the service, and the cryptographic keys or Critical Security Parameters and their access rights are listed. The following convention is used to specify access rights to a CSP:

- **Create**: the calling application can create a new CSP.
- **Read**: the calling application can read the CSP.
- **Update**: the calling application can write a new value to the CSP.
- **Zeroize**: the calling application can zeroize the CSP.
- **n/a**: the calling application does not access any CSP or key during its operation.

The complete list of cryptographic algorithms, modes and key lengths, and their corresponding Cryptographic Algorithm Validation Program (CAVP) certificate numbers can be found in Table 8 and Table 9 of this security policy. Notice that the algorithms mentioned in the Network Protocol Services correspond to the same implementation of the algorithms described in the Cryptographic Library Services.

Service	Algorithms	Role	Access	Keys/CSP
Cryptographic Library Services				
Symmetric Encryption and Decryption	AES	User	Read	AES key
	Triple-DES	User	Read	Triple-DES key
RSA key generation	RSA, DRBG	User	Create	RSA public-private key
RSA digital signature generation and verification	RSA	User	Read	RSA public-private key
DSA key generation	DSA, DRBG	User	Create	DSA public-private key

Service	Algorithms	Role	Access	Keys/CSP
DSA domain parameter generation and verification	DSA	User	n/a	n/a
DSA digital signature generation and verification	DSA	User	Read	DSA public-private key
ECDSA key generation	ECDSA, DRBG	User	Create	ECDSA public-private key
ECDSA public key validation	ECDSA	User	Read	ECDSA public key
ECDSA signature generation and verification	ECDSA	User	Read	ECDSA public and private keys
Random number generation	DRBG	User	Read, Update	Entropy input string, seed, Internal state
Message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	User	n/a	n/a
Message authentication code (MAC)	HMAC	User	Read	HMAC key
	AES-GMAC	User	Read	AES key
	AES-CMAC	User	Read	AES key
	Triple-DES-CMAC	User	Read	Triple-DES key
Key wrapping	AES-KW, AES-KWP	User	Read	AES key
Key encapsulation	RSA	User	Read	RSA public and private keys
Diffie-Hellman Shared Secret Computation	KAS-FFC-SSC	User	Create, Read	Diffie-Hellman public and private keys, shared secret
Safe Primes Key Generation and Verification		User	Create, Read	Diffie-Hellman Domain Parameters and key pair
EC Diffie-Hellman Shared Secret Computation	KAS-ECC-SSC	User	Create, Read	EC Diffie-Hellman public and private keys, shared secret
Key Derivation Function	PBKDF2	User	Create, Read	Password, PBKDF2 derived key
	SSH KDF	User	Create, Read	Shared secret, SSH KDF derived key

Service	Algorithms	Role	Access	Keys/CSP
	HKDF	User	Create, Read	Shared secret, HKDF derived key
	KDF TLS	User	Create, Read	Shared secret, KDF TLS derived key
Network Protocols Services				
Transport Layer Security (TLS) network protocol v1.0, v1.1, v1.2 and v1.3	See Appendix A for the complete list of supported cipher suites.	User	Create, Read	AES or Triple-DES key, RSA, DSA or ECDSA public-private key, HMAC Key, shared secret, TLS master secret, Diffie-Hellman or EC Diffie-Hellman public and private keys
TLS extensions	n/a	User	Read	RSA, DSA or ECDSA public and private keys
Certificates management	n/a	User	Read	RSA, DSA or ECDSA public and private keys
Other FIPS-Related Services				
Show status	n/a	User	n/a	None
Zeroization	n/a	User	Zeroize	All CSPs
Self-Tests	AES, Triple-DES, SHS, HMAC, DSA, RSA, ECDSA, DRBG, Diffie-Hellman, EC Diffie-Hellman, TLS KDF, PBKDF, SSH KDF	User	n/a	None
Module installation	n/a	Crypto Officer	n/a	None
Module initialization	n/a	User	n/a	None

Table 6 - Services in FIPS mode of operation

The table below lists the services only available in non-FIPS mode of operation.

Service	Algorithms / Key sizes	Role	Access	Keys
Cryptographic Library Services				
Symmetric encryption and decryption	ARIA, Blowfish, Camellia, CAST, CAST5, ChaCha20, DES, RC2, RC4, SEED, SM4, Chacha20 and Poly1305	User	Read	Symmetric key

Service	Algorithms / Key sizes	Role	Access	Keys
Symmetric encryption and decryption	AES-XTS using 192 bit keys	User	Read	Symmetric key
Symmetric encryption	2-key Triple-DES listed in Table 11	User	Read	2-key Triple-DES key
Authenticated Encryption cipher for encryption and decryption	AES and SHA from multi-buffer or stitch ciphers listed in Table 11	User	Read	AES key, HMAC key
Asymmetric key generation using keys disallowed by [SP800-131A]	RSA, DSA, ECDSA listed in Table 11	User	Create	RSA, DSA or ECDSA public and private keys
Digital signature generation using message digest or keys disallowed by [SP800-131A].	RSA, DSA, ECDSA listed in Table 11	User	Read	RSA, DSA or ECDSA private keys
Digital signature verification using keys disallowed by [SP800-131A].	DSA listed in Table 11	User	Read	DSA public key
Digital signature generation and verification	SM2	User	Read	SM2 public and private keys
Key establishment using keys disallowed by [SP800-131A].	RSA, Diffie-Hellman, EC Diffie-Hellman listed in Table 11	User	Read	Diffie-Hellman, EC Diffie-Hellman or RSA public and private keys
Message digest	Blake2, MD4, MD5, RMD160, SM3	User	n/a	none
Message authentication code (MAC) using keys disallowed by [SP800-131A]	HMAC listed in Table 11, CMAC with 2-key Triple-DES	User	Read	HMAC key, 2-key Triple-DES key

Table 7 – Services in non-FIPS mode of operation

3.3. Algorithms

The algorithms implemented in the module are tested and validated by the CAVP for the operating environments listed in Table 3.

The Ubuntu 20.04 OpenSSL Cryptographic Module is compiled to use the support from the processor and assembly code for AES, SHA and GHASH operations to enhance the performance of the module. Different implementations can be invoked by using a processor capability mask in the operational environment. Please note that only one AES, SHA and/or GHASH implementation can be executed in runtime.

Notice that for the Transport Layer Security (TLS) and the Secure Shell (SSH) protocols, no parts of these protocols, other than the key derivation functions (KDF), have been tested by the CAVP.

3.3.1. Ubuntu 20.04 LTS 64-bit Running on Intel(R) Xeon(R) Gold 6226

On the platform that runs the Intel Xeon processor, the module supports the use of AES-NI, SSSE3 and strict assembler for AES implementation, the use of AVX2, AVX, SSSE3 and strict assembler for SHA implementation (SSSE3 implementation is only for SHA-1, SHA-224 and SHA-256), and the use of CLMUL instruction set and strict assembler for GHASH that is used for GCM mode. The module uses the most efficient implementation based on the processor's capability; this behavior can be also controlled through the use of the capability mask environment variable `OPENSSL_ia32cap`.

The following table shows all algorithms with the associated CAVP certificates for the different implementations validated in the module. See Appendix B for a description of each implementation.

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
AES	ECB	128, 192, 256	Data Encryption and Decryption	[FIPS197], [SP800-38A]	#A1519 #A1520 #A1521 #A1522 #A1527 #A1528 #A1529
	CBC, CTR, CFB1, CFB8, CFB128, OFB				#A1527 #A1528 #A1529
	CMAC		MAC Generation and Verification	[SP800-38B]	
	CCM		Data Encryption and Decryption	[SP800-38C]	
	GCM		Data Encryption and Decryption	[SP800-38D]	#A1535 #A1536 #A1537 #A1538 #A1539 #A1540 #A1541 #A1542 #A1543
	GMAC		Message Authentication Code		
	KW, KWP		Key Wrapping and Unwrapping	[SP800-38F]	#A1527 #A1528 #A1529
	XTS	128, 256	Data Encryption and Decryption for Data Storage	[SP800-38E]	
DRBG	**CTR_DRBG:** AES-128, AES-192, AES-256 with/without DF, without PR	n/a	Deterministic Random Bit Generation	[SP800-90A]	#A1527 #A1528 #A1529

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs

DSA	N/A	L=2048, N=224;
		L=2048, N=256;
		L=3072, N=256
			Key Pair Generation	[FIPS186-4]
				[FIPS180-4]
					#A1544
					#A1545
					#A1546
					#A1547

	SHA-224, SHA-256, SHA-384, SHA-512	L=2048, N=224;	Domain Parameter Generation Digital Signature Generation		
	SHA-256, SHA-384, SHA-512	L=2048, N=256; L=3072, N=256			
	SHA-224,	L=2048, N=224	Domain Parameter Verification		
	SHA-256	L=2048, N=256			
	SHA-256	L=3072, N=256			

