SUSE Linux Enterprise Server OpenSSL Cryptographic Module

version 4.0

FIPS 140-2 Non-Proprietary Security Policy

Doc version 4.0.4

Last update: 2021-11-23

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of contents

1 Cryptographic Module Specification

This document is the non-proprietary security policy for the SUSE Linux Enterprise Server OpenSSL Cryptographic Module version 4.0. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS 140-2 (Federal Information Processing Standards Publication 140-2) for a security level 1 module.

This document was prepared in partial fulfillment of the FIPS 140-2 requirements for cryptographic modules and is intended for security officers, developers, system administrators and end-users.

FIPS 140-2 details the requirements of the Governments of the U.S. and Canada for cryptographic modules, aimed at the objective of protecting sensitive but unclassified information. For more information on the FIPS 140-2 standard and validation program please refer to the NIST website at http://csrc.nist.gov/.

Throughout the document, “the OpenSSL module” and “the module” are also used to refer to the SUSE Linux Enterprise Server OpenSSL Cryptographic Module version 4.0.

1.1 Module Overview

The SUSE Linux Enterprise Server OpenSSL Cryptographic Module is a software cryptographic module that implements the Transport Layer Security (TLS) protocol versions 1.0, 1.1 and 1.2, the Datagram Transport Layer Security (DTLS) protocol versions 1.0 and 1.2, and general-purpose cryptographic services.

This Module provides cryptographic services to applications running in the user space of the underlying operating system through a C language application program interface (API). The Module may utilize processor instructions to optimize and increase performance. The Module can act as a TLS server or TLS 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. Table 1 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

Table 1: Security Levels

Table 2 lists the software components of the cryptographic module, which defines its logical boundary. The module is provided for 32-bit and 64-bit Intel architectures.

Processor Architecture	Component	Description
Intel 64-bit	/usr/lib64/libcrypto.so.1.1	Shared library for cryptographic algorithms.
	/usr/lib64/libssl.so.1.1	Shared library for TLS/DTLS network protocols.
	/usr/lib64/.libcrypto.so.1.1.hmac	Integrity check HMAC value for the libcrypto shared library.
	/usr/lib64/.libssl.so.1.1.hmac	Integrity check HMAC value for the libssl shared library.
Intel 32-bit	/usr/lib/libcrypto.so.1.1	Shared library for cryptographic algorithms.
	/usr/lib/libssl.so.1.1	Shared library for TLS/DTLS network protocols.
	/usr/lib/.libcrypto.so.1.1.hmac	Integrity check HMAC value for the libcrypto shared library.
	/usr/lib/.libssl.so.1.1.hmac	Integrity check HMAC value for the libssl shared library.

Table 2: Cryptographic Module Components

The software block diagram below shows the logical boundary of the module, and its interfaces with the operational environment.

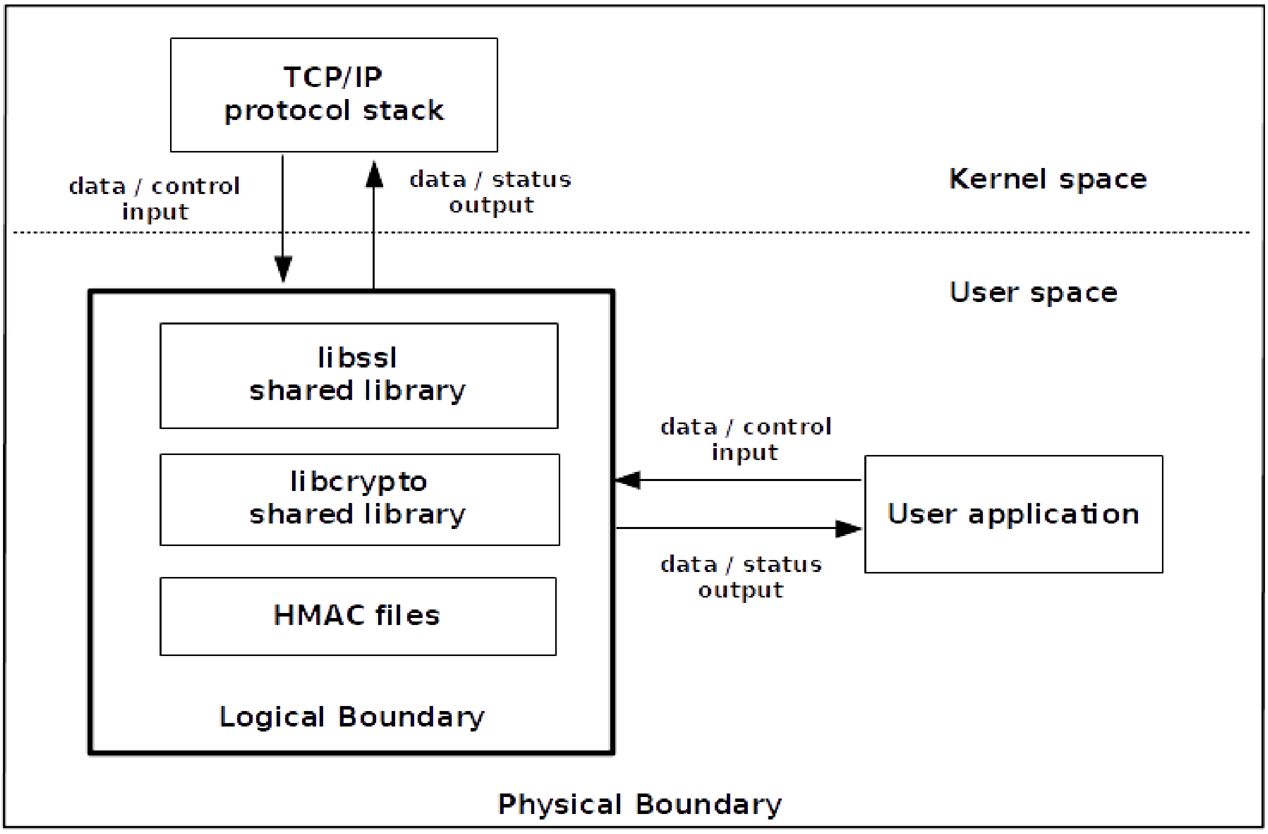

Figure 1: Software Block Diagram

The module is aimed to run on a general purpose computer (GPC). Table 3 shows the platform on which the module has been tested:

Platform	Processor	Test Configuration
Dell EMC PowerEdge 640	Intel® Cascade Lake Xeon® Gold 6234	SUSE Linux Enterprise Server 15 SP0 with and 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.

The physical boundary of the module is the surface of the case of the tested platform. Figure 2 shows the hardware block diagram including major hardware components of a GPC.

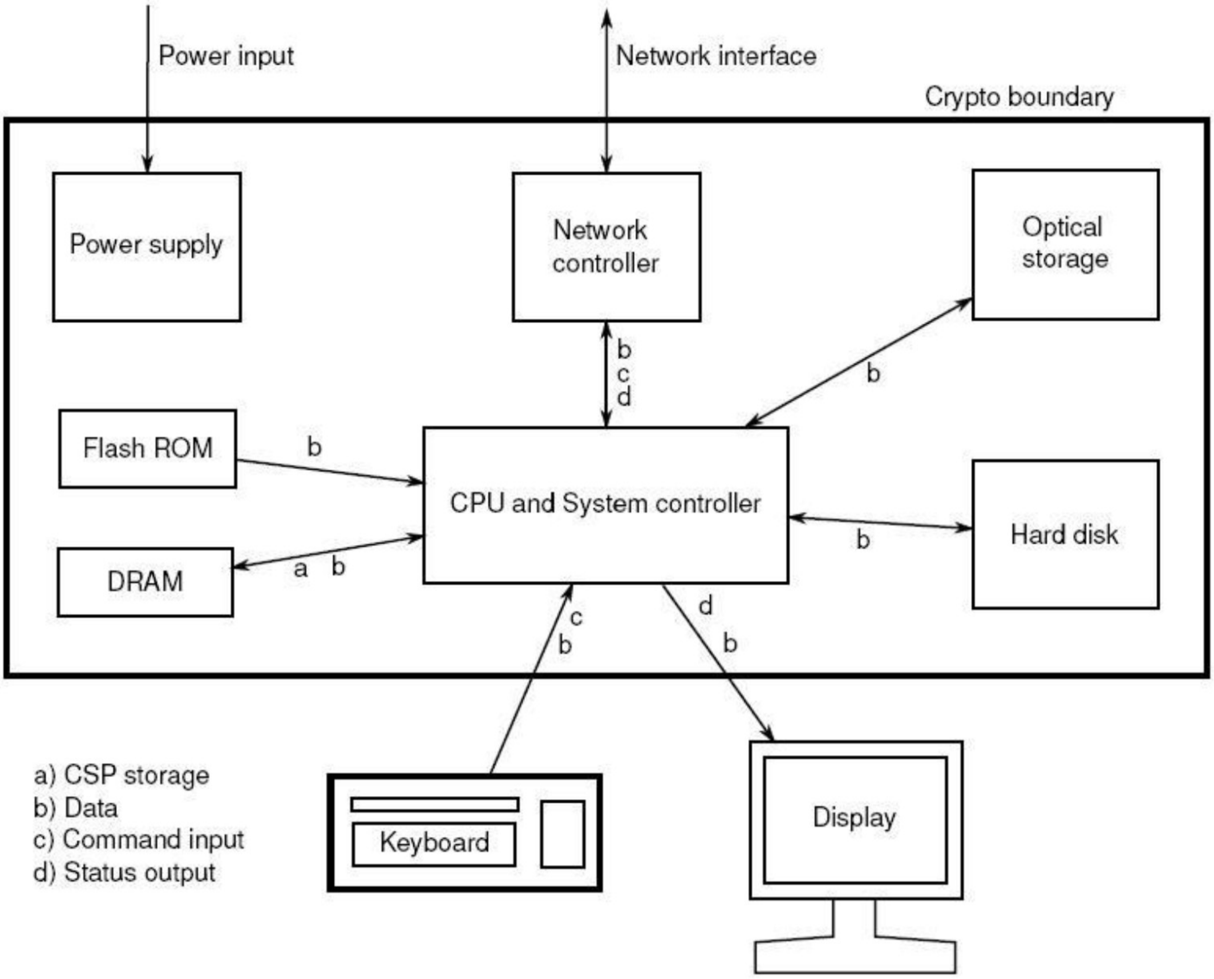

Figure 2: Hardware Block Diagram

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 (CSPs) 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 ports and interfaces are shown in the following table.

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

Table 4: Ports and Interfaces

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 configuration.

3.2 Services

The module provides services to the users that assume one of the available roles. All services are shown in Table 5 and Table 6.

Table 5 lists the services available in FIPS mode. For each service, the table lists the associated cryptographic algorithm(s), the role to perform the service, the cryptographic keys or CSPs involved, and their access type(s). 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 details of the approved cryptographic algorithms including the CAVP certificate numbers can be found in Table 7.

Service	Algorithm	Role	Keys/CSPs	Access
Cryptographic Services				
Symmetric encryption and decryption	AES	User	AES key	Read
	Three-key Triple-DES	User	Three-key Triple-DES key	Read
Symmetric decryption	Two-key Triple-DES	User	Two-key Triple-DES key	Read
RSA key generation	RSA, DRBG	User	RSA public and private keys	Create
RSA digital signature generation and verification	RSA, SHS	User	RSA public and private keys	Read
DSA key generation	DSA, DRBG	User	DSA public and private keys	Create
DSA domain parameter generation	DSA	User	None	n/a

Service	Algorithm	Role	Keys/CSPs	Access
DSA digital signature generation and verification	DSA, SHS	User	DSA public and private keys	Read
ECDSA key generation	ECDSA, DRBG	User	ECDSA public and private keys	Create
ECDSA public key validation	ECDSA	User	ECDSA public key	Read
ECDSA signature generation and verification	ECDSA, DRBG, SHS	User	ECDSA public and private keys	Read
Random number generation	DRBG	User	Entropy input string, seed material	Read
			Internal state	Update
Message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	User	None	N/A
Message authentication code (MAC)	HMAC	User	HMAC key	Read
	CMAC with AES	User	AES key	Read
	CMAC with Triple-DES	User	Triple-DES key	Read
Key encapsulation	RSA	User	RSA public and private keys	Read
Key wrapping	AES-KW, AES-KWP	User	AES key	Read
Diffie-Hellman Shared Secret Computation	KAS-FFC-SSC	User	Diffie-Hellman public and private keys	Create, Read
			Shared secret	
Diffie-Hellman key generation and verification using safe primes	Safe Primes Key Generation and Verification	User	Diffie-Hellman public and private keys	Create, Read
EC Diffie-Hellman Shared Secret Computation	KAS-ECC-SSC	User	EC Diffie-Hellman public and private keys	Create , Read
			Shared secret	
Key derivation	TLS KDF	User	Shared secret	Read
			Derived key	Create
	SSH KDF	User	Shared secret	Read
			Derived key	Create
	PBKDF KDF	User	Password/passphrase	Read
			Derived key	Create
Network Protocol Services				
Transport Layer Security (TLS)	Supported cipher suites in FIPS mode (see	User	RSA, DSA or ECDSA public and private keys	Read

Service	Algorithm	Role	Keys/CSPs	Access
network protocol v1.0, v1.1 and v1.2	Appendix A for the complete list of valid cipher suites)		TLS pre_master_secret, TLS master_secret, Diffie Hellman or EC Diffie Hellman public and private keys, AES or Triple-DES key, HMAC key	Create
TLS extensions	n/a	User	RSA, DSA or ECDSA public and private keys	Read
Certificate management	n/a	Crypto Officer	RSA, DSA or ECDSA public and private keys	Read
Other FIPS-related Services				
Show status	N/A	User	None	N/A
Zeroization	N/A	User	All CSPs	Zeroize
Self-tests	AES, Diffie-Hellman, DSA, EC Diffie-Hellman, ECDSA, DRBG, HMAC, RSA, SHS, Triple-DES	User	None	N/A
Module installation and configuration	N/A	Crypto Officer	None	N/A
Module initialization	N/A	Crypto Officer	None	N/A

Table 5: Services in FIPS mode of operation

Table 6 lists the services only available in non-FIPS mode of operation. The details of the non-approved cryptographic algorithms available in non-FIPS mode can be found in Table 9.

Service	Algorithm / Modes	Role	Keys	Access
Cryptographic Services				
Symmetric encryption and decryption	ARIA, Blowfish, Camellia, CAST, CAST5, ChaCha20, DES, RC2, RC4, SEED, and Poly1305	User	Symmetric key	Read
Symmetric encryption	Two-key Triple-DES	User	Two-key Triple-DES key	Read
Authenticated encryption cipher for encryption and decryption	AES and SHA from multi-buffer or stitch implementations listed in Table 9	User	AES key, HMAC key	Read
Asymmetric key generation	RSA, DSA and ECDSA restrictions listed in Table 9	User	RSA, DSA or ECDSA public and private keys	Create

Service	Algorithm / Modes	Role	Keys	Access
Digital signature generation and verification	RSA, DSA and ECDSA and message digest restrictions listed in Table 9	User	RSA, DSA or ECDSA public and private keys	Read
Message digest	Blake2, Gost, MD4, MD5, MDC2, RMD160	User	None	N/A
Message authentication code (MAC)	HMAC and CMAC restrictions listed in Table 9 GMAC	User	HMAC key, two-key Triple-DES key	Read
RSA key encapsulation	RSA keys smaller than 2048 bits.	User	RSA key pair	Read
Diffie-Hellman shared secret computation	Diffie-Hellman restrictions listed in Table 9	User	Diffie-Hellman public and private keys	Read
EC Diffie-Hellman shared secret computation	Restrictions listed in Table 9	User	EC Diffie-Hellman public and private keys	Read
Key derivation	KDF PBKDF using non-approved message digest.	User	Password/passphrase	Read
			Derived key	Create
Network Protocol Services				
Transport Layer Security (TLS) network protocol v1.0, v1.1 and v1.2	Non-supported cipher suites (see Appendix A for the complete list of valid cipher suites)	User	RSA, DSA or ECDSA public and private keys	Read
			TLS pre_master_secret, TLS master_secret, Diffie Hellman or EC Diffie Hellman public and private keys, AES or Triple-DES key, HMAC key	Create

Table 6: Services in non-FIPS mode of operation

3.3 Operator Authentication

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

3.4 Algorithms

The module provides multiple implementations of algorithms. 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, and the use of the CLMUL instruction set and strict assembler for GHASH that is used in 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.

Notice that for the Transport Layer Security (TLS) and Secure Shell (SSH) protocols, no parts of these protocols, other than the key derivation functions (SP800-135 TLS and SSH KDFs), have been tested by the CAVP.

Table 7 lists the approved algorithms, the CAVP certificates, and other associated information of the cryptographic implementations in FIPS mode. Please refer to Appendix B for more detailed information about the algorithm implementations tested for each CAVP certificate.

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	#A77 #A78 #A79 #A83 #A86 #A89 #A90 #A91 #A92 #A93 #A94 #A95 #A99 #A100 #A101 #A187 #A188 #A189 #A192 #A194 #A196 #A200 #A201 #A206 #A207 #A208 #A209 #A210 #A211 #A212
	CBC, CFB1, CFB8, CFB128, OFB, CTR	128, 192, 256	Data Encryption and Decryption	FIPS197, SP800-38A	#A86 #A90 #A99 #A196 #A201 #A212
	CMAC	128, 192, 256	MAC Generation and Verification	SP800-38B	
	CCM	128, 192, 256	Data Encryption and Decryption	SP800-38C	
	XTS	128, 256	Data Encryption and Decryption for Data Storage	SP800-38E	
	KW, KWP	128, 192, 256	Key Wrapping and Unwrapping	SP800-38F	

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
	GCM	128, 192, 256	Data Encryption and Decryption	SP800-38D	#A77 #A78 #A89 #A91 #A92 #A94 #A95 #A100 #A101 #A187 #A189 #A192 #A206 #A207 #A208 #A209 #A210 #A211
DRBG	CTR_DRBG: AES-128, AES-192, AES-256 with/without DF, with/without PR	N/A	Deterministic Random Bit Generation	SP800-90A	#A77 #A78 #A79 #A83 #A89 #A91 #A92 #A93 #A94 #A95 #A100 #A101 #A187 #A188 #A189 #A192 #A194 #A200 #A206 #A207 #A208 #A209 #A210 #A211
	Hash_DRBG: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 with/without PR	N/A	Deterministic Random Bit Generation	SP800-90A	#A76 #A84 #A87 #A88 #A191 #A193 #A198 #A199

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
	HMAC_DRBG: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 with/without PR	N/A	Deterministic Random Bit Generation	SP800-90A	#A76 #A84 #A87 #A88 #A191 #A193 #A198 #A199
DSA		L=2048, N=224 L=2048, N=256 L=3072, N=256	Key Pair Generation	FIPS186-4	#A80 #A81 #A82 #A85 #A186 #A195 #A202 #A203
	SHA-224	L=2048, N=224	Domain Parameter Generation		
	SHA-256	L=2048, N=256 L=3072, N=256			
	SHA-224, SHA-256, SHA-384, SHA-512	L=2048, N=224	Digital Signature Generation		
	SHA-256, SHA-384, SHA-512	L=2048, N=256 L=3072, N=256			
	SHA-1	L=1024, N=160	Domain Parameter Verification		
	SHA-224	L=2048, N=224			
	SHA-256	L=2048, N=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		
ECDSA		P-256, P-384, P-521	Key Pair Generation Public Key Verification	FIPS186-4	#A80 #A81 #A82 #A85 #A186 #A195 #A202 #A203
	SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Digital Signature Generation		
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Digital Signature Verification		

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
HMAC	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code	FIPS198-1	#A76 #A80 #A81 #A82 #A84 #A85 #A87 #A88 #A186 #A191 #A193 #A195 #A198 #A199 #A202 #A203
KAS-ECC-SSC	ECC Ephemeral Unified Scheme	P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement	SP800-56ARev3	#A676 #A677
KAS-FCC-SSC	dhEphem Scheme with safe prime groups.	2048, 3072, 4096, 6144, 8192	Diffie-Hellman Key Agreement	SP800-56ARev3	#A676 #A677
Safe Primes Key Generation and Verification	Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048[1], MODP-3072, MODP-4096, MODP-6144, MODP-8192	2048, 3072, 4096, 6144, 8192	Diffie-Hellman Key Agreement	SP800-56ARev3	#A676 #A677
KDF PBKDF	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512		Key Derivation	SP800-132	#A80 #A81 #A82 #A85 #A186 #A195 #A202 #A203
KDF SSH	AES with SHA-1, SHA-256, SHA-384, SHA-512	128, 192, 256	Key Derivation	SP800-135	CVLs. #A96 #A98

1 Note that the module only implements key generation and verification, and shared secret computation of the MODP safe prime groups defined in RFC3526 for the Internet Key Exchange (IKE) protocol. The module does not implement any other part of the IKE protocol.

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
	Triple-DES with SHA-1, SHA-256, SHA-384, SHA-512	192			#A102 #A103 #A190 #A204 #A205 #A213
KDF TLS	TLS v1.0, v1.1, v1.2		Key Derivation	SP800-135r1	CVLs. #A80 #A81 #A82 #A85 #A186 #A195 #A202 #A203
RSA	B.3.3	2048, 3072, 4096	Key Pair Generation	FIPS186-4	#A80 #A81 #A82 #A85 #A186 #A195 #A202 #A203
	PKCS#1v1.5: SHA-224, SHA-256, SHA-384, SHA-512	2048, 3072, 4096	Digital Signature Generation		
	PSS: SHA-224, SHA-256, SHA-384, SHA-512	2048, 3072, 4096			
	X9.31: SHA-256, SHA-384, SHA-512	2048, 3072, 4096			
	PKCS#1v1.5: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024, 2048, 3072, 4096	Digital Signature Verification		
	PSS: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1024, 2048, 3072, 4096			
	X9.31: SHA-1, SHA-256, SHA-384, SHA-512	1024, 2048, 3072, 4096			

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	#A76 #A80 #A81 #A82 #A84 #A85 #A87 #A88 #A186 #A191 #A193 #A195 #A198 #A199 #A202 #A203
Triple-DES	ECB, CBC, CFB1, CFB8, CFB64, OFB	192 (two-key Triple-DES)	Data Decryption	SP800-67 SP800-38A	#A97 #A197
		192 (three-key Triple-DES)	Data Encryption and Decryption		
	CMAC	192	MAC Generation and Verification	SP800-67 SP800-38B	#A97 #A197
KTS	AES KW, KWP	128, 192, 256	Key Wrapping and unwrapping	SP800-38F	#A86 #A90 #A99 #A196 #A201 #A212
	AES CCM	128, 256			#A86 #A90 #A99 #A196 #A201 #A212
	AES GCM	128, 256			#A77 #A78 #A89 #A91 #A92 #A94 #A95 #A100 #A101 #A187 #A189 #A192 #A206 #A207 #A208 #A209 #A210 #A211

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
	AES CBC and HMAC	128, 256			#A86 #A90 #A99 #A196 #A201 #A212 #A76 #A80 #A81 #A82 #A84 #A85 #A87 #A88 #A186 #A191 #A193 #A195 #A198 #A199 #A202 #A203
	Triple-DES CBC and HMAC	192[2]			#A97 #A197 #A76 #A80 #A81 #A82 #A84 #A85 #A87 #A88 #A186 #A191 #A193 #A195 #A198 #A199 #A202 #A203

Table 7: Approved Cryptographic Algorithms for Intel Xeon Processor

3.5 Allowed Algorithms

Table 8 describes the non-approved but allowed algorithms in FIPS mode:

Algorithm	Use
RSA Key Encapsulation with Encryption and Decryption Primitives with keys equal or larger than 2048 bits up to 15360 or more.	Key Establishment; allowed per [FIPS140-2_IG] D.9

2 The algorithm provides 112 bits of security strength.

Algorithm	Use
MD5	Pseudo-random function (PRF) in TLS v1.0 and v1.1; allowed per [SP800-52] and [SP800-135] section 4.2.1.
NDRNG	The module obtains the entropy data from a NDRNG to seed the DRBG.

Table 8: Non-Approved but Allowed Algorithms

3.5.1 Non-Approved Algorithms

Table 9 shows the non-Approved cryptographic algorithms implemented in the module that are only available in non-FIPS mode.

Algorithm	Use
ARIA, Blowfish, Camellia, CAST, CAST5, ChaCha20, DES, RC2, RC4, SEED, Camellia	Data Encryption and Decryption.
2-key Triple-DES	Data Encryption.
Chacha20 and Poly1305	Authenticated Data Encryption and Decryption.
Blake2, MD4, MD5, MDC2, RMD160, GHASH	Message Digest.
HMAC with less than 112-bit keys	Message Authentication Code.
CMAC with 2-key Triple-DES	Message Authentication Code.
GMAC	Message Authentication Code.
SHA-1	Digital Signature Generation, DSA Domain Parameter Generation.
DSA with keys smaller than 2048 bits or greater than 3072 bits.	Key Pair Generation, Domain Parameter Generation.
DSA with keys smaller than 2048 bits or greater than 3072 bits. DSA with L=2048, N=256 or L=3072, N=256 and using SHA-1 or SHA-224.	Digital Signature Generation.
DSA with keys smaller than 1024 bits or greater than 3072 bits.	Domain Parameter Verification, Digital Signature Verification.
RSA with keys smaller than 2048 bits or greater than 4096 bits.	Key Pair Generation, Digital Signature Generation.
RSA with keys smaller than 1024 bits or greater than 4096 bits.	Digital Signature Verification.
RSA with keys smaller than 2048 bits	Key Encapsulation.
ECDSA with P-192 and P-224 curves, K curves, B curves and non-NIST curves.	Key Pair Generation, Public Key Validation , Digital Signature Generation and Verification.
Diffie-Hellman with keys generated with domain parameters other than safe primes.	Key Agreement, Shared Secret computation.
EC Diffie-Hellman with P-192 curve, K curves, B curves and non-NIST curves.	Key Agreement, Shared Secret computation.

Algorithm	Use
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.
AES and SHA from multi-buffer or stitch implementations	Data Encryption and Decryption, Message Digest.
PBKDF with non-approved message digest algorithms.	Key Derivation.

Table 9: Non-Approved Cryptographic Algorithms

4 Physical Security

The module is comprised of software only and thus does not claim any physical security.

5 Operational Environment

This module operates in a modifiable operational environment per the 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.

The SUSE Linux Enterprise Server operating system is used as the basis of other products which include but are not limited to:

- SLES
- SLES for SAP
- SLED
- SLE Micro

Compliance is maintained for these products whenever the binary is found unchanged.

Note: The CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when so ported if the specific operational environment is not listed on the validation certificate.

5.1 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.

The ptrace system call, the debugger gdb and strace shall not be used. In addition, other tracing mechanisms offered by the Linux environment, such as ftrace or systemtap shall not be used.

6 Cryptographic Key Management

Table 10 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	Key material is entered via API parameters or derived during Diffie-Hellman or EC Diffie-Hellman key agreement.	Keys are 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	Public and private keys are generated using the FIPS 186-4 key generation method; random values are obtained from the SP800-90A DRBG.	Keys are passed into the module via API input parameters in plaintext. Keys are 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	Public and private keys are generating using the SP800-56ARev3 Safe Primes key generation method, random values are obtained from the SP800-90A DRBG.	The key is passed into the module via API input parameters in plaintext. Keys are passed out of the module via API output parameters in plaintext.	DH_free()
EC Diffie-Hellman public and private keys	Public and private keys are generated using the FIPS 186-4 key generation method, and the random values are obtained from the SP800 90A DRBG.	The key is passed into the module via API input parameters in plaintext. Keys are passed out of the module via API output parameters in plaintext.	EC_KEY_free()
Shared secret	Generated during the Diffie-Hellman or EC Diffie-Hellman key agreement and shared secret computation.	N/A	DH_free(), EC_KEY_free()
Password or passphrase	Not Applicable. Key material is entered via API parameters.	The key is passed into the module via API input parameters in plaintext.	EVP_PKEY_free()
Derived key	Generated during the TLS KDF, SSH KDF or PBKDF	Keys are passed out of the module via API output parameters in plaintext.	EVP_PKEY_free()
Entropy input string and seed material	Obtained from NDRNG	N/A	FIPS_drbg_free()
DRBG internal state: V value, C value, key (if applicable)	Derived from entropy input as defined in SP800-90A	N/A	FIPS_drbg_free()

Name	Generation	Entry and Output	Zeroization
TLS pre_master_secret	Generated from the SP800-90A DRBG when module acts as a TLS client, for RSA cipher suites.	Received from TLS client (network), wrapped with TLS server's RSA public key, when module acts as a TLS server with RSA cipher suites.	`SSL_free()`, `SSL_clear()`
	Generated during key agreement for Diffie-Hellman or EC Diffie-Hellman cipher suites.	N/A	
TLS master_secret	Derived from TLS pre_master_secret using TLS KDF.	N/A	`SSL_free()`, `SSL_clear()`

Table 10: Life cycle of Keys or CSPs

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 Hash_DRBG, HMAC_DRBG and CTR_DRBG mechanisms. The DRBG is initialized during module initialization; the module loads by default the DRBG using the CTR_DRBG mechanism with AES-256, with derivation function, and 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 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].

6.2 Key/CSP 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 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 keys used in the EC Diffie-Hellman key agreement schemes are generated internally by the module using ECDSA key generation compliant with [FIPS186-4] and [SP800-56ARev3]. The Diffie-Hellman key agreement scheme is also compliant with [SP800-56ARev3], and generates keys using safe primes defined in RFC7919, as described in the next section.

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

6.3 Key Agreement / Key Transport / Key Derivation

The module provides Diffie-Hellman and EC Diffie-Hellman key agreement schemes compliant with SP800-56ARev3, and used as part of the TLS protocol key exchange in accordance with scenario X1 (2) of IG D.8; that is, the shared secret computation (KAS-FFC-SSC and KAS-ECC-SSC) followed by the derivation of the keying material using SP800-135 KDF.

For Diffie-Hellman, the module supports the use of safe primes defined in RFC7919 for domain parameters and key generation. which are used by the TLS key agreement implemented by the module.

- TLS (RFC7919)
  - ffdhe2048 (ID = 256)
  - ffdhe3072 (ID = 257)
  - ffdhe4096 (ID = 258)
  - ffdhe6144 (ID = 259)
  - ffdhe8192 (ID = 260)

The module also supports the use of safe primes defined in RFC3526, which are part of the Modular Exponential (MODP) Diffie-Hellman groups that can be used for Internet Key Exchange (IKE). Note that the module only implements key generation and verification, and shared secret computation of safe primes, and no other part of the IKE.

- IKEv2 (RFC3526)
  - MODP-2048 (ID=14)
  - MODP-3072 (ID=15)
  - MODP-4096 (ID=16)
  - MODP-6144 (ID=17)
  - MODP-8192 (ID=18)

The module also provides the following key transport mechanisms:

- Key wrapping using AES-KW and AES-KWP.
- Key wrapping using AES-CCM, AES-GCM and AES in CBC mode and HMAC, used by the TLS protocol cipher suites with 128-bit or 256-bit keys.
- Key wrapping using Triple-DES in CBC mode and HMAC, used by the TLS protocol cipher suites with 192-bit keys.
- RSA key encapsulation using private key encryption and public key decryption (also used as part of the TLS protocol key exchange).

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

- AES key wrapping using AES in KW, KWP provides between 128 and 256 bits of encryption strength.
- AES key wrapping using AES-CCM, AES-GCM, and AES in CBC mode and HMAC, provides between 128 or 256 bits of encryption strength.
- Triple-DES key wrapping using HMAC provides 112 bits of encryption strength.
- RSA key wrapping[3] provides between 112 and 256 bits of encryption strength.

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

- Diffie-Hellman key agreement provides between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman key agreement provides between 128 and 256 bits of encryption strength.

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

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

- KDF for the TLS protocol, used as pseudo-random functions (PRF) for TLSv1.0/1.1 and TLSv1.2.
- KDF for the SSHv2 protocol.

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.

6.4 Key/CSP Entry and 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 and 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 corresponding API functions listed in Table 10 to zeroize 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 as shown in Table 3 are compliant to 47 CFR FCC Part 15, Subpart B, Class A (Business use).

8 Self Tests

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 section 9.3 and displays the specific error message associated with the returned error code, and then enters the Error state. The subsequent calls to the module will also fail; 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 and decryption (separately tested) KAT AES CCM mode with 192-bit key, encryption and decryption (separately tested) KAT AES GCM mode with 256-bit key, encryption and decryption (separately tested) KAT AES XTS mode with 128 and 256-bit keys, encryption and decryption (separately tested)
CMAC	KAT AES CMAC with 128, 192 and 256 bit keys, MAC generation KAT Triple-DES CMAC, MAC generation
Diffie-Hellman	Primitive “Z” Computation KAT with 2048-bit key
DRBG	KAT CTR_DRBG with AES with 256-bit keys with and without DF, with and without PR KAT Hash_DRBG with SHA-256 with and without PR KAT HMAC_DRBG with SHA-256 with and without PR
DSA	PCT DSA with L=2048, N=224 and SHA-256
EC Diffie-Hellman	Primitive “Z” Computation KAT with P-256 curve

Algorithm	Power-Up Tests
ECDSA	PCT ECDSA with P-256 and SHA-256
HMAC	KAT HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512
PBKDF KDF	KAT with SHA-256
RSA	KAT RSA with 2048-bit key, PKCS#1 v1.5 scheme and SHA-224, SHA-256, SHA-384 and SHA-512, signature generation and verification (separately tested) KAT RSA with 2048-bit key, PSS scheme and SHA-224, SHA-256, SHA-384 and SHA-512, signature generation and verification (separately tested) KAT RSA with 2048-bit key, public key encryption and private key decryption (separately tested)
SHS[4]	KAT SHA-1, SHA-256 and SHA-512
SSH KDF	KAT with SHA-256
TLS KDF	KAT with SHA-256
Triple-DES	KAT Triple-DES ECB mode, encryption and decryption (separately tested)

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 fails and the module enters the Error state. For the PCT, if the signature generation or verification fails, the module enters the Error state.

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.

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. If the conditional test fails, the module returns an error code and enters the Error state. When the module is in the Error state, no data is output and cryptographic operations are not allowed.

4 SHA-224 and SHA-384 are not required per IG 9.4.

Algorithm	Conditional Tests
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 public encryption and private decryption.

Table 12: Conditional Tests

9 Guidance

9.1 Crypto Officer Guidance

The binaries of the module are contained in the RPM 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 RPM packages contain the FIPS validated module:

Processor Architecture	RPM Packages
Intel 64-bit	libopenssl1_1-1.1.0i-4.51.1.x86_64.rpm libopenssl1_1-hmac-1.1.0i-4.51.1.x86_64.rpm
Intel 32-bit	libopenssl1_1-32bit-1.1.0i-4.51.1.x86_64 libopenssl1_1-hmac-32bit-1.1.0i-4.51.1.x86_64

Table 13: RPM packages

9.1.1 Module Installation

The Crypto Officer can install the RPM packages containing the module as listed in Table 13 using the zypper tool. The integrity of the RPM package is automatically verified during the installation, and the Crypto Officer shall not install the RPM package if there is any integrity error.

9.1.2 Operating Environment Configuration

The operating environment needs to be configured to support FIPS, so the following steps shall be performed with the root privilege:

1. Install the dracut-fips RPM package:

```
# zypper install dracut-fips
```

2. Recreate the INITRAMFS image:

```
# dracut -f
```

3. After regenerating the initrd, the Crypto Officer has to append the following parameter in the /etc/default/grub configuration file in the GRUB_CMDLINE_LINUX_DEFAULT line:

```
fips=1
```

4. After editing the configuration file, please run the following command to change the setting in the boot loader:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<partition of /boot or /boot/efi> must be supplied. The partition can be identified with the command "df /boot" or "df /boot/efi" respectively. For example:

```
# df /boot
Filesystem     1K-blocks     Used     Available     Use%     Mounted on
/dev/sda1      233191        30454    190296        14%      /boot
```

The partition of /boot is located on /dev/sda1 in this example. Therefore, the following string needs to be appended in the aforementioned grub file:

```
"boot=/dev/sda1"
```

5. Reboot to apply these 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 verify it contains a numeric value “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.

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). In addition, key sizes must comply with [SP800-131A].

9.2.1 TLS

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, for Diffie-Hellman only the safe prime groups listed in RFC7919 are approved to be used in FIPS mode.

9.2.2 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.

9.2.3 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 14: Ciphers not allowed in FIPS mode of operation

9.2.4 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 stated in IG A.9, the module implements a check to ensure that the two AES keys used in AES XTS mode 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.

9.2.5 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 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”); thus, the module is compliant with [SP800-52].

When a GCM IV is used for decryption, the responsibility for the IV generation lies with the party that performs the AES GCM encryption and therefore there is no restriction on the IV generation.

9.2.6 Triple-DES encryption

Data encryption using the same three-key Triple-DES key shall not exceed 2^{16} Triple-DES blocks (2GB of data), in accordance to SP800-67 and IG A.13.

[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.

9.2.7 Environment Variables

OPENSSL_ENFORCE_MODULUS_BITS

Setting the environment variable OPENSSL_ENFORCE_MODULUS_BITS can restrict the module to only generate the acceptable key sizes of RSA. If the environment variable is set, the module enforces the generation of keys of 2048 bits or more.

9.2.8 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] and IG D.6, 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].

9.3 Handling FIPS Related Errors

When the module fails any power-on self-test or conditional test, the module will return an error code to indicate the error and will enter the Error state. Any further cryptographic operation is inhibited.

The calling application can obtain the module state by calling the `FIPS_selftest_failed() API function`. The function returns 1 if the module is in the Error state, 0 if the module is in the Operational state.

The following table shows the error codes and the corresponding condition:

Error Message / Codes	Error Condition
FIPS_R_FINGERPRINT_DOES_NOT_MATCH (110)	The integrity test fails at power-up.
FIPS_R_SELFTEST_FAILED (101)	Any of the AES, CMAC, DRBG, HMAC, SHA, or Triple-DES KATs fails at power-up.
FIPS_R_TEST_FAILURE (117)	Any of the KATs for RSA, the PCT for ECDSA or the PCT for DSA fails at power-up.
FIPS_R_NOPR_TEST1_FAILURE (145) FIPS_R_NOPR_TEST2_FAILURE(146) FIPS_R_PR_TEST1_FAILURE (147) FIPS_R_PR_TEST2_FAILURE (148)	The KAT of a DRBG fails at power-up.
FIPS_R_FIPS_SELFTEST_FAILED (106)	A cryptographic operation is invoked and the module is in the error state.
FIPS_R_PAIRWISE_TEST_FAILED (127)	The PCT of a newly generated RSA, DSA or ECDSA key pair fails during conditional tests.
FIPS_R_ENTROPY_SOURCE_STUCK (142)	The CRNGT for the NDRNG fails during conditional tests.

Table 15: Error Codes and Error Events

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 setup 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.

10.2 Weak Triple-DES Key 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:

```
/* Weak and semi week keys as taken from
 * %A D.W. Davies
 * %A W.L. Price
 * %T Security for Computer Networks
 * %I John Wiley & Sons
 * %D 1984
 * Many thanks to smb@ulysses.att.com (Steven Bellovin) for the reference
 * (and actual cblock values).
 */
#define NUM_WEAK_KEY    16
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}};
```

Please note that there is no weak key detection by default. The caller can explicitly set the DES_check_key to 1 or call DES_check_key_parity() and/or DES_is_weak_key() functions on its own.

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_DH_DSS_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DH_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_DH_DSS_WITH_AES_128_CBC_SHA	RFC3268
TLS_DH_RSA_WITH_AES_128_CBC_SHA	RFC3268
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	RFC3268
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	RFC3268
TLS_DH_anon_WITH_AES_128_CBC_SHA	RFC3268
TLS_RSA_WITH_AES_256_CBC_SHA	RFC3268
TLS_DH_DSS_WITH_AES_256_CBC_SHA	RFC3268
TLS_DH_RSA_WITH_AES_256_CBC_SHA	RFC3268
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	RFC3268
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	RFC3268
TLS_DH_anon_WITH_AES_256_CBC_SHA	RFC3268
TLS_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DH_DSS_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DH_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DH_DSS_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DH_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DH_anon_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DH_anon_WITH_AES_256_CBC_SHA256	RFC5246

Cipher Suite	Reference
TLS_PSK_WITH_3DES_EDE_CBC_SHA	RFC4279
TLS_PSK_WITH_AES_128_CBC_SHA	RFC4279
TLS_PSK_WITH_AES_256_CBC_SHA	RFC4279
TLS_RSA_WITH_AES_128_GCM_SHA256	RFC5288
TLS_RSA_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DH_RSA_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DH_RSA_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DH_DSS_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DH_DSS_WITH_AES_256_GCM_SHA384	RFC5288
TLS_DH_anon_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DH_anon_WITH_AES_256_GCM_SHA384	RFC5288
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDH_ECDSA_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_RSA_WITH_3DES_EDE_CBC_SHA	RFC4492
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	RFC4492
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	RFC4492
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_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_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	RFC5289
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	RFC5289

Cipher Suite	Reference
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	RFC5289
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	RFC5289
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDH_RSA_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_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	RFC5289
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	RFC5289
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	RFC5289

Table 16: TLS Cipher Suites

Appendix B - CAVP certificates

The table below shows the certificates obtained from the CAVP that validate all algorithm implementations used as approved or allowed security functions in FIPS mode of operation. The table includes the certificate number, the processor architecture tested, the label used in the CAVP and a description of the algorithm implementation.

CAVP#	PA	CAVP Label	Algorithm Implementation
A76	64-bit	DRBG_10X_SHA_ASM	HMAC_DRBG and Hash_DRBG with SHA assembler implementation.
A77	64-bit	BAES_CTASM_AVX	AES-GCM using SSSE3 instruction for Constant Time assembler and Bit Slice AES, and AVX instruction for multiplication and GHASH.
A78	64-bit	AESNI_AVX	AES-GCM using AESNI instructions, and AVX instruction for multiplication and GHASH.
A79	64-bit	DRBG_10X_AESASM	CTR_DRBG with AES assembler implementation.
A80	64-bit	SHA_SSSE3	SHA using SSSE3 instruction.
A81	64-bit	SHA_AVX	SHA using AVX instruction.
A82	64-bit	SHA_AVX2	SHA using AVX2 instruction.
A83	64-bit	DRBG_10X_BAES_CTASM	CTR_DRBG with AES using SSSE3 instruction for Constant Time assembler and Bit Slice AES.
A84	64-bit	DRBG_10X_SHA_AVX2	HMAC_DRBG and Hash_DRBG with SHA using AVX2 instruction.
A85	64-bit	SHA_ASM	SHA assembler implementation.
A86	64-bit	AESASM	AES assembler implementation.
A87	64-bit	DRBG_10X_SHA_SSSE3	HMAC_DRBG and Hash_DRBG with SHA using SSSE3 instruction.
A88	64-bit	DRBG_10X_SHA_AVX	HMAC_DRBG and Hash_DRBG with SHA using AVX instruction.
A89	64-bit	AESNI_CLMULNI	AES-GCM using AESNI instructions, and PCLMULQDQ instruction for multiplication and GHASH.
A90	64-bit	BAES_CTASM	AES using SSSE3 instruction for Constant Time assembler and Bit Slice AES.
A91	64-bit	AESNI_ASM	AES-GCM using AESNI, and assembler implementation for multiplication and GHASH.
A92	64-bit	AESASM_CLMULNI	AES-GCM using assembler implementation, and PCLMULQDQ instruction for multiplication and GHASH.
A93	64-bit	DRBG_10X_AESNI	CTR_DRBG with AES using AESNI instructions.
A94	64-bit	BAES_CTASM_ASM	AES-GCM using SSSE3 instruction for Constant Time assembler and Bit Slice, and assembler implementation for multiplication and GHASH.
A95	64-bit	AESASM_AVX	AES-GCM using assembler implementation, and AVX instruction for multiplication and GHASH.
A96	64-bit	SSH_ASM	KDF SSH with SHA assembler implementation.
A97	64-bit	TDES_C	Triple-DES C implementation
A98	64-bit	SSH_AVX	KDF SSH with SHA using AVX instruction.

CAVP#	PA	CAVP Label	Algorithm Implementation
A99	64-bit	AESNI	AES using AESNI instructions.
A100	64-bit	BAES_CTASM_CLMULNI	AES-GCM using SSSE3 instruction for Constant Time assembler and Bit Slice, and PCLMULQDQ instruction for multiplication and GHASH.
A101	64-bit	AESASM_ASM	AES-GCM using assembler implementation.
A102	64-bit	SSH_SSSE3	KDF SSH with SHA using SSSE3 instruction.
A103	64-bit	SSH_AVX2	KDF SSH with SHA using AVX2 instruction.
A677	64-bit	SP800 56A rev 3	SP800-56ARev3 compliant implementation

Table 17: CAVP certificates for 64-bit algorithm implementations

CAVP#	PA	CAVP Label	Algorithm Implementation
A186	32-bit	SHA_AVX	SHA using AVX instruction.
A187	32-bit	BAES_CTASM_AVX	AES-GCM using SSSE3 instruction for Constant Time assembler and Bit Slice AES, and AVX instruction for multiplication and GHASH.
A188	32-bit	DRBG_10X_AESNI	CTR_DRBG with AES using AESNI instructions.
A189	32-bit	AESNI_CLMULNI	AES-GCM using AESNI instructions, and PCLMULQDQ instruction for multiplication and GHASH.
A190	32-bit	SSH_ASM	KDF SSH with SHA assembler implementation.
A191	32-bit	DRBG_10X_SHA_ASM	HMAC_DRBG and Hash_DRBG with SHA assembler implementation.
A192	32-bit	AESASM_CLMULNI	AES-GCM using assembler implementation, and PCLMULQDQ instruction for multiplication and GHASH.
A193	32-bit	DRBG_10X_SHA_SSSE3	HMAC_DRBG and Hash_DRBG with SHA using SSSE3 instruction.
A194	32-bit	DRBG_10X_BAES_CTASM	CTR_DRBG with AES using SSSE3 instruction for Constant Time assembler and Bit Slice AES.
A195	32-bit	SHA_AVX2	SHA using AVX2 instruction.
A196	32-bit	AESASM	AES assembler implementation.
A197	32-bit	TDES_C	Triple-DES C implementation
A198	32-bit	DRBG_10X_SHA_AVX2	HMAC_DRBG and Hash_DRBG with SHA using AVX2 instruction.
A199	32-bit	DRBG_10X_SHA_AVX	HMAC_DRBG and Hash_DRBG with SHA using AVX instruction.
A202	32-bit	SHA_SSSE3	SHA using SSSE3 instruction.
A203	32-bit	SHA_ASM	SHA assembler implementation.
A200	32-bit	DRBG_10X_AESASM	CTR_DRBG with AES assembler implementation.
A201	32-bit	BAES_CTASM	AES using SSSE3 instruction for Constant Time assembler and Bit Slice AES.
A204	32-bit	SSH_AVX	KDF SSH with SHA using AVX instruction.

CAVP#	PA	CAVP Label	Algorithm Implementation
A205	32-bit	SSH_SSSE3	KDF SSH with SHA using SSSE3 instruction.
A206	32-bit	BAES_CTASM_ASM	AES-GCM using SSSE3 instruction for Constant Time assembler and Bit Slice, and assembler implementation for multiplication and GHASH.
A207	32-bit	AESASM_ASM	AES-GCM using assembler implementation.
A208	32-bit	AESNI_AVX	AES-GCM using AESNI instructions, and AVX instruction for multiplication and GHASH.
A209	32-bit	AESASM_AVX	AES-GCM using assembler implementation, and AVX instruction for multiplication and GHASH.
A210	32-bit	BAES_CTASM_CLMULNI	AES-GCM using SSSE3 instruction for Constant Time assembler and Bit Slice, and PCLMULQDQ instruction for multiplication and GHASH.
A211	32-bit	AESNI_ASM	AES-GCM using AESNI, and assembler implementation for multiplication and GHASH.
A212	32-bit	AESNI	AES using AESNI instructions.
A213	32-bit	SSH_AVX2	KDF SSH with SHA using AVX2 instruction.
A676	32-bit	SP800 56A rev 3	SP800-56ARev3 compliant implementation

Table 18: CAVP certificates for 32-bit algorithm implementations

Appendix C - Glossary and Abbreviations

AES	Advanced Encryption Specification
AES_NI	Intel® Advanced Encryption Standard (AES) New Instructions
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining Message Authentication Code
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
HMAC	Hash Message Authentication Code
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
PKCS	Public Key Cryptography Standards
RNG	Random Number Generator
RPM	Red hat Package Manager
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
TDES	Triple-DES
XTS	XEX Tweakable Block Cipher with Ciphertext Stealing

Appendix D - References

FIPS 140-2	**FIPS PUB 140-2 - Security Requirements for Cryptographic Modules** http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
FIPS 140-2_IG	**Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program** December 3, 2019 http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf
FIPS180-4	**Secure Hash Standard (SHS)** http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
FIPS186-4	**Digital Signature Standard (DSS)** http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1** http://www.ietf.org/rfc/rfc3447.txt
RFC2246	**The TLS Protocol Version 1.0** https://www.ietf.org/rfc/rfc2246.txt
RFC3268	**Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)** https://www.ietf.org/rfc/rfc3268.txt
RFC4279	**Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)** https://www.ietf.org/rfc/rfc4279.txt
RFC4346	**The Transport Layer Security (TLS) Protocol Version 1.1** https://www.ietf.org/rfc/rfc4346.txt
RFC4492	**Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)** https://www.ietf.org/rfc/rfc4492.txt
RFC5116	**An Interface and Algorithms for Authenticated Encryption** https://www.ietf.org/rfc/rfc5116.txt
RFC5246	**The Transport Layer Security (TLS) Protocol Version 1.2** https://tools.ietf.org/html/rfc5246.txt
RFC5288	**AES Galois Counter Mode (GCM) Cipher Suites for TLS** https://tools.ietf.org/html/rfc5288.txt
RFC5487	**Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode** https://tools.ietf.org/html/rfc5487.txt

RFC5489	**ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)** https://tools.ietf.org/html/rfc5489.txt
RFC6655	**AES-CCM Cipher Suites for Transport Layer Security (TLS)** https://tools.ietf.org/html/rfc6655.txt
RFC7251	**AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS** https://tools.ietf.org/html/rfc7251.txt
RFC7296	**Internet Key Exchange Protocol Version 2 (IKEv2)** https://tools.ietf.org/html/rfc7296
SP800-38A	**NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques** http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf
SP800-38B	**NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication** http://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** 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** http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf
SP800-38E	**NIST Special Publication 800-38E - Recommendation for Block Cipher Modes of Operation: The XTS AES Mode for Confidentiality on Storage Devices** http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38e.pdf
SP800-38F	**NIST Special Publication 800-38F - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping** http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf
SP800-56ARev3	**NIST Special Publication 800-56A Revision 3 - Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography** https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf
SP800-67	**NIST Special Publication 800-67 Revision 1 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-67r1.pdf
SP800-90A	**NIST Special Publication 800-90A Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

SP800-131A	**NIST Special Publication 800-131A Revision 1- Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths** http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-132	**NIST Special Publication 800-132 - Recommendation for Password-Based Key Derivation - Part 1: Storage Applications** https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf
SP800-135r1	**NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions** https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf