CANONICAL

ubuntu 16.04 OpenSSL Cryptographic Module

version 2.0

FIPS 140-2 Non-Proprietary Security Policy

Version 2.3

Last update: September 25, 2020

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of Contents

1. Cryptographic Module Specification 6
1.1. Module Overview 6
1.2. Modes of Operation 9
2. Cryptographic Module Ports and Interfaces 11
3. Roles, Services and Authentication 12
3.1. Roles 12
3.2. Services 12
3.3. Algorithms 15
3.3.1. Ubuntu 16.04 LTS 64-bit Running on Intel® Xeon® E5 Processor 15
3.3.2. Allowed Algorithms 20
3.3.3. Non-Approved Algorithms 21
3.4. Operator Authentication 22
4. Physical Security 23
5. Operational Environment 24
5.1. Applicability 24
5.2. Policy 24
6. Cryptographic Key Management 25
6.1. Random Number Generation 27
6.2. Key Generation 27
6.3. Key Agreement / Key Transport / Key Derivation 28
6.4. Key Entry / Output 28
6.5. Key / CSP Storage 29
6.6. Key / CSP Zeroization 29
7. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC) 30
8. Self-Tests 31
8.1. Power-Up Tests 31
8.1.1. Integrity Tests 31
8.1.2. Cryptographic Algorithm Tests 31
8.2. On-Demand Self-Tests 32
8.3. Conditional Tests 32
9. Guidance 34
9.1. Crypto Officer Guidance 34

9.1.1. Operating Environment Configurations 34
9.1.2. Module Installation 35
9.2. User Guidance 35
9.2.1. TLS 35
9.2.2. AES GCM IV 36
9.2.3. AES XTS 36
9.2.4. Triple-DES 36
9.2.5. Random Number Generator 36
9.2.6. API Functions 36
9.2.7. Use of ciphers 37
9.2.8. Environment Variables 37
9.2.9. Handling FIPS Related Errors 37
10. Mitigation of Other Attacks 39
10.1. Blinding Against RSA Timing Attacks 39
10.2. Weak Triple-DES Keys Detection 39

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 2.0 of the Ubuntu 16.04 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 16.04 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 and v1.2 and Datagram Transport Layer Security (DTLS) protocol v1.0 and v1.2, 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.1.0.0	Shared library for TLS/DTLS network protocols.
libcrypto.so.1. 0.0	Shared library for cryptographic implementations.
.libssl.so.1.0.0.hmac	Integrity check signature for libssl shared library.
.libcrypto.so.1.0.0.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.

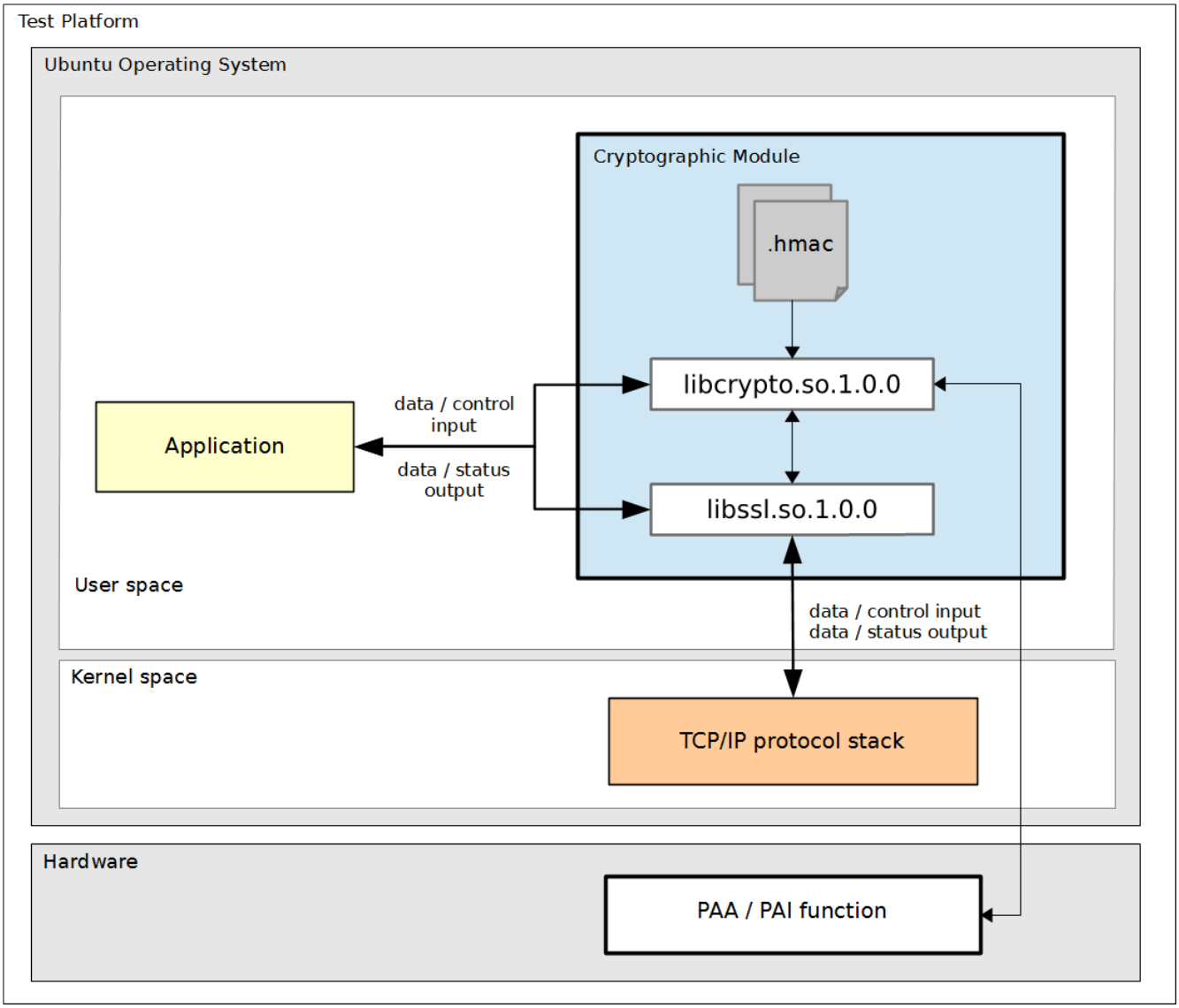

Figure 1 - Software Block Diagram

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.

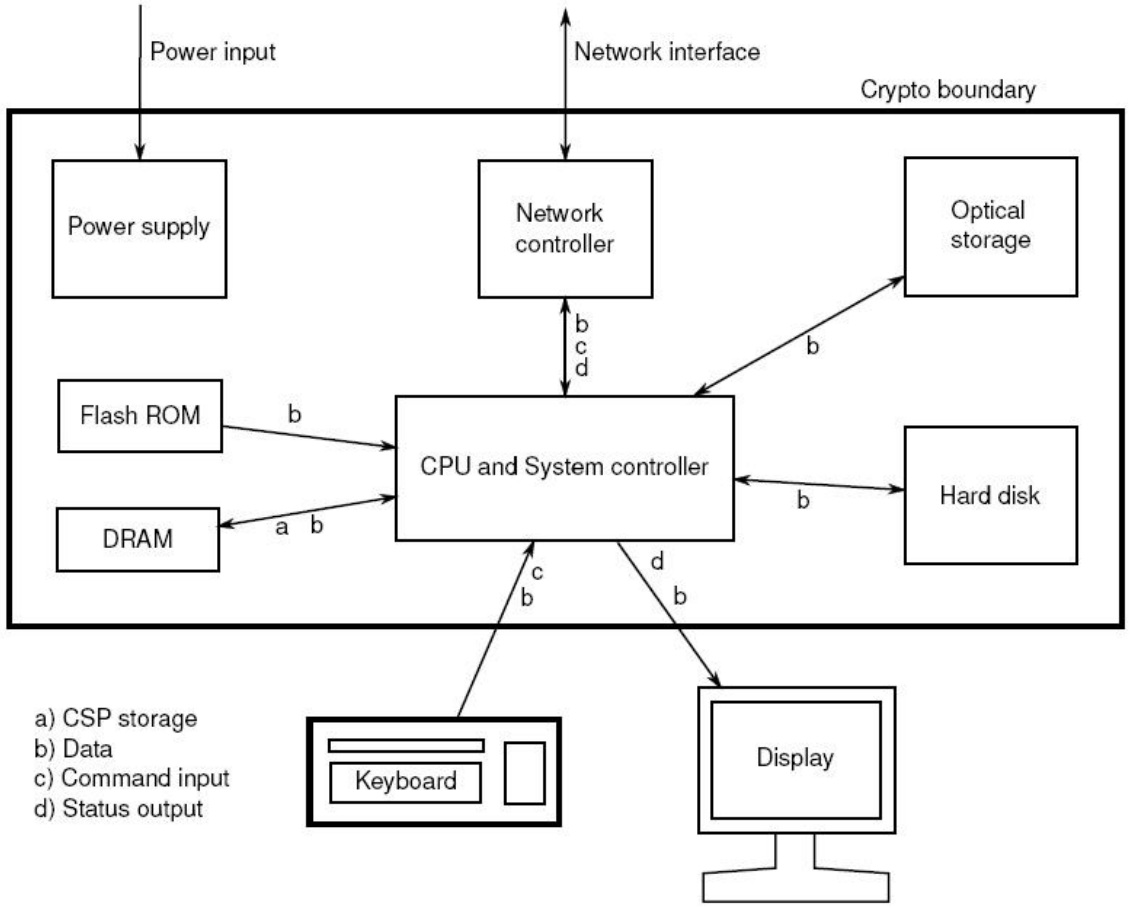

Figure 2 - Cryptographic Module Physical Boundary

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

Test Platform	Processor	Test Configuration
Supermicro SYS-5018R-WR	Intel® Xeon® E5	Ubuntu 16.04 LTS 64-bit with/without AES-NI (PAA)

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.

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 the TLS protocol internal state 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, TLS protocol internal state.
Status Output	Serial port, Ethernet port, Network	API return codes, TLS protocol internal state.
Power Input	PC Power Supply Port	N/A

Table 4 - 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 and initialization, and certificates management.

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 5 and Table 6, 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.

If the services involve the use of the cryptographic algorithms, the corresponding Cryptographic Algorithm Validation System (CAVS) certificate numbers of the cryptographic algorithms can be found in Table 7 and Table 8 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
DSA domain parameter generation	DSA	User	n/a	n/a

Service	Algorithms	Role	Access	Keys/CSP
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, Internal state
Message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 (SHS)	User	n/a	n/a
Message authentication code (MAC)	HMAC	User	Read	HMAC key
	GMAC	User	Read	AES key
	CMAC with AES	User	Read	AES key
	CMAC with Triple-DES	User	Read	Triple-DES key
Key wrapping	AES	User	Read	AES key
Key encapsulation	RSA	User	Read	RSA public and private keys
Diffie-Hellman Key Agreement	KAS FFC	User	Create, Read	Diffie-Hellman domain parameters
EC Diffie-Hellman Key Agreement	KAS ECC, ECC CDH primitive	User	Create, Read	EC Diffie-Hellman public and private keys
Network Protocols Services				
Transport Layer Security (TLS) network protocol v1.0, v1.1 and v1.2	See Appendix A for the complete list of supported cipher suites. TLS KDF	User	Read	AES or Triple-DES key, RSA, DSA or ECDSA public-private key, HMAC Key, Shared Secret, Diffie-Hellman domain parameters 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	Crypto Officer	Read	RSA, DSA or ECDSA public and private keys

Service	Algorithms	Role	Access	Keys/CSP
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	User	n/a	None
Module installation	n/a	Crypto Officer	n/a	None
Module initialization	n/a	Crypto Officer	n/a	None

Table 5 - 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 decryption	2-key Triple-DES listed in Table 9	User	Read	2-key Triple-DES key
Authenticated Encryption cipher for encryption and decryption	AES and SHA from multi-buffer or stitch implementation listed in Table 9	User	Read	AES key, HMAC key
Asymmetric key generation using keys disallowed by [SP800-131A]	RSA, DSA listed in Table 9	User	Create	RSA, DSA or ECDSA public and private keys
Digital signature generation using message digest or keys disallowed by [SP800-131A].	RSA, DSA listed in Table 9	User	Read	RSA or DSA public and private keys
Key establishment using keys disallowed by [SP800-131A].	Diffie-Hellman, RSA listed in Table 9	User	Read	Diffie-Hellman domain parameters or RSA public and private keys
Message digest	MD5	User	n/a	none
Message authentication code (MAC) using keys disallowed by [SP800-131A]	HMAC listed in Table 9, CMAC with 2-key Triple-DES	User	Read	HMAC key, 2-key Triple-DES key
X9.31 RSA Key Generation	ANSI X9.31 RSA Key Generation	User	Create	RSA public and private keys

Table 6 – Services in non-FIPS mode of operation

3.3. Algorithms

The algorithms implemented in the module are tested and validated by CAVP for the following operating environment:

- Ubuntu 16.04 LTS 64-bit running on Intel® Xeon® E5 processor

The Ubuntu 16.04 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) protocol, no parts of this protocol, other than the key derivation function (KDF), have been tested by the CAVP.

3.3.1. Ubuntu 16.04 LTS 64-bit Running on Intel® Xeon® E5 Processor

The module supports the use of AES-NI, SSSE3 and strict assembler for AES implementation, the use of AVX2, SSSE3 and strict assembler for SHA implementation (SSSE3 is only used 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 the CAVS certificates and their associated information of the cryptographic implementation in FIPS mode.

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
AES from AESNI #C1260 AES from strict assembler #C1265 AES using SSSE3 #C1267	AES	[FIPS197], [SP800-38A]	ECB, CBC, OFB, CFB1, CFB8, CFB128, CTR	128, 192, 256	Data Encryption and Decryption
		[SP800-38B]	CMAC	128, 192, 256	MAC Generation and Verification
		[SP800-38C]	CCM	128, 192, 256	Data Encryption and Decryption
		[SP800-38E]	XTS	128, 256	Data Encryption and Decryption for Data Storage
AES from AESNI and PCLMULQDQ for GHASH #C1258 AES from AESNI and assembler for GHASH #C1259 AES from strict	AES	[SP800-38D]	GCM	128, 192, 256	Data Encryption and Decryption
		[SP800-38D]	GMAC	128, 192, 256	Message Authentication Code

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
assembler and PCLMULQDQ for GHASH #C1261 AES and GHASH from strict assembler #C1264 AES using SSSE3 and PCLMULQDQ for GHASH #C1266 AES using SSSE3 and strict assembler for GHASH #C1270					
AES from strict assembler #C1265	AES	[SP800-38F]	KW	128, 192, 256	Key Wrapping and Unwrapping
AVX2 for SHA #C1305 SHA from strict assembler #C1304 SSSE3 for SHA #C1269[1]	DRBG	[SP800-90A]	**Hash_DRBG:** SHA 1, SHA 224, SHA 256, SHA 384, SHA 512 without PR	n/a	Deterministic Random Bit Generation
AVX2 for SHA #C1305 SHA from strict assembler #C1304 SSSE3 for SHA #C1269[1]			**HMAC_DRBG:** SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 without PR		

[1] SSSE3 for SHA implementation only covers SHA-1, SHA-224 and SHA-256 algorithms.

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
AES from strict assembler #C1265			**CTR_DRBG:** AES-128, AES-192, AES-256 with/without DF, without PR		
AVX2 for SHA #C1305 SHA from strict assembler #C1304 SSSE3 for SHA #C1269[1]	DSA	[FIPS186-4]		L=2048, N=224; L=2048, N=256; L=3072, N=256	Key Pair Generation
			SHA-224, SHA-256, SHA-384, SHA-512		Domain Parameter Generation Digital Signature Generation
			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	Domain Parameter Verification Digital Signature Verification
AVX2 for SHA #C1305 SHA from strict assembler #C1304 SSSE3 for SHA #C1269[1]	ECC CDH Primitive (CVL)	[SP800-56A] Section 5.7.1.2	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	EC Diffie-Hellman Key Agreement
	Partial EC Diffie-Hellman (CVL)	[SP800-56A]	ECC Ephemeral Unified scheme	P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement
	Partial Diffie-Hellman (CVL)	[SP800-56A]	FCC dhEphem scheme	p=2048, q=224; p=2048, q=256	Diffie-Hellman Key Agreement
AVX2 for SHA #C1305 SHA from strict assembler #C1304	ECDSA	[FIPS186-4]		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
			SHA-224, SHA-256, SHA-384, SHA-512		Digital Signature Generation

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
SSSE3 for SHA #C1269[1]			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 Digital Signature Verification
AVX2 for SHA #C1305 SHA from strict assembler #C1304 SSSE3 for SHA #C1269[1]	HMAC	[FIPS198-1]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message Authentication Code
AVX2 for SHA #C1305 SHA from strict assembler #C1304 SSSE3 for SHA #C1269[1]	TLS v1.0, v1.1 and v1.2 KDF	[SP800-135]	n/a	n/a	Key Derivation
AVX2 for SHA #C1305 SHA from strict assembler #C1304 SSSE3 for SHA[1] #C1269	RSA	[FIPS186-4]	**X9.31**	2048, 3072	Key Pair Generation
			X9.31 with SHA-256, SHA-384, SHA-512	2048, 3072	Digital Signature Generation
			X9.31 with SHA-1, SHA-256, SHA-384, SHA-512	1024, 2048, 3072	Digital Signature Verification
			PKCS#1v1.5, PSS with SHA-224, SHA-256, SHA-384, SHA-512	2048, 3072	Digital Signature Generation

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
			PKCS#1v1.5, PSS with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024, 2048, 3072	Digital Signature Verification
AVX2 for SHA #C1305 SHA from strict assembler #C1304 SSSE3 for SHA[1] #C1269	RSA	[FIPS186-2]	**PKCS#1v1.5, PSS** with SHA-224, SHA-256, SHA-384, SHA-512	4096	Digital Signature Generation
AVX2 for SHA #C1305 SHA from strict assembler #C1304	SHS	[FIPS180-4]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	n/a	Message Digest
SSSE3 for SHA #C1269[1]	SHS	[FIPS180-4]	SHA-1, SHA-224, SHA-256	n/a	Message Digest
TDES from C #C1257	Two-key Triple-DES	[SP800-67], [SP800-38A]	ECB, CBC, CFB1, CFB8, CFB64, OFB	192	Data Decryption
	Three-key Triple-DES				Data Encryption and Decryption
		[SP800-67], [SP800-38B]	CMAC	192	MAC Generation and Verification

CAVP Cert	Algorithm	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
AES-GCM: #C1258, #C1259, #C1261, #C1264, #C1266, #C1270 (any GCM implementation) AES-CCM: #C1260, #C1265, #C1267 (any CCM implementation) AES-KW: #C1265 (any KW implementation) AES: #C1258, #C1259, #C1260, #C1261, #C1264, #C1265, #C1266, #C1267, #C1270 Triple-DES: #C1257 HMAC: #C1269, #C1304, #C1305	KTS[3] (AES)	[FIPS 198-1] [FIPS180-4] [SP800-67] [SP800-38A] [SP800-38C] [SP800-38D] [SP800-38F]	AES-GCM AES-CCM AES-KW AES- CBC+HMAC-SHA1/224/256/384/512 Triple-DES-CBC+HMAC-SHA1/224/256/384/512	AES keys: 128, 192, 256 bits Triple-DES keys: 192 bits HMAC keys: 112 bits and larger	Key wrapping and unwrapping

Table 7 – Cryptographic Algorithms for Intel® Xeon® E5 Processor

3.3.2. Allowed Algorithms

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

Algorithm	Caveat	Use
RSA Key Encapsulation with key sizes between 2048 and 15360 bits or more	Provides between 112 and 256 bits of encryption strength	Key Establishment; allowed per [FIPS140-2_IG] D.9 and compliant to IG A.14
Diffie-Hellman with key sizes between 2048 and 10000 bits (CVL certs. #C1269, #C1305, #C1304)	Provides between 112 and 219 bits of encryption strength	Key Agreement; allowed per [FIPS140-2_IG] D.8

[3] Approved per IG D.9

Algorithm	Caveat	Use
EC Diffie-Hellman with P-224, P-256, P-384, P-521 curves (CVL certs. #C1269, #C1305, #C1304)	Provides between 112 and 256 bits of encryption strength	Key Agreement; allowed per [FIPS140-2_IG] D.8
RSA Key Generation and Digital Signature Verification with key size greater than 3072 bits	n/a	Digital Signature; allowed per [SP800-131A] and compliant to IG A.14
RSA Digital Signature Generation with key size greater than 4096 bits	n/a	Digital Signature; allowed per [SP800-131A] and compliant to IG A.14
MD5[4]	n/a	Pseudo-random function (PRF) in TLS v1.0 and v1.1; allowed per [SP800-52]
NDRNG	n/a	The module obtains the entropy data from NDRNG to seed the DRBG.

Table 8 – FIPS-Allowed Cryptographic Algorithms

3.3.3. 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	Key Agreement
SHA-1	Message Digest in Digital Signature Generation

[4] According [SP800-52], 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].

MD5	Message Digest
HMAC with key size less than 112 bits	Message Authentication Code
AES in XTS mode with 192-bit key	Data Encryption and Decryption
2-key Triple-DES	Data Encryption
Multiblock 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
SSLeay Deterministic Random Number Generator (PRNG)	Random Number Generation

Table 9 - 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	CSP size / mode	Generation	Entry and Output	Zeroization
AES keys	128, 192, 256 bits (ECB, CBC, OFB, CFB1, CFB8, CFB128, CTR, CCM, GCM, GMAC, KW) 128, 256 bits (XTS)	Not Applicable. The key material is entered via API parameter or generated during the Diffie-Hellman or EC Diffie-Hellman key agreement.	The key is passed into the module via API input parameters in plaintext.	EVP_CIPHER_CTX_cleanup()
Triple-DES keys	192 bits (ECB, CBC, CFB1, CFB8, CFB64, OFB, CMAC)			EVP_CIPHER_CTX_cleanup()
HMAC keys	112 bits or greater (SHA-1, SHA2-224/256/384/512, SHA3-224/256/384/512)			HMAC_CTX_cleanup()
RSA private keys (key generation)	2048 bits or greater	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 out of the module via API output parameters in plaintext.	RSA_free()
RSA private keys (signature verification)	1024 bits of greater		The key is passed into the module via API input parameters in plaintext.	
RSA private keys (signature generation)	2048 bits or greater		The key is passed into the module via API input parameters in plaintext.	
RSA private keys	2048 bits or		The key is passed	

(key encapsulation)	greater		into the module via API input parameters in plaintext.	
DSA private keys (key generation)	2048 or 3072 bits		The key is passed out of the module via API output parameters in plaintext.	DSA_free()
DSA private keys (signature generation/verification)			The key is passed into the module via API input parameters in plaintext.	
ECDSA private keys (key generation)	P-224, P-256, P-384, P-521		The key is passed out of the module via API output parameters in plaintext.	EC_KEY_free()
ECDSA private keys (signature generation/verification)			The key is passed into the module via API input parameters in plaintext.	
Diffie-Hellman domain parameters	2048 up to 10000 bits	The domain parameters used in Diffie-Hellman and the components to generate the public and private keys used in EC Diffie-Hellman 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 private keys	P-224, P-256, P-384, P-521			EC_KEY_free()
Diffie-Hellman Shared secret	2048 bits or greater	Generated during the Diffie-Hellman key agreement.	None	SSL_free(), SSL_clear()
EC Diffie-Hellman Shared secret	P-256, P-384, P-521	Generated during the EC Diffie-Hellman key agreement.	None	SSL_free(), SSL_clear()
Entropy input string	According to SP800-90A	Obtained from the NDRNG.	None	FIPS_drbg_free()
DRBG internal state (V, C)	According to SP800-90A (Hash)	During DRBG initialization.	None	FIPS_drbg_free()

DRBG internal state (V, C, Key)	According to SP800-90A (HMAC, CTR)			

Table 10 - 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 HMAC keys, key components of asymmetric keys, symmetric keys, server and client random numbers for the TLS protocol, and internal CSPs. In addition, the module provides a Random Number Generation service to calling applications.

The module supports all three DRBG mechanisms: Hash, HMAC and CTR. The DRBG is initialized during module initialization; the module loads by default the DRBG using the CTR_DRBG mechanism 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 Number Generator (NDRNG), getrandom() system call, as the entropy source for seeding the DRBG. The NDRNG is provided by the operational environment (i.e., Linux RNG), which is within the module's physical boundary but outside of the module's logical boundary. The NDRNG provides at least 128 bits of entropy to the DRBG during initialization (seed) and reseeding (reseed).

The Ubuntu Linux kernel performs conditional self-tests on the output of NDRNG to ensure that consecutive random numbers do not repeat. The module performs the 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 NDRNG 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.

CAVEAT: The module generates cryptographic keys whose strengths are modified by available entropy.

6.2. Key Generation

The Module provides an SP800-90A-compliant Deterministic Random Bit Generator (DRBG) for creation of key components of asymmetric keys, and random number generation.

The getrandom() system call from the Operational Environment is used as a source of random numbers for DRBG seeds and entropy input string.

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

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 key pairs used in the Diffie-Hellman and EC Diffie-Hellman KAS are generated internally by the module using the same DSA and ECDSA key generation compliant with [FIPS186-4] which is compliant with [SP800-56A].

6.3. Key Agreement / Key Transport / Key Derivation

The module provides Diffie-Hellman and EC Diffie-Hellman key agreement schemes. These key agreement schemes are also used as part of the TLS protocol key exchange.

The module provides key wrapping using the AES with KW mode, and AES in CCM or GCM modes.

The module provides RSA key encapsulation using private key encryption and public key decryption primitives. RSA key encapsulation is also used as part of the TLS protocol key exchange.

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), 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.

According to Table 2: Comparable strengths in [SP 800-57], the key sizes of AES, Triple-DES, RSA, Diffie-Hellman and EC Diffie-Hellman provides the following security strength in FIPS mode of operation:

- RSA key wrapping[5] provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman key agreement provides between 112 and 219 bits of encryption strength.
- EC Diffie-Hellman key agreement provides between 112 and 256 bits of encryption strength.
- AES KW, GCM and CCM key wrapping provides between 128 and 256 bits of encryption strength.
- Combination of AES encryption in CBC mode and HMAC (as part of the TLS protocol) provides 128 or 256 bits of encryption strength).
- Combination of Triple-DES encryption in CBC mode and HMAC (as part of the TLS protocol) provides 112 bits of encryption strength).

The module supports key derivation for the TLS protocol. The module implements the pseudo-random functions (PRF) for TLSv1.0/1.1 and TLSv1.2.

Note: As the module supports RSA key sizes of 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.

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

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 10. Calling the SSL_free() and SSL_clear() will zeroize the keys and CSPs stored in the TLS protocol internal state and also invoke the module's API listed in Table 10 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 14 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) and Pair-wise Consistency Tests (PCT) 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 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#1 v1.5 scheme and SHA-256, signature generation • KAT RSA with 2048-bit key, PKCS#1 v1.5 scheme and SHA-256, signature verification
DRBG	• KAT CTR_DRBG with AES with 128, 192 and 256 bit keys, without PR, with DF • KAT CTR_DRBG with AES with 128, 192 and 256 bit keys, without PR, without DF • KAT Hash_DRBG without PR • KAT HMAC_DRBG without PR
EC Diffie-Hellman	• Primitive “Z” Computation KAT with P-256 curve
Diffie-Hellman	• Primitive “Z” Computation KAT with 2048-bit key
TLS KDF	• KAT KDF for TLSv1.0 and v1.1 • KAT KDF for TLSv1.2

Table 11- 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.

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) and Continuous Random Number Generator Test (CRNGT), 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 for encryption and decryption.
DRBG	• CRNGT is not required per IG 9.8

Table 12 - 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	libssl1.0.0_1.0.2g-1ubuntu4.fips.4.15.2_amd64.deb libssl1.0.0-hmac_1.0.2g-1ubuntu4.fips.4.15.2_amd64.deb

Table 13 – Ubuntu packages

The libssl-doc_1.0.2g-1ubuntu4.fips.4.15.2_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:

(1) Install the linux-fips Ubuntu package.

(2) Install the fips-initramfs Ubuntu package.

(3) Create the file /etc/default/grub.d/99-fips.cfg with the content: GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT fips=1".

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

(5) Execute update-grub to update the boot loader.

(6) Execute 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 libssl1.0-dev Ubuntu package provides include files that are necessary to build applications using the module.

9.1.2. Module Installation

Once the operating environment configuration is finished, the Crypto Officer can install the Ubuntu packages containing the module listed in Table 13 using normal packaging tool such as Advanced Package Tool (APT). For example:

```
$ sudo apt-get install libssl1.0 libssl1.0-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.

To download the FIPS validated version of the module, please email "sales@canonical.com" or contact a Canonical representative, https://www.ubuntu.com/contact-us.

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 and 1.2. 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]. In addition, as required also by [SP800-131A], Diffie-Hellman with keys smaller than 2048 bits must not be used.

The TLS protocol lacks the support to negotiate the used Diffie-Hellman key sizes. To ensure full support for all TLS protocol versions, the TLS client implementation of the module accepts Diffie-Hellman key sizes smaller than 2048 bits offered by the TLS server.

The TLS server implementation allows the application to set the Diffie-Hellman key size. The server side must always set the DH parameters with the API call of SSL_CTX_set_tmp_dh(ctx, dh).

For complying with the requirement to not allow Diffie-Hellman key sizes smaller than 2048 bits, the Crypto Officer must ensure that:

- in case the module is used as a TLS server, the Diffie-Hellman parameters of the aforementioned API call must be 2048 bits or larger;

- in case the module is used as a TLS client, the TLS server must be configured to only offer Diffie-Hellman keys of 2048 bits or larger.

9.2.2. AES GCM 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 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 SP800-52 Rev2.

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. Random Number Generator

The RAND_cleanup() API function must not be used. This call will clean up the internal DRBG state. This call also replaces the DRBG instance with the non-FIPS Approved SSLeay Deterministic Random Number Generator when using the RAND_* API calls.

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.

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 CAVS; 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 14 - 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 [SP800-52], 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, TDES, 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 CRNGT fails the output of the NDRNG	FIPS_R_ENTROPY_SOURCE_STUCK (142) "entropy source stuck"
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	FIPS_R_AES_XTS_WEAK_KEY (201) "identical keys are weak"

Table 15 – 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
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
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
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

Appendix B. 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
CAVS	Cryptographic Algorithm Validation System
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
CRNGT	Continuous Random Number Generator Test
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
LPAR	Logical Partitions
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
NDRNG	Non-Deterministic Random Number Generator
OFB	Output Feedback
PAA	Processor Algorithm Acceleration
PAI	Processor Algorithm Implementation
PCT	Pair-wise Consistency Test
PR	Prediction Resistance
PRNG	Pseudo-Random Number Generator
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 C. 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**
October 23, 2019
http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf

FIPS180-4 **Secure Hash Standard (SHS)**
March 2012
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

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
http://csrc.nist.gov/publications/nistpubs/800-38B/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

SP800-52 **NIST Special Publication 800-52 Revision 1 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations**
April 2014
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf

SP800-56A **NIST Special Publication 800-56A Revision 2 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography**
May 2013
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800 56Ar2.pdf

SP800-56B **NIST Special Publication 800-56B Revision 1 - Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography**
September 2014
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br1.pdf

SP800-57 **NIST Special Publication 800-57 Part 1 Revision 4 - Recommendation for Key Management Part 1: General**
January 2016
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

SP800-67 **NIST Special Publication 800-67 Revision 1 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher**
January 2012
http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.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