	SHA-1
	SHA-224,
	SHA-256,
	SHA-384,
	SHA-512
		L=1024, N=160;
		L=2048, N=224;
		L=2048, N=256;
		L=3072, N=256
			Digital Signature Verification		

	SHA3-224,
	SHA3-256,
	SHA3-384,
	SHA3-512
		L=2048, N=224;
		L=2048, N=256;
		L=3072, N=256
			Digital Signature Generation	[FIPS186-4]
				[FIPS202]
					Vendor Affirmed

	SHA3-224,
	SHA3-256,
	SHA3-384,
	SHA3-512
		L=1024, N=160;
		L=2048, N=224;
		L=2048, N=256;
		L=3072, N=256;
			Digital Signature Verification		

KAS-ECC-SSC	ECC Ephemeral Unified scheme	P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571	EC Diffie-Hellman Shared Secret Computation and Key Agreement	[SP800-56Ar3]	#A1544 #A1545 #A1546 #A1547

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
KAS-FFC-SSC	FCC dhEphem scheme	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Diffie-Hellman Shared Secret Computation and Key Agreement	[SP800-56Ar3]	#A1550
Safe Prime Key Generation and Verification		ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	Key Pair Generation and Verification	[SP800-56Ar3]	
ECDSA	N/A	P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571	Key Pair Generation	[FIPS186-4] [FIPS180-4]	#A1544 #A1545 #A1546 #A1547
	SHA-224, SHA-256, SHA-384, SHA-512	P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571	Digital Signature Generation		
	SHA3-224 SHA3-256 SHA3-384 SHA3-512				#A1523 #A1524 #A1525
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-192, P-224, P-256, P-384, P-521, K-163, K-233, K-283, K-409, K-571, B-163, B-233, B-283, B-409, B-571	Public Key Verification		#A1544 #A1545 #A1546 #A1547
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-192, P-224, P-256, P-384, P-521, K-163, K-233, K-283, K-409, K-571, B-163, B-233, B-283, B-409, B-571	Digital Signature Verification		
	SHA3-224 SHA3-256 SHA3-384 SHA3-512				#A1523 #A1524 #A1525
HMAC	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	160, 224, 256, 384, 512 bits	Message Authentication Code	[FIPS198-1]	#A1544 #A1545 #A1546 #A1547
	SHA3-224 SHA3-256 SHA3-384 SHA3-512	224, 256, 384, 512 bits			#A1523 #A1524 #A1525

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
TLS v1.0, v1.1 and v1.2 KDF	SHA-256 SHA-384	N/A	Key Derivation	[SP800-135]	CVL. #A1544 #A1545 #A1546 #A1547
RSA	**X9.31**	2048, 3072, 4096	Key Pair Generation	[FIPS186-4] [FIPS180-4]	#A1544 #A1545 #A1546 #A1547
	X9.31 with SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital Signature Generation		
	X9.31 with SHA-1, SHA-256, SHA-384, SHA-512	1024, 2048, 3072, 4096	Digital Signature Verification		
	PKCS#1v1.5, PSS with SHA-224, SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital Signature Generation		
	PKCS#1v1.5, PSS with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024, 2048, 3072, 4096	Digital Signature Verification		
RSA	**PKCS#1v1.5, PSS** with SHA3-224, SHA3-256, SHA3-384, SHA3-512	2048, 3072, 4096	Digital Signature Generation	[FIPS186-4] [FIPS202]	Vendor Affirmed
	PKCS#1v1.5, PSS with SHA3-224, SHA3-256, SHA3-384, SHA3-512	1024, 2048, 3072, 4096	Digital Signature Verification		

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
SHS	SHA-1, SHA-224, SHA-256 SHA-384, SHA-512	n/a	Message Digest	[FIPS180-4]	#A1544 #A1545 #A1546 #A1547
SHA-3	SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	n/a	Message Digest	[FIPS202]	#A1523 #A1524 #A1525
Triple-DES	ECB	192 (two-key Triple-DES)	Data Decryption	[SP800-67] [SP800-38A]	#A1519 #A1520 #A1521 #A1522 #A1526
		192 (three-key Triple-DES)	Data Encryption and Decryption		
	CBC, CFB1, CFB8, CFB64, OFB				#A1526
	CMAC	192	MAC Generation and Verification	[SP800-67] [SP800-38B]	
PBKDF2	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Key length 128-4096 Increment 8	Password-based key derivation function	[SP800-132]	#A1544 #A1545 #A1546 #A1547
	SHA3-224, SHA3-256, SHA3-384, SHA3-512				#A1523 #A1524 #A1525
SSHKDF	SHA-1, SHA-256, SHA-384, SHA-512	AES-128, AES-192, AES-256, Triple-DES	SSH Key Derivation Function	[SP800-135]	CVL. #A1519 #A1520 #A1521 #A1522
KTS	AES-GCM	128, 256 bits	Key wrapping and unwrapping	[FIPS197] [SP800-38D]	#A1535 #A1536 #A1537 #A1538 #A1539 #A1540 #A1541 #A1542 #A1543
	AES-CCM	128, 256 bits		[FIPS197] [SP800-38C]	#A1527 #A1528

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
	AES-KW, KWP	128, 192, 256 bits		[FIPS197] [SP800-38F]	#A1529
	AES-CBC and HMAC-SHA1/ SHA-224/256/ 384/512	AES keys: 128, 256 bits HMAC keys: 112 bits and larger		[FIPS197] [SP800-38A] [FIPS 198-1] [FIPS180-4]	AES: #A1527 #A1528 #A1529 HMAC: #A1544 #A1545 #A1546 #A1547
KTS	Triple-DES-CBC and HMAC-SHA1 /224/256/ 384/512	Triple-DES keys: 192 bits HMAC keys: 112 bits and larger		[SP800-67] [FIPS 198-1] [FIPS180-4] [SP800-38F]	Triple-DES: #A1526 HMAC: #A1544 #A1545 #A1546 #A1547
KTS-IFC	RSA-OAEP with SHA-224, SHA- 256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Mod: 2048, 3072, 4096, 6144, 8192		[SP800-56Br2]	#A1544 #A1545 #A1546 #A1547
KDA	HKDF with SHA-224, SHA-256, SHA-384, SHA-512		Key Derivation	[SP800-56Cr1]	#A1516
ENT(NP)				[SP800-90B]	N/A

Table 8 - Cryptographic Algorithms for Intel(R) Xeon(R) Gold 6226 Processor

3.3.2. Ubuntu 20.04 LTS 64-bit Running on IBM z15

On the platform that runs the IBM Z processor, the module supports the use of CPACF or strict assembler for AES, SHA and GHASH implementations. If the CPACF is available in the operational environment, the module uses the support from CPACF automatically. If CPACF is unavailable, the module uses strict assembler implemented in the module.

The following table shows all algorithms with the associated CAVP certificates for the different implementations validated in the module. See Appendix B for a description of each implementation.

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
AES	ECB	128, 192, 256	Data Encryption and Decryption	[FIPS197] [SP800-38A]	#A1522 #A1528 #A1530 #A1533
	CBC, CTR, CFB1, CFB8, CFB128, OFB				#A1528 #A1530
	CMAC		MAC Generation and Verification	[SP800-38B]	
	CCM		Data Encryption and Decryption	[SP800-38C]	
	GCM		Data Encryption and Decryption	[SP800-38D]	#A1531 #A1548 #A1549
	GMAC		Message Authentication Code		
	KW, KWP		Key Wrapping and Unwrapping	[SP800-38F]	#A1528 #A1530
	XTS	128, 256	Data Encryption and Decryption for Data Storage	[SP800-38E]	
DRBG	**CTR_DRBG:** AES-128, AES-192, AES-256 with / without DF, without PR	n/a	Deterministic Random Bit Generation	[SP800-90A]	#A1528 #A1530

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs

DSA	N/A	L=2048, N=224;
		L=2048, N=256;
		L=3072, N=256
			Key Pair Generation	[FIPS186-4]
				[FIPS180-4]
					#A1532
					#A1547

	SHA-224, SHA-256, SHA-384, SHA-512	L=2048, N=224;	Domain Parameter Generation Digital Signature Generation		
	SHA-256, SHA-384, SHA-512	L=2048, N=256; L=3072, N=256			
	SHA-224,	L=2048, N=224	Domain Parameter Verification		
	SHA-256	L=2048, N=256			
	SHA-256	L=3072, N=256			
	SHA-1 SHA-224, SHA-256, SHA-384, SHA-512	L=1024, N=160 L=2048, N=224 L=2048, N=256; L=3072, N=256	Digital Signature Verification		

	SHA3-224,
	SHA3-256,
	SHA3-384,
	SHA3-512
		L=2048, N=224;
		L=2048, N=256;
		L=3072, N=256
			Digital Signature Generation	[FIPS186-4]
				[FIPS202]
					Vendor Affirmed

	SHA3-224, SHA3-256, SHA3-384, SHA3-512	L=1024, N=160 L=2048, N=224 L=2048, N=256; L=3072, N=256	Digital Signature Verification		
KAS-ECC-SSC	ECC Ephemeral Unified scheme	P-224, P-256, P-384, P-521 K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571 112 to 256 bits (see section 6.3 for key strength caveats)	EC Diffie-Hellman Shared Secret Computation and Key Agreement	[SP800-56Ar3]	#A1532 #A1547

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
KAS-FFC-SSC	FCC dhEphem scheme	ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 112 to 200 bits (see section 6.3 for key strength caveats)	Diffie-Hellman Shared Secret Computation and Key Agreement	[SP800-56Ar3]	#A1550
Safe Prime Key Generation and Verification		ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 112 to 200 bits	Key Pair Generation and Verification	[SP800-56Ar3]	#A1550
ECDSA	N/A	P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571	Key Pair Generation	[FIPS186-4] [FIPS180-4]	#A1532 #A1547
	SHA-224, SHA-256, SHA-384, SHA-512	P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571	Digital Signature Generation		
	SHA3-224 SHA3-256 SHA3-384 SHA3-512				#A1525 #A1534
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-192, P-224, P-256, P-384, P-521, K-163, K-233, K-283, K-409, K-571, B-163, B-233, B-283, B-409, B-571	Public Key Verification		#A1532 #A1547
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-192, P-224, P-256, P-384, P-521, K-163, K-233, K-283, K-409, K-571, B-163, B-233, B-283, B-409, B-571	Digital Signature Verification		#A1532 #A1547
	SHA3-224 SHA3-256 SHA3-384 SHA3-512				#A1525 #A1534 #A1547
HMAC	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	160, 224, 256, 384, 512 bits	Message Authentication Code	[FIPS198-1]	#A1532 #A1547
	SHA3-224	224, 256, 384, 512 bits			#A1525

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
	SHA3-256 SHA3-384 SHA3-512				#A1534
TLS v1.0, v1.1 and v1.2 KDF	SHA-256 SHA-384	N/A	Key Derivation	[SP800-135]	CVL. #A1532 #A1547
RSA	**X9.31**	2048, 3072, 4096	Key Pair Generation	[FIPS186-4] [FIPS180-4]	#A1532 #A1547
	X9.31 with SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital Signature Generation		
	X9.31 with SHA-1, SHA-256, SHA-384, SHA-512	1024, 2048, 3072, 4096	Digital Signature Verification		
	PKCS#1v1.5, PSS with SHA-224, SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital Signature Generation		
	PKCS#1v1.5, PSS with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024, 2048, 3072, 4096	Digital Signature Verification		
RSA	**PKCS#1v1.5, PSS** with SHA3-224, SHA3-256, SHA3-384, SHA3-512	2048, 3072, 4096	Digital Signature Generation	[FIPS186-4] [FIPS202]	Vendor Affirmed
	PKCS#1v1.5, PSS with SHA3-224, SHA3-256, SHA3-384, SHA3-512	1024, 2048, 3072, 4096	Digital Signature Verification		

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
SHS	SHA-1, SHA-224, SHA-256 SHA-384, SHA-512	n/a	Message Digest	[FIPS180-4]	#A1532 #A1547
SHA-3	SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	n/a	Message Digest	[FIPS202]	#A1525 #A1534
Triple-DES	ECB	192 (two-key Triple-DES)	Data Decryption	[SP800-67] [SP800-38A]	#A1522 #A1526 #A1533
		192 (three-key Triple-DES)	Data Encryption and Decryption		
	CBC, CFB1, CFB8, CFB64, OFB				#A1526
	CMAC	192	MAC Generation and Verification	[SP800-38B]	
PBKDF2	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	Key length 128-4096 Increment 8	Password-based key derivation function	[SP800-132]	#A1532 #A1547
	SHA3-224, SHA3-256, SHA3-384, SHA3-512				#A1525 #A1534
SSH KDF	SHA-1, SHA2-256, SHA2-384, SHA2-512	AES-128, AES-192, AES-256, Triple-DES	SSH Key Derivation Function	[SP800-135]	CVL. #A1522 #A1533
KTS	AES-GCM	128, 256 bits	Key wrapping and unwrapping	[FIPS197] [SP800-38D]	#A1531 #A1548 #A1549
	AES-CCM	128, 256 bits		[FIPS197] [SP800-38C]	#A1528 #A1530
	AES KW, KWP	128, 192, 256 bits		[FIPS197] [SP800-38F]	
	AES-CBC and HMAC-SHA1 / SHA-224 / SHA-256/ 384/512	AES keys: 128 or 256 bits HMAC keys: 112 bits and larger		[FIPS197] [SP800-38A] [FIPS 198-1] [FIPS180-4]	AES: #A1528 #A1530 HMAC: #A1532 #A1547

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
KTS Triple-DES	Triple-DES-CBC and HMAC-SHA1/224/256/ 384/512	Triple-DES keys: 192 bits HMAC keys: 112 bits and larger		[SP800-67] [FIPS 198-1] [FIPS180-4] [SP800-38F]	Triple-DES: #A1526 HMAC: #A1532 #A1547
KTS-IFC	RSA-OAEP with SHA-224, SHA- 256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	Mod: 2048, 3072, 4096, 6144, 8192		[SP800-56Br2]	#A1532 #A1547
KDA	HKDF with SHA-224, SHA-256, SHA-384, SHA-/512		Key Derivation PRF for TLSv1.3	[SP800-56Cr1]	#A1516
ENT(NP)				[SP800-90B]	N/A

Table 9 – Cryptographic Algorithms for IBM z15 Processor

3.3.3. Allowed Algorithms

The following table describes the non-Approved but allowed algorithms in FIPS mode:

Algorithm	Caveat	Use
Key Encapsulation using Encryption and Decryption Primitives with RSA PKCS#1v1.5 padding; keys equal or larger than 2048 bits up to 15360 or more.	Provides between 112 and 256 bits of encryption strength	Key Establishment; allowed per [FIPS140-2_IG] D.9
MD5[1]	n/a	Pseudo-random function (PRF) in TLS v1.0 and v1.1; allowed per [SP800-52r2]

Table 10 – FIPS-Allowed Cryptographic Algorithms

3.3.4. Non-Approved Algorithms

The table below shows the non-Approved cryptographic algorithms implemented in the module that are only available in non-FIPS mode.

Algorithm	Use
RSA with key size smaller than 2048 bits	Key Pair Generation, Digital Signature Generation, Key Encapsulation
DSA with key size smaller than 2048 bits or greater than 3072 bits	Key Pair Generation, Domain Parameters Generation, Digital Signature Generation
DSA with key size smaller than 1024 bits or greater than 3072 bits	Digital Signature Verification
ECDSA with curves P-192, K-163 or B-163 and non-NIST curves.	Key Pair Generation, Domain Parameters Generation, Digital Signature Generation
Diffie-Hellman with key size smaller than 2048 bits	Shared Secret Computation or key agreement
EC Diffie-Hellman with curves P-192, K-163 or B-163 and non-NIST curves.	Shared Secret Computation or key agreement
SHA-1	Digital Signature generation
HMAC with less than 112 bits key	Message Authentication Code
AES in XTS mode with 192-bit key	Data Encryption and Decryption
2-key Triple-DES	Data Encryption
CMAC with 2-key Triple-DES	Authenticated Data Encryption and Decryption

[1] According [SP800-52r2], MD5 is allowed to be used in TLS versions 1.0 and 1.1 as the hash function used in the PRF, as defined in [RFC2246] and [RFC4346].

Algorithm	Use
“Non-Compliant” multi-buffer or stitch ciphers using AES in CBC mode with 128 and 256-bit keys and HMAC-SHA-1 and SHA-256 (available only in Intel processors with AES-NI capability).	Authenticated Data Encryption and Decryption
ARIA, Blowfish, Camellia, CAST, CAST5, ChaCha20, DES, RC2, RC4, SEED, SM4	Data Encryption and Decryption
Blake2, MD4, MD5, RMD160, SM3	Message Digest
Chacha20 and Poly1305	Authenticated Data Encryption and Decryption
SM2	Digital Signature Generation and Verification

Table 11 – Non-Approved Cryptographic Algorithms

3.4. Operator Authentication

The module does not implement user authentication. The role of the user is implicitly assumed based on the service requested.

4. Physical Security

The module is comprised of software only and therefore this security policy does not make any claims on physical security.

5. Operational Environment

5.1. Applicability

The module operates in a modifiable operational environment per FIPS 140-2 level 1 specifications. The module runs on a commercially available general-purpose operating system executing on the hardware specified in Table 3 - Tested Platforms.

5.2. Policy

The operating system is restricted to a single operator; concurrent operators are explicitly excluded.

The application that requests cryptographic services is the single user of the module.

6. Cryptographic Key Management

The following table summarizes the Critical Security Parameters (CSPs) that are used by the cryptographic services implemented in the module:

Name	Generation	Entry and Output	Zeroization
AES keys	The key material is entered via API parameter or established during TLS handshake (as a KDF TLS derived key).	The key is passed into the module via API input parameters in plaintext.	`EVP_CIPHER_CTX_free()`, `EVP_CIPHER_CTX_reset()`
Triple-DES keys			`EVP_CIPHER_CTX_free()`, `EVP_CIPHER_CTX_reset()`
HMAC keys			`HMAC_CTX_free()`
RSA public and private keys	The public-private keys are generated using FIPS 186-4 Key Generation method, and the random value used in the key generation is generated using SP800-90A DRBG.	The key is passed into the module via API input parameters in plaintext. The key is passed out of the module via API output parameters in plaintext.	`RSA_free()`
DSA public and private keys			`DSA_free()`
ECDSA public and private keys			`EC_KEY_free()`
Diffie-Hellman public and private keys	The public-private keys are generated using SP800-56A Safe Primes Key Generation method, and the random value used in the key generation is generated using SP800-90A DRBG.	The key is passed into the module via API input parameters in plaintext. The key is passed out of the module via API output parameters in plaintext.	`DH_free()`
EC Diffie-Hellman public and private keys	The public-private keys are generated using the FIPS 186-4 Key Generation method, and the random value used in the key generation is generated using SP800-90A DRBG.		`EC_KEY_free()`
Shared secret	Generated during the Diffie-Hellman or EC Diffie-Hellman key agreement.	None	`SSL_free()`, `SSL_clear()`
	Generated from the SP800-90A DRBG when module acts as a TLS client, for RSA cipher suites.	Entry: if received by module as TLS server, wrapped with server's public RSA key; otherwise no entry.	

		Output: if generated by module as TLS client, wrapped with server's public RSA key; otherwise, no output.	
TLS master secret	Derived from shared secret using TLS KDF.	None	`SSL_free()`, `SSL_clear()`
Entropy input string and seed	Obtained from the NRBG.	None	`RAND_DRBG_free()`
DRBG internal state (V, Key)	During DRBG initialization.	None	`RAND_DRBG_free()`
PBKDF2 Password	N/A	The password is passed into the module via API input parameters in plaintext.	`OPENSSL_cleanse()`
PBKDF2 Derived Key	Derived using the SP800-132 KDF	The key is passed out of the module via API output parameters in plaintext.	`OPENSSL_cleanse()`
SSH KDF Derived Key	Derived using the SP800-135 SSH KDF.	The key is passed out of the module via API output parameters in plaintext.	`EVP_PKEY_CTX_free()`
HKDF Derived Key	Derived using the SP800-56Cr1 KDF	None	`EVP_PKEY_CTX_free()`

Table 12 – Life cycle of Critical Security Parameters (CSP)

The following sections describe how CSPs, in particular cryptographic keys, are managed during its life cycle.

6.1. Random Number Generation

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of seeds for asymmetric keys, and server and client random numbers for the TLS protocol. In addition, the module provides a Random Number Generation service to calling applications.

The DRBG supports the CTR_DRBG mechanisms with key sizes and modes specified in Table 7 and Table 8. The DRBG is initialized during module initialization; the module loads by default the DRBG using CTR_DRBG with AES-256 and derivation function without prediction resistance. A different DRBG mechanism can be chosen through an API function call.

The module uses a Non-Deterministic Random Bit Generator (NRBG) provided by the operational environment to obtain entropy for the DRBG; the NRBG is located within the module's physical boundary but outside of the module's logical boundary. The NRBG uses CPU jitter as a physical noise source and is compliant with [SP800-90B]; the NRBG is marked as ENT(NP) in the certificate.

The module makes use of `getrandom()` system call, to access the output of NRBG which is used for seeding the DRBG. The NRBG provides at least 256 bits of entropy to the DRBG during initialization (seed) and reseeding (reseed).

The module performs DRBG health tests as defined in section 11.3 of [SP800-90A].

Note: According to Linux man pages [LMAN] `random`(4) and `getrandom`(2), the `getrandom()` system call is prohibited until the Linux kernel has initialized its NRBG during the kernel boot-up. This blocking behavior is only observed during boot time. When defining systemd units using OpenSSL, the Crypto Officer should ensure that these systemd units do not block the general systemd operation as otherwise the entire boot process may be blocked based on the getrandom blocking behavior.

6.2. Key Generation

The Module provides an SP800-90A-compliant DRBG for creation of key components of asymmetric keys, and random number generation.

The Key Generation methods implemented in the module for Approved services in FIPS mode is compliant with [SP800-133] (vendor affirmed).

For generating RSA, DSA and ECDSA keys the module implements asymmetric key generation services compliant with [FIPS186-4]. A seed (i.e. the random value) used in asymmetric key generation is directly obtained from the [SP800-90A] DRBG.

The public and private keys used in the EC Diffie-Hellman key agreement schemes are generated internally by the module using the ECDSA key generation method compliant with [FIPS186-4] and [SP800-56Ar3]. The Diffie-Hellman key agreement scheme is also compliant with [SP800-56Ar3], and generates keys using safe primes defined in RFC7919 and RFC3526, as described in the next section.

6.3. Key Agreement / Key Transport / Key Derivation

The module provides Diffie-Hellman and EC Diffie-Hellman shared secret computation, which consists of SP800-56Ar3 Diffie-Hellman and EC Diffie-Hellman primitives. These security functions are approved per FIPS 140-2 IG D.8 Scenario X1(1).

The module also provides Diffie-Hellman and EC Diffie-Hellman key agreement schemes that are used as part of the TLS. Specifically, the key agreement scheme consists of SP800-56Ar3 Diffie-Hellman and EC Diffie-Hellman primitives (i.e. KAS-SSC) and SP800-135 TLS KDF (CVL) listing in IG G.20 per FIPS 140-2 IG D.8 Scenario X1(2).

The module now exclusively supports SP800-56Ar3 shared secret computation and key agreement schemes in FIPS mode of operation. For Diffie-Hellman, the module supports the use of safe primes from RFC 7919 for domain parameters and key generation that is used by the TLS key agreement implemented by the module. The module also supports the use of safe primes from RFC3526 that can be used by the IKE key agreement implemented in the Strongswan module. Note that the current module only implements the shared secret computation of safe primes used in IKE RFC3526 and not the entire IKE key agreement:

IKEv2 (RFC 3526)	TLS (RFC 7919)
MODP-2048 (ID=14)	ffdhe2048 (ID = 256)

MODP-3072 (ID=15)	ffdhe3072 (ID = 257)
MODP-4096 (ID=16)	ffdhe4096 (ID = 258)
MODP-6144 (ID=17)	ffdhe6144 (ID = 259)
MODP-8192 (ID=18)	ffdhe8192 (ID = 260)

The module provides key wrapping using the AES with KW and KWP modes.

The module also provides key wrapping in the context of using the TLS protocol to send and receive key material in the payload. The key wrapping methods are provided by the TLS record layer either using an approved authenticated encryption mode (i.e. AES GCM, AES-CCM), or a combination method including symmetric encryption (i.e. AES or Triple-DES in CBC mode) and an approved authentication method (i.e. HMAC with SHA); the method depends on the TLS cipher suite negotiated during the TLS handshake. All methods provided by the TLS cipher suites included in Appendix A are approved key transport methods according to IG D.9.

The module also provides key encapsulation using the following methods:

- RSA public key encryption and private key decryption with PKCS#1v1.5 padding. This method is an allowed method per IG D.9 and is used as part of the TLS protocol key exchange.
- RSA public key encryption and private key decryption (KTS-IFC) with OEAP padding.

According to Table 2: Comparable strengths in [SP800-57], the key sizes of AES, Triple-DES, RSA, Diffie-Hellman and EC Diffie-Hellman provides the following security strength in FIPS mode of operation:

- AES KW and KWP key wrapping, provides between 128 and 256 bits of encryption strength.
- AES GCM and CCM key wrapping (as part of TLS protocol) provides 128 or 256 bits of encryption strength.
- Key wrapping using AES encryption in CBC mode with HMAC (as part of TLS protocol) provides 128 or 256 bits of encryption strength.
- Key wrapping using Triple-DES encryption in CBC mode with HMAC (as part of TLS protocol) provides 112 bits of encryption strength.
- RSA key wrapping[2] with PKCS#1v1.5 padding provides between 112 and 256 bits of encryption strength.
- RSA key wrapping with OAEP padding provides between 112 and 200 bits of encryption strength.
- Diffie-Hellman shared secret computation and key agreement provide between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman shared secret computation and key agreement provide between 112 and 256 bits of encryption strength.

The module supports the following key derivation methods according to [SP800-135]:

[2] “Key wrapping” is used instead of “key encapsulation” to show how the algorithm will appear in the certificate per IG G.13.

- KDF for the TLS protocol. The module implements the pseudo-random functions (PRF) for TLSv1.0/1.1 and TLSv1.2.
- HKDF for the TLS protocol, compliant with SP800-56Cr1. The module implements the pseudo-random functions (PRF) for TLSv1.3.
- KDF for the SSH using SHA-1, SHA-256, SHA-384, SHA-512.

The module also supports password-based key derivation (PBKDF). The implementation is compliant with option 1a of [SP-800-132]. Keys derived from passwords or passphrases using this method can only be used in storage applications.

Note: As the module supports the size of RSA key pair greater than 2048 bits up to 15360 bits or more, the encryption strength 256 bits is claimed for RSA key encapsulations.

6.4. Key Entry / Output

The module does not support manual key entry or intermediate key generation key output. The keys are provided to the module via API input parameters in plaintext form and output via API output parameters in plaintext form. This is allowed by [FIPS140-2_IG] IG 7.7, according to the “CM Software to/from App Software via GPC INT Path” entry on the Key Establishment Table.

6.5. Key / CSP Storage

Symmetric keys, HMAC keys, public and private keys are provided to the module by the calling application via API input parameters, and are destroyed by the module when invoking the appropriate API function calls.

The module does not perform persistent storage of keys. The keys and CSPs are stored as plaintext in the RAM. The only exception is the HMAC key used for the Integrity Test, which is stored in the module and relies on the operating system for protection.

6.6. Key / CSP Zeroization

The memory occupied by keys is allocated by regular memory allocation operating system calls. The application is responsible for calling the appropriate zeroization functions provided in the module's API listed in Table 12 . Calling the `SSL_free()` and `SSL_clear()` will zeroize the keys and CSPs used in the TLS protocol and also invoke the module’s API listed in Table 12 automatically to zeroize the keys and CSPs. The zeroization functions overwrite the memory occupied by keys with “zeros” and deallocate the memory with the regular memory deallocation operating system call.

7. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test platforms listed in Table 3 - Tested Platforms have been tested and found to conform to the EMI/EMC requirements specified by 47 Code of Federal Regulations, FCC PART 15, Subpart B, Unintentional Radiators, Digital Devices, Class A (i.e., Business use). These devices are designed to provide reasonable protection against harmful interference when the devices are operated in a commercial environment. They shall be installed and used in accordance with the instruction manual.

8. Self-Tests

FIPS 140-2 requires that the module perform power-up tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. In addition, some functions require continuous testing of the cryptographic functionality, such as the asymmetric key generation. If any self-test fails, the module returns an error code and enters the error state. No data output or cryptographic operations are allowed in error state.

See section 9.2.9 for descriptions of possible self-test errors and recovery procedures.

8.1. Power-Up Tests

The module performs power-up tests when the module is loaded into memory, without operator intervention. Power-up tests ensure that the module is not corrupted and that the cryptographic algorithms work as expected.

While the module is executing the power-up tests, services are not available, and input and output are inhibited. The module is not available for use by the calling application until the power-up tests are completed successfully.

If any power-up test fails, the module returns the error code listed in Table 18 and displays the specific error message associated with the returned error code, and then enters error state. The subsequent calls to the module will also fail - thus no further cryptographic operations are possible. If the power-up tests complete successfully, the module will return 1 in the return code and will accept cryptographic operation service requests.

8.1.1. Integrity Tests

The integrity of the module is verified by comparing an HMAC-SHA-256 value calculated at run time with the HMAC value stored in the .hmac file that was computed at build time for each software component of the module. If the HMAC values do not match, the test fails and the module enters the error state.

8.1.2. Cryptographic Algorithm Tests

The module performs self-tests on all FIPS-Approved cryptographic algorithms supported in the Approved mode of operation, using the Known Answer Tests (KAT), Pair-wise Consistency Tests (PCT), as well as DRBG health tests shown in the following table:

Algorithm	Power-Up Tests
AES	• KAT AES ECB mode with 128-bit key, encryption • KAT AES ECB mode with 128-bit key, decryption
Triple DES	• KAT three-key Triple-DES ECB mode, encryption • KAT three-key Triple-DES ECB mode, decryption
CMAC	• KAT AES CMAC with 128, 192 and 256 bit keys, MAC generation • KAT three-key Triple-DES, MAC generation

Algorithm	Power-Up Tests
SHS	• KAT SHA-1 and SHA-512 • KAT SHA3-256, SHA3-512 • KAT SHAKE128 and SHAKE256 Note: SHA-224 and SHA-384 are not required per IG 9.4. SHA-256 is covered in the Integrity Test which is allowed per IG 9.3.
HMAC	Note: HMAC is covered in the Integrity Test which is allowed per IG 9.3 and 9.4
DSA	• PCT DSA with L=2048, N=256 and SHA-256
ECDSA	• PCT ECDSA with P-256 and SHA-256 • PCT ECDSA with K-233 and SHA-256
RSA	• KAT RSA with 2048-bit key, PKCS#1v1.5 scheme and SHA-256, signature generation • KAT RSA with 2048-bit key, PKCS#1v1.5 scheme and SHA-256, signature verification • KAT RSA with 2048-bit key, public key encryption • KAT RSA with 2048-bit key, private key decryption
DRBG	• KAT CTR_DRBG with AES with 256 bit key, without PR, with DF • KAT CTR_DRBG with AES with 256 bit key, without PR, without DF • Health Test
EC Diffie-Hellman	• Primitive "Z" Computation KAT with P-256 curve
Diffie-Hellman	• Primitive "Z" Computation KAT with 3072-bit key
TLS KDF	• KAT KDF for TLSv1.0 and v1.1 • KAT KDF for TLSv1.2
SSH KDF	• KAT using HMAC-SHA-256
PBKDF2	• KAT using SHA-256
HKDF	• KAT KDF for TLSv1.3

Table 13 – Self-Tests

For the KAT, the module calculates the result and compares it with the known value. If the answer does not match the known answer, the KAT is failed and the module enters the Error state.

For the PCT, if the signature generation or verification fails, the module enters the Error state. As described in section 3.3, only one AES or SHA implementation is available at run-time.

The KATs cover the different cryptographic implementations available in the operating environment.

8.2. On-Demand Self-Tests

On-Demand self-tests can be invoked by powering-off and reloading the module which cause the module to run the power-up tests again. During the execution of the on-demand self-tests, services are not available and no data output or input is possible.

8.3. Conditional Tests

The module performs conditional tests on the cryptographic algorithms, using the Pair-wise Consistency Tests (PCT), shown in the following table:

Algorithm	Conditional Test
DSA key generation	• PCT using SHA-256, signature generation and verification.
ECDSA key generation	• PCT using SHA-256, signature generation and verification.
RSA key generation	• PCT using SHA-256, signature generation and verification. • PCT using encryption and decryption

Table 14 – Conditional Tests

9. Guidance

9.1. Crypto Officer Guidance

The binaries of the module are contained in the Ubuntu packages for delivery. The Crypto Officer shall follow this Security Policy to configure the operational environment and install the module to be operated as a FIPS 140-2 validated module.

The following Ubuntu packages contain the FIPS validated module:

Processor Architecture	Ubuntu packages
x86_64	`libssl1.1-1.1.1f_1ubuntu2.fips.7.1_amd64.deb` `libssl1.1-hmac-1.1.1f_1ubuntu2.fips.7.1_amd64.deb`
z15	`libssl1.1-1.1.1f_1ubuntu2.fips.7.1_s390.deb` `libssl1.1-hmac-1.1.1f_1ubuntu2.fips.7.1_s390.deb`

Table 15 – Ubuntu packages

The `libssl-doc_1.1.1f_1ubuntu2.fips.7.1_all.deb` Ubuntu package contains the man pages for the module.

Note: The prelink is not installed on Ubuntu, by default. For proper operation of the in-module integrity verification, the prelink should be disabled.

9.1.1. Operating Environment Configurations

To configure the operating environment to support FIPS, the following shall be performed with the root privilege:

Install the following `linux-fips and fips-initramfs` Ubuntu packages depending on the target operational environment:

Processor Architecture	Ubuntu packages

x86_64	`fips-initramfs-generic_0.0.15+generic1_amd64.deb`
	`linux-image-5.4.0-1024.28+recert1-fips 5.4.0-1024.28+recert1_amd64.deb`

z15	`fips-initramfs-generic_0.0.15+generic1_s390.deb`
	`linux-image-5.4.0-1024.28+recert1-fips 5.4.0-1024.28+recert1_s390.deb`

Table 16 – Prerequisite Ubuntu packages

(1) Add `fips=1` to the kernel command line.

- For x86_64 systems, create the file `/etc/default/grub.d/99-fips.cfg` with the content: `GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT fips=1".`
- For z systems, edit `/etc/zipl.conf` file and append the "`fips=1`" in the parameters line for the specified boot image.

(2) If /boot resides on a separate partition, the kernel parameter `bootdev=UUID=<UUID of partition>` must also be appended in the aforementioned grub or zipl.conf file. Please see the following **Note** for more details.

(3) Update the boot loader.

- Run the `update-grub` command (not necessary on S390X systems with zipl loader).

(4) Run `reboot` to reboot the system with the new settings.

Now, the operating environment is configured to support FIPS operation. The Crypto Officer should check the existence of the file, `/proc/sys/crypto/fips_enabled`, and that it contains "1". If the file does not exist or does not contain "1", the operating environment is not configured to support FIPS and the module will not operate as a FIPS validated module properly.

Note: If /boot resides on a separate partition, the kernel parameter `bootdev=UUID=<UUID of partition>` must be supplied. The partition can be identified with the command `df /boot`. For example:

```
$ df /boot

Filesystem     1K-blocks   Used Available Use% Mounted on

/dev/sdb2         241965 127948    101525  56% /boot
```

The UUID of the /boot partition can be found by using the command `grep /boot /etc/fstab`. For example:

```
$ grep /boot /etc/fstab

# /boot was on /dev/sdb2 during installation

UUID=cec0abe7-14a6-4e72-83ba-b912468bbb38 /boot    ext2    defaults     0     2
```

Then, the UUID shall be added in the `/etc/default/grub.d/99-fips.cfg`. For example:

```
GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT fips=1 bootdev=UUID=Insert boot UUID"
```

Optionally, the following packages may be also installed:

- The openssl Ubuntu package provides the command line interface.
- The libssl1.1-dev package provides include files that are necessary to build applications using the module.

9.1.2. Module Installation

Canonical distributes the module via Personal Package Archives (PPA), whose access is granted to users with a valid subscription. In order to obtain a subscription and download the FIPS validated version of the module, please email "sales@canonical.com" or contact a Canonical representative, https://www.ubuntu.com/contact-us. Canonical provides specific instructions to configure the system to get access to the corresponding PPA.

Once the operating environment is configured following the instructions provided in section 9.1.1, and configuration to access the PPA is complete, the Crypto Officer can install the Ubuntu packages containing the module listed in Table 15 using the Advanced Package Tool (APT) with the following command line:

```
$ sudo apt-get install libssl1.1 libssl1.1-hmac libssl-doc
```

All the Ubuntu packages are associated with hashes for integrity check. The integrity of the Ubuntu package is automatically verified by the packing tool during the installation of the module. The Crypto Officer shall not install the package if the integrity fails.

9.2. User Guidance

In order to run in FIPS mode, the module must be operated using the FIPS Approved services, with their corresponding FIPS Approved and FIPS allowed cryptographic algorithms provided in this Security Policy (see section 3.2 Services). In addition, key sizes must comply with [SP800-131A].

9.2.1. TLS

The module implements TLS versions 1.0, 1.1, 1.2 and 1.3. The TLS protocol implementation provides both server and client sides. In order to operate in FIPS mode, digital certificates used for server and client authentication shall comply with the restrictions of key size and message digest algorithms imposed by [SP800-131A].

9.2.2. AES-GCM's IV

In case the module's power is lost and then restored, the key used for the AES GCM encryption or decryption shall be redistributed.

The nonce_explicit part of the IV does not exhaust the maximum number of possible values for a given session key. The design of the TLS protocol in this module implicitly ensures that the nonce_explicit, or counter portion of the IV will not exhaust all of its possible values.

The AES GCM IV generation is in compliance with the [RFC5288] and shall only be used for the TLS protocol version 1.2 to be compliant with [FIPS140-2_IG] IG A.5, provision 1 ("TLS protocol IV generation"). Moreover, the module is compliant with Section 3.3.1 of [SP800-52r2].

9.2.3. AES-XTS

The AES algorithm in XTS mode can be only used for the cryptographic protection of data on storage devices, as specified in [SP800-38E]. The length of a single data unit encrypted with the XTS-AES shall not exceed 2^{20} AES blocks that is 16MB of data.

To meet the requirement in [FIPS140-2_IG] A.9, the module implements a check to ensure that the two AES keys used in XTS-AES algorithm are not identical.

Note: AES-XTS shall be used with 128 and 256-bit keys only. AES-XTS with 192-bit keys is not an Approved service.

9.2.4. Triple-DES

[SP800-67] imposes a restriction on the number of 64-bit block encryptions performed under the same three-key Triple-DES key.

When the three-key Triple-DES is generated as part of a recognized IETF protocol, the module is limited to 2^{20} 64-bit data block encryptions. This scenario occurs in the following protocols:

- Transport Layer Security (TLS) versions 1.1 and 1.2, conformant with [RFC5246]
- Secure Shell (SSH) protocol, conformant with [RFC4253]
- Internet Key Exchange (IKE) versions 1 and 2, conformant with [RFC7296]

In any other scenario, the module cannot perform more than 2^{16} 64-bit data block encryptions.

The user is responsible for ensuring the module's compliance with this requirement.

9.2.5. Key derivation using SP800-132 PBKDF

The module provides password-based key derivation (PBKDF), compliant with SP800-132. The module supports option 1a from section 5.4 of [SP800-132], in which the Master Key (MK) or a segment of it is used directly as the Data Protection Key (DPK). In accordance to [SP800-132], the following requirements shall be met.

- Derived keys shall only be used in storage applications. The Master Key (MK) shall not be used for other purposes. The length of the MK or DPK shall be of 112 bits or more.
- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the SP800-90A DRBG.
- The iteration count shall be selected as large as possible; as long as the time required to generate the key using the entered password is acceptable for the users. The minimum value shall be 1000.
- Passwords or passphrases, used as an input for the PBKDF, shall not be used as cryptographic keys.
- The length of the password or passphrase shall be of at least 20 characters, and shall consist of lower-case, upper-case and numeric characters. The probability of guessing the value is estimated to be 1/62^20 = 10^36, which is less than 2^112.

The calling application shall also observe the rest of the requirements and recommendations specified in [SP800-132].

9.2.6. API Functions

Passing "0" to the `FIPS_mode_set()` API function is prohibited.

Executing the `CRYPTO_set_mem_functions()` API function is prohibited as it performs like a null operation in the module.

The FIPS required selftests that run during power-on of the module will render OPENSSL_init_crypto() useless in application code since it cannot be run first.

Calling DH_generate_parameters_ex() will return an error in FIPS mode since the module only supports safe primes Diffie-Hellman parameters. When generating a key pair using some safe primes domain parameters, the NID of the safe prime group shall be used. DH_check(), DH_check_ex(), DH_check_params(), DH_check_params_ex() will only check that an appropriate safe prime NID has been set when in FIPS mode.

9.2.7. Use of ciphers

The following ciphers (usually obtained by calling the `EVP_get_cipherbyname()` function) use multiblock implementations of the AES, HMAC and SHA algorithms that are not validated by the CAVP; therefore, they cannot be used in FIPS mode of operation.

Cipher Name	NID
AES-128-CBC-HMAC-SHA1	NID_aes_128_cbc_hmac_sha1
AES-256-CBC-HMAC-SHA1	NID_aes_256_cbc_hmac_sha1
AES-128-CBC-HMAC-SHA256	NID_aes_128_cbc_hmac_sha256

AES-256-CBC-HMAC-SHA256	NID_aes_256_cbc_hmac_sha256

Table 17- Ciphers not allowed in FIPS mode of operation

9.2.8. Environment Variables

OPENSSL_ENFORCE_MODULUS_BITS

As described in [SP800-131A], less than 2048 bits of DSA and RSA key sizes are disallowed by NIST. Setting the environment variable `OPENSSL_ENFORCE_MODULUS_BITS` can restrict the module to only generate the acceptable key sizes of RSA and DSA. If the environment variable is set, the module can generate 2048 or 3072 bits of RSA key, and at least 2048 bits of DSA key.

OPENSSL_FIPS_NON_APPROVED_MD5_ALLOW

As described in [SP800-52r2], MD5 is allowed to be used in TLS versions 1.0 and 1.1 as the hash function used in the PRF, as defined in [RFC2246] and [RFC4346]. By default, the module disables the MD5 algorithm. Setting the environment variable `OPENSSL_FIPS_NON_APPROVED_MD5_ALLOW` can enable the MD5 algorithm in the module. The MD5 algorithm shall not be used for other purposes other than the PRF in TLS version 1.0 and 1.1.

9.2.9. Handling FIPS Related Errors

When the module fails any self-test, the module will return an error code to indicate the error and enters error state that any further cryptographic operation is inhibited. Errors occurred during the self-tests and conditional tests transition the module into an error state. Here is the list of error codes when the module fails any self-test, in error state or not supported in FIPS mode:

Error Events	Error Codes/Messages
When the Integrity Test fails at the power-up	FIPS_R_FINGERPRINT_DOES_NOT_MATCH (111) “fingerprint does not match”
When the AES, Triple-DES, SHA-1, SHA-512 KAT fails at the power-up	FIPS_R_SELFTEST_FAILED (134) “selftest failed”
When the KAT for RSA fails, or the PCT for ECDSA or DSA fails at the power-up	FIPS_R_TEST_FAILURE (137) “test failure”
When the KAT of DRBG fails at the power-up	FIPS_R_NOPR_TEST1_FAILURE (145) “nopr test1 failure”
When the KAT of Diffie-Hellman or EC Diffie-Hellman fails at the power-up	0
When the new generated RSA, DSA or ECDSA key pair fails the PCT	FIPS_R_PAIRWISE_TEST_FAILED (127) “pairwise test failed”
When the SSLv2.0 or SSL v3.0 are called	SSL_R_ONLY_TLS_ALLOWED_IN_FIPS_MODE (297) “only tls allowed in fips mode”
When the module is in error state and any cryptographic operation is called	FIPS_R_FIPS_SELFTEST_FAILED (115) “fips selftest failed”

	FIPS_R_SELFTEST_FAILED (134) “selftest failed”
When the AES key and tweak keys for XTS-AES are the same	EVP_R_XTS_DUPLICATED_KEYS (183) “xts duplicated keys”

Table 18 – Error Events, Error Codes and Error Messages

These errors are reported through the regular `ERR` interface of the modules and can be queried by functions such as `ERR_get_error()`. See the OpenSSL man pages for the function description.

When the module is in the error state and the application calls a crypto function of the module that cannot return an error in normal circumstances (void return functions), the error message: “`OpenSSL internal error, assertion failed: FATAL FIPS SELFTEST FAILURE`” is printed to `stderr` and the application is terminated with the `abort()` call. The only way to recover from this error is to restart the application. If the failure persists, the module must be reinstalled.

10. Mitigation of Other Attacks

10.1. Blinding Against RSA Timing Attacks

RSA is vulnerable to timing attacks. In a configuration where attackers can measure the time of RSA decryption or signature operations, blinding must be used to protect the RSA operation from that attack.

The module provides the API functions `RSA_blinding_on()` and `RSA_blinding_off()` to turn the blinding on and off for RSA. When the blinding is on, the module generates a random value to form a blinding factor in the RSA key before the RSA key is used in the RSA cryptographic operations.

Please note that the DRBG must be seeded prior to calling `RSA_blinding_on()` to prevent the RSA Timing Attack.

10.2. Weak Triple-DES Keys Detection

The module implements the `DES_set_key_checked()` for checking the weak Triple-DES key and the correctness of the parity bits when the Triple-DES key is going to be used in Triple-DES operations. The checking of the weak Triple-DES key is implemented in the API function `DES_is_weak_key()` and the checking of the parity bits is implemented in the API function `DES_check_key_parity()`. If the Triple-DES key does not pass the check, the module will return -1 to indicate the parity check error and -2 if the Triple-DES key matches to any value listed below:

```
static const DES_cblock weak_keys[NUM_WEAK_KEY] = {
    /* weak keys */
    {0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01},
    {0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE},
    {0x1F, 0x1F, 0x1F, 0x1F, 0x0E, 0x0E, 0x0E, 0x0E},
    {0xE0, 0xE0, 0xE0, 0xE0, 0xF1, 0xF1, 0xF1, 0xF1},
    /* semi-weak keys */
    {0x01, 0xFE, 0x01, 0xFE, 0x01, 0xFE, 0x01, 0xFE},
    {0xFE, 0x01, 0xFE, 0x01, 0xFE, 0x01, 0xFE, 0x01},
    {0x1F, 0xE0, 0x1F, 0xE0, 0x0E, 0xF1, 0x0E, 0xF1},
    {0xE0, 0x1F, 0xE0, 0x1F, 0xF1, 0x0E, 0xF1, 0x0E},
    {0x01, 0xE0, 0x01, 0xE0, 0x01, 0xF1, 0x01, 0xF1},
    {0xE0, 0x01, 0xE0, 0x01, 0xF1, 0x01, 0xF1, 0x01},
    {0x1F, 0xFE, 0x1F, 0xFE, 0x0E, 0xFE, 0x0E, 0xFE},
    {0xFE, 0x1F, 0xFE, 0x1F, 0xFE, 0x0E, 0xFE, 0x0E},
    {0x01, 0x1F, 0x01, 0x1F, 0x01, 0x0E, 0x01, 0x0E},
    {0x1F, 0x01, 0x1F, 0x01, 0x0E, 0x01, 0x0E, 0x01},
    {0xE0, 0xFE, 0xE0, 0xFE, 0xF1, 0xFE, 0xF1, 0xFE},
    {0xFE, 0xE0, 0xFE, 0xE0, 0xFE, 0xF1, 0xFE, 0xF1}
};
```

Appendix A. TLS Cipher Suites

The module supports the following cipher suites for the TLS protocol. Each cipher suite defines the key exchange algorithm, the bulk encryption algorithm (including the symmetric key size) and the MAC algorithm.

Cipher Suite	Reference
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_RSA_WITH_AES_128_CBC_SHA	RFC3268
TLS_RSA_WITH_AES_256_CBC_SHA	RFC3268
TLS_DH_DSS_WITH_AES_128_CBC_SHA	RFC3268
TLS_DH_DSS_WITH_AES_256_CBC_SHA	RFC3268
TLS_DH_RSA_WITH_AES_128_CBC_SHA	RFC3268
TLS_DH_RSA_WITH_AES_256_CBC_SHA	RFC3268
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	RFC3268
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	RFC3268
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	RFC3268
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	RFC3268
TLS_DH_anon_WITH_AES_128_CBC_SHA	RFC3268
TLS_DH_anon_WITH_AES_256_CBC_SHA	RFC3268
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	RFC4492
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	RFC4492
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDH_anon_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDH_anon_WITH_AES_256_CBC_SHA	RFC4492
TLS_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_RSA_WITH_AES_128_GCM_SHA256	RFC5288

Cipher Suite	Reference
TLS_RSA_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DH_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DH_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DH_RSA_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DH_RSA_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DH_DSS_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DH_DSS_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DH_DSS_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DH_DSS_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	RFC5288
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	RFC5289
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	RFC5289
TLS_DH_anon_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DH_anon_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DH_anon_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DH_anon_WITH_AES_256_GCM_SHA384	RFC5288
RSA_WITH_AES_128_CCM	RFC5116
RSA_WITH_AES_256_CCM	RFC5116
DHE_RSA_WITH_AES_128_CCM	RFC5116

Cipher Suite	Reference
DHE_RSA_WITH_AES_256_CCM	RFC5116
RSA_WITH_AES_128_CCM_8	RFC6655
RSA_WITH_AES_256_CCM_8	RFC6655
DHE_RSA_WITH_AES_128_CCM_8	RFC6655
DHE_RSA_WITH_AES_256_CCM_8	RFC6655
ECDHE_ECDSA_WITH_AES_128_CCM	RFC7251
ECDHE_ECDSA_WITH_AES_256_CCM	RFC7251
ECDHE_ECDSA_WITH_AES_128_CCM_8	RFC7251
ECDHE_ECDSA_WITH_AES_256_CCM_8	RFC7251
TLS_PSK_WITH_3DES_EDE_CBC_SHA	RFC4279
TLS_PSK_WITH_AES_128_CBC_SHA	RFC4279
TLS_PSK_WITH_AES_256_CBC_SHA	RFC4279
DHE_PSK_WITH_3DES_EDE_CBC_SHA	RFC4279
DHE_PSK_WITH_AES_128_CBC_SHA	RFC4279
DHE_PSK_WITH_AES_256_CBC_SHA	RFC4279
RSA_PSK_WITH_3DES_EDE_CBC_SHA	RFC4279
RSA_PSK_WITH_AES_128_CBC_SHA	RFC4279
RSA_PSK_WITH_AES_256_CBC_SHA	RFC4279
PSK_WITH_AES_128_GCM_SHA256	RFC5487
PSK_WITH_AES_256_GCM_SHA384	RFC5487
DHE_PSK_WITH_AES_128_GCM_SHA256	RFC5487
DHE_PSK_WITH_AES_256_GCM_SHA384	RFC5487
RSA_PSK_WITH_AES_128_GCM_SHA256	RFC5487
RSA_PSK_WITH_AES_256_GCM_SHA384	RFC5487
PSK_WITH_AES_128_CBC_SHA256	RFC5487
PSK_WITH_AES_256_CBC_SHA384	RFC5487
DHE_PSK_WITH_AES_128_CBC_SHA256	RFC5487
DHE_PSK_WITH_AES_256_CBC_SHA384	RFC5487
RSA_PSK_WITH_AES_128_CBC_SHA256	RFC5487
RSA_PSK_WITH_AES_256_CBC_SHA384	RFC5487
PSK_WITH_AES_128_CCM	RFC6655
PSK_WITH_AES_256_CCM	RFC6655

Cipher Suite	Reference
DHE_PSK_WITH_AES_128_CCM	RFC6655
DHE_PSK_WITH_AES_256_CCM	RFC6655
PSK_WITH_AES_128_CCM_8	RFC6655
PSK_WITH_AES_256_CCM_8	RFC6655
DHE_PSK_WITH_AES_128_CCM_8	RFC6655
DHE_PSK_WITH_AES_256_CCM_8	RFC6655
ECDHE_PSK_WITH_3DES_EDE_CBC_SHA	RFC5489
ECDHE_PSK_WITH_AES_128_CBC_SHA	RFC5489
ECDHE_PSK_WITH_AES_256_CBC_SHA	RFC5489
ECDHE_PSK_WITH_AES_128_CBC_SHA256	RFC5489
ECDHE_PSK_WITH_AES_256_CBC_SHA384	RFC5489

Table 19 – SSL/TLS Ciphersuites

Appendix B. CAVP certificates

The following tables show all CAVP certificates referenced in this Security Policy for both testing platforms, including the description of their implementation name.

CAVP Cert.	Implementation Name
#A1516	OpenSSL for TLSv1.3 implementation
#A1519	OpenSSL using AVX2 SHA.
#A1520	OpenSSL using AVX SHA.
#A1521	OpenSSL using SSSE3 SHA.
#A1522	OpenSSL using assembler SHA.
#A1523	OpenSSL using AVX2 SHA-3.
#A1524	OpenSSL using Intel AVX-512 SHA-3.
#A1525	OpenSSL using assembler SHA-3.
#A1526	OpenSSL using Generic C non-optimized Triple-DES.
#A1527	OpenSSL using Intel AES-NI AES.
#A1528	OpenSSL using assembler AES.
#A1529	OpenSSL using constant-time bit slice AES.
#A1535	OpenSSL using Intel AES-NI AES using GCM with AVX GHASH.
#A1536	OpenSSL using Intel AES-NI AES using GCM with Intel CLMULNI.
#A1537	OpenSSL using Intel AES-NI AES using assembler block mode.
#A1538	OpenSSL using Assembler AES using GCM with AVX GHASH.
#A1539	OpenSSL using assembler AES using GCM with Intel CLMULNI.
#A1540	OpenSSL using Assembler AES using GCM with assembler GHASH.
#A1541	OpenSSL using Constant-time bit slice AES using GCM with AVX GHASH.
#A1542	OpenSSL using Constant-time bit slice AES using GCM with Intel CLMULNI.
#A1543	OpenSSL using Constant-time bit slice AES using GCM with assembler GHASH.
#A1544	OpenSSL using AVX2 SHA.
#A1545	OpenSSL using AVX SHA.
#A1546	OpenSSL using SSSE3 SHA.
#A1547	OpenSSL using Assembler SHA.

Table 18 – Algorithm implementations in Intel® Xeon® Gold 6226 processor

CAVP Cert.	Implementation Name
#A1516	OpenSSL for TLSv1.3 implementation
#A1522	OpenSSL using assembler SHA.
#A1525	OpenSSL using assembler SHA-3.
#A1526	OpenSSL using Generic C non-optimized Triple-DES.
#A1528	OpenSSL using assembler AES.
#A1530	OpenSSL using CPACF AES.
#A1531	OpenSSL using CPACF AES GCM.
#A1532	OpenSSL using CPACF SHA.
#A1533	OpenSSL using CPACF SHA.
#A1534	OpenSSL using CPACF SHA-3.
#A1547	OpenSSL using Assembler SHA.

Table 20 - Algorithm implementations in IBM z15 processor

Appendix C. Glossary and Abbreviations

AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard New Instructions
API	Application Program Interface
APT	Advanced Package Tool
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CFB	Cipher Feedback
CLMUL	Carry-less Multiplication
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CPACF	CP Assist for Cryptographic Function
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DF	Derivation Function
DSA	Digital Signature Algorithm
DTLS	Datagram Transport Layer Security
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
EMI/EMC	Electromagnetic Interference/Electromagnetic Compatibility
FCC	Federal Communications Commission
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
GPC	General Purpose Computer
HMAC	Hash Message Authentication Code
IG	Implementation Guidance
KAS	Key Agreement Schema
KAT	Known Answer Test
KDF	Key Derivation Function
KW	Key Wrap
KWP	Key Wrap with Padding

MAC	Message Authentication Code
NIST	National Institute of Science and Technology
NRBG	Non-Deterministic Random Bit Generator
OFB	Output Feedback
PAA	Processor Algorithm Acceleration
PAI	Processor Algorithm Implementation
PCT	Pair-wise Consistency Test
PPA	Personal Package Archive
PSS	Probabilistic Signature Scheme
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SSSE3	Supplemental Streaming SIMD Extensions 3
TLS	Transport Layer Security
XTS	XEX-based Tweaked-codebook mode with ciphertext Stealing

Appendix D. References

FIPS140-2	**FIPS PUB 140-2 - Security Requirements For Cryptographic Modules** May 2001 http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
FIPS140-2_IG	**Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program** May 2021 http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf
FIPS180-4	**Secure Hash Standard (SHS)** August 2015 http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
FIPS186-4	**Digital Signature Standard (DSS)** July 2013 http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** November 2001 http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** July 2008 http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
FIPS202	**SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions** August 2015 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
LMAN	**Linux Man Pages** http://man7.org/linux/man-pages/
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1** February 2003 http://www.ietf.org/rfc/rfc3447.txt
RFC2246	**The TLS Protocol Version 1.0** January 1999 https://www.ietf.org/rfc/rfc2246.txt
RFC3268	**Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)** June 2002 https://www.ietf.org/rfc/rfc3268.txt

RFC4279	**Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)** December 2005 https://www.ietf.org/rfc/rfc4279.txt
RFC4346	**The Transport Layer Security (TLS) Protocol Version 1.1** April 2006 https://www.ietf.org/rfc/rfc4346.txt
RFC4492	**Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)** May 2006 https://www.ietf.org/rfc/rfc4492.txt
RFC5116	**An Interface and Algorithms for Authenticated Encryption** January 2008 https://www.ietf.org/rfc/rfc5116.txt
RFC5246	**The Transport Layer Security (TLS) Protocol Version 1.2** August 2008 https://tools.ietf.org/html/rfc5246.txt
RFC5288	**AES Galois Counter Mode (GCM) Cipher Suites for TLS** August 2008 https://tools.ietf.org/html/rfc5288.txt
RFC5487	**Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode** March 2009 https://tools.ietf.org/html/rfc5487.txt
RFC5489	**ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)** March 2009 https://tools.ietf.org/html/rfc5489.txt
RFC6655	**AES-CCM Cipher Suites for Transport Layer Security (TLS)** July 2012 https://tools.ietf.org/html/rfc6655.txt
RFC7251	**AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS** June 2014 https://tools.ietf.org/html/rfc7251.txt
SP800-38A	**NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques** December 2001 http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
SP800-38B	**NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication** May 2005 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf

SP800-38C	**NIST Special Publication 800-38C - Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality** May 2004 http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf
SP800-38D	**NIST Special Publication 800-38D - Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC** November 2007 http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf
SP800-38E	**NIST Special Publication 800-38E - Recommendation for Block Cipher Modes of Operation: The XTS AES Mode for Confidentiality on Storage Devices** January 2010 http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf
SP800-38F	**NIST Special Publication 800-38F - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping** December 2012 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf
SP800-52r2	**NIST Special Publication 800-52 Revision 2 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations** August 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf
SP800-56Ar3	**NIST Special Publication 800-56A Revision 3 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** April 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf
SP800-56Br2	**NIST Special Publication 800-56B Revision 2 - Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography** March 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br2.pdf
SP800-56Cr1	**NIST Special Publication 800-56C Revision 1 – Recommendation for Key-Derivation Methods in Key-Establishment Schemes** August 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr1.pdf
SP800-57	**NIST Special Publication 800-57 Part 1 Revision 5 - Recommendation for Key Management Part 1: General** May 2020 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf
SP800-67	**NIST Special Publication 800-67 Revision 2 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** November 2017 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf

SP800-90A	**NIST Special Publication 800-90A - Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-131A	**NIST Special Publication 800-131A – Revision 2 - Transitioning the Use of Cryptographic Algorithms and Key Lengths** March 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf
SP800-135	**NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions** December 2011 http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf