SUSE Linux Enterprise Server OpenSSL Module version 3.0

FIPS 140-2 Non-Proprietary Security Policy

Doc version 1.1

Last update: 2017-10-02

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Contents

1 Cryptographic Module Specification

This document is the non-proprietary security policy for the SUSE Linux Enterprise Server OpenSSL Module, and was prepared as part of the requirements for conformance to Federal Information Processing Standard (FIPS) 140-2, Level 1.

1.1 Description of the Module

The SUSE Linux Enterprise Server OpenSSL Module (hereafter referred to as the "Module") is a software library implementing the Transport Layer Security (TLS) protocol v1.0, v1.1, and v1.2, and the Datagram Transport Layer Security (DTLS) protocol v1.0 and v1.2, as well as supporting FIPS 140-2 Approved cryptographic algorithms. The current version of the Module is 3.0. An earlier version of this Module has gone through FIPS 140-2 validation under certificate #2435.

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 FIPS 140-2 purposes, the Module is classified as a multi-chip standalone module validated at security level 1. The following table shows the claimed security level for each of the eleven sections comprising the FIPS 140-2 standard.

Security Component	Security Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self Tests	1
Design Assurance	1
Mitigation of Other Attacks	1

Table 1: Security Level of the Module

The Module's logical cryptographic boundary is the shared library files and their integrity check HMAC files that are delivered with the following RPM packages:

- 64-bit libssl and libcrypto shared libraries (libssl.so.1.0.0 and libcrypto.so.1.0.0) delivered in

libopenssl1_0_0-1.0.2j-60.11.2.x86_64.rpm and libopenssl1_0_0-1.0.2j-60.11.2.s390x.rpm – note that the RPM also delivers other shared libraries implementing the OpenSSL engines which are not part of the Module.

- HMAC integrity verification files for the 64-bit shared libraries (.libssl.so.1.0.0.hmac and .libcrypto.so.1.0.0.hmac) delivered in libopenssl1_0_0-hmac-1.0.2j-60.11.2.x86_64.rpm and libopenssl1_0_0-hmac-1.0.2j-60.11.2.s390x.rpm.

As shown in the following figure, the Module's logical boundary is the rectangle containing the libcrypto shared libary, libssl shared library, and the associated HMAC files.

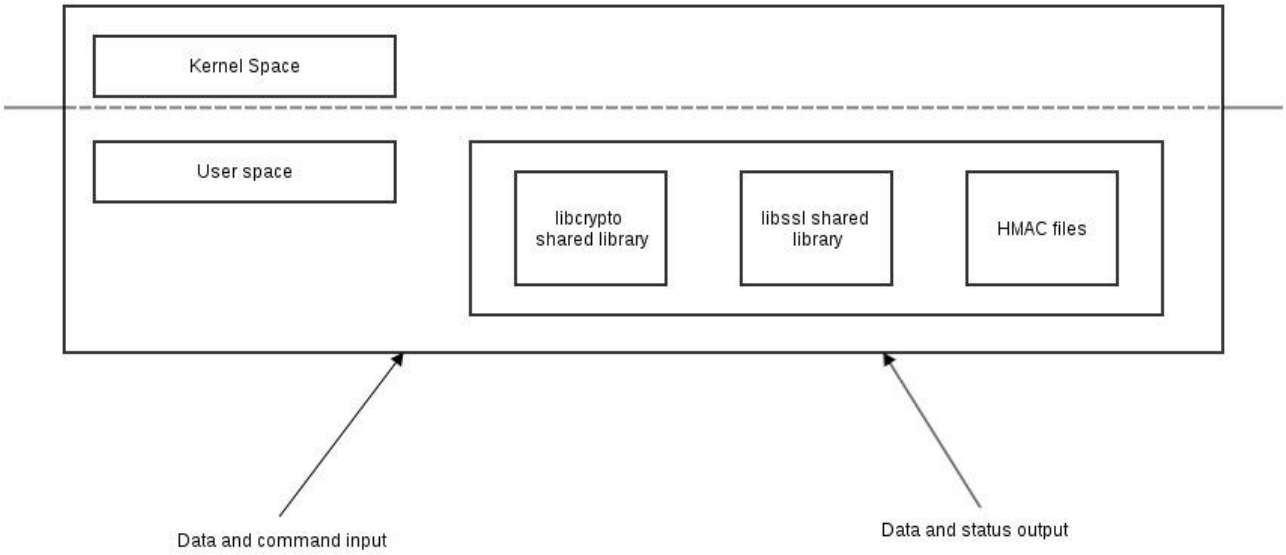

Figure 1: Software Block Diagram

The Module has been tested on the following multi-chip standalone platforms:

Platform	Processor	Test Configuration
FUJITSU Server PRIMERGY CX2570 M2 inside a CX400 M1 enclosure	Intel Xeon E5 family	SUSE Linux Enterprise Server 12 SP2 with and without AES-NI (PAA)
IBM z13	z13	SUSE Linux Enterprise Server 12 SP2 with and without CPACF (PAI)

Table 2: 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 module is aimed to run on a general-purpose computer (GPC). The Module's physical boundary is the tested platforms (depicted in Figure 2).

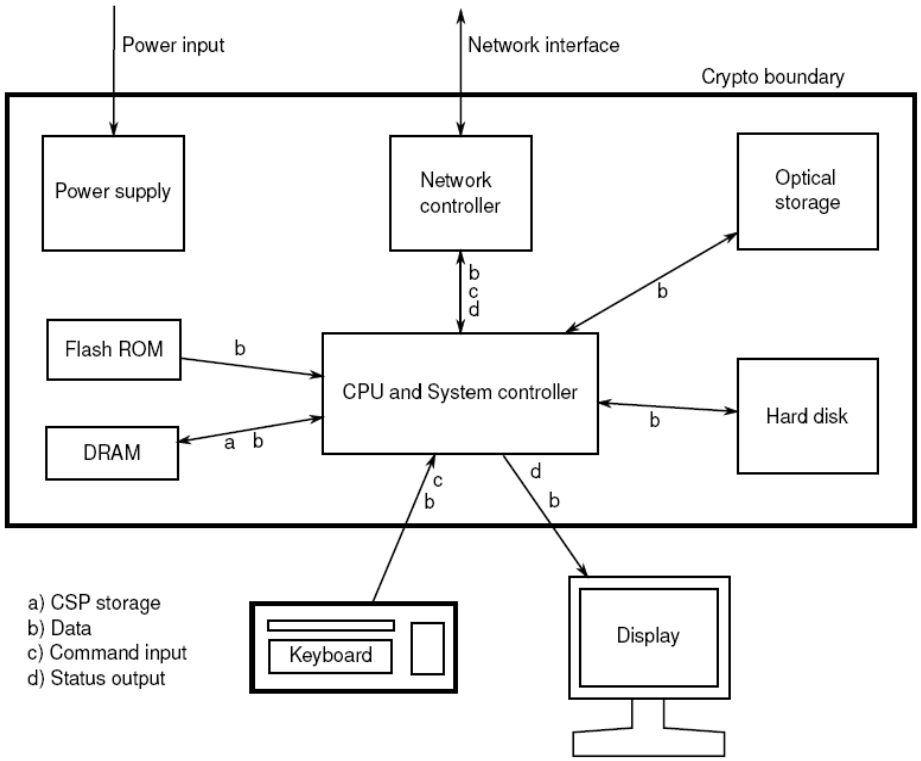

Figure 2: Hardware Block Diagram

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 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, 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	Keyboard, Serial port, Ethernet port	API function calls, API input parameters for control, TLS protocol internal state.
Status Output	Serial port, Ethernet port	API return codes, TLS protocol internal state.
Power Input	PC Power Supply Port	N/A

Table 3: Ports and Interfaces

3 Roles, Services, and Authentication

This section defines the roles, services and authentication mechanisms, and methods with respect to the applicable FIPS 140-2 requirements.

3.1 Roles

The Module assumes two roles: User role and Crypto Officer role, which are identified along with their allowed services in Table 4 (the services are further detailed in Table 5 and Table 6).

Role	Descriptions
User	Perform general security services, including Approved and non-Approved security functions. (Please see Table 5 and Table 6 for more details)
Crypto Officer	Perform Module installation and initialization.

Table 4: Roles

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

3.2 Services

The Module supports services that are available to users in the various roles. All of the services are described in detail in the Manual Pages. The introduction page is crypto(3) for the crypto operations and ssl(3) for the SSL/TLS protocol API.

The following table lists the Approved or non-Approved but allowed services available in FIPS Approved mode. Please refer to Table 7, Table 8 and Table 9 for the Approved or Allowed key size of each algorithm used in the services.

Service	Role	Algorithm	Keys/CSPs	Access
Cryptographic Library Services				
Symmetric encryption/decryption	User	AES or Triple-DES	AES or Triple-DES key	read
RSA key generation	User	RSA, DRBG	RSA public-private key	create
RSA signature generation/verification	User	RSA	RSA public-private key	read
DSA key generation	User	DSA, DRBG	DSA public-private key	create
DSA signature generation/verification	User	DSA	DSA public-private key	read
ECDSA key generation	User	ECDSA, DRBG	ECDSA public-private key	create
ECDSA signature generation/verification	User	ECDSA	ECDSA public-private key	read
ECDSA public key validation	User	ECDSA	ECDSA public key	read
Random number generation	User	DRBG	Seed, entropy input string and internal state	read, update
Message digest	User	SHA-1, SHA224, SHA256, SHA-384, SHA-512	N/A	N/A
Message authentication code (MAC)	User	HMAC	HMAC key	read
	User	CMAC with AES	AES key	read
	User	CMAC with Triple-DES	Triple-DES key	read
Key wrapping	User	AES	AES key	read
	User	RSA	RSA private key	read
Diffie-Hellman Key Agreement	User	KAS FFC	Diffie-Hellman domain parameters	create, read
EC Diffie-Hellman Key Agreement	User	KAS ECC, ECC CDH primitive	EC Diffie-Hellman public-private keys	create, read
Network Protocol Services				
Transport Layer Security (TLS) network protocol v1.0, v1.1 and v1.2	User	See Appendix A for a complete list of supported cipher suites	AES or Triple-DES key, RSA, DSA or ECDSA private key, HMAC key, Pre-Master Secret, Master Secret, Diffie-Hellman domain parameters or EC Diffie-Hellman public-private keys	read
TLS extensions	User	N/A	RSA, DSA or ECDSA public-private key	read
Certificate Management	User	N/A	RSA, DSA or ECDSA public-private key	read
Other FIPS Related Services				
Show status	User	N/A	None	N/A
Module installation	Crypto Officer	N/A	None	N/A
Module initialization	Crypto Officer	N/A	None	N/A

Self-tests	User	AES, Triple-DES, SHS, HMAC, DSA, ECDSA, RSA, DRBG, Diffie-Hellman, EC DIffie-Hellman	HMAC-SHA-256 key for integrity test	read
Zeroize	User	N/A	All aforementioned CSPs	Zeroize

Table 5: Approved Service Details

The following table lists the non-Approved services available in non-FIPS mode. Please refer to Table 10 for the non-Approved key size or algorithm.

Service	Role
Diffie-Hellman key agreement using non-Approved key size	User
RSA key encapsulation using non-Approved RSA key size	User
Asymmetric key generation using non-Approved RSA or DSA key size	User
Digital signature generation/verification using non-Approved RSA or DSA key size	User
Random number generation using ANSI X9.31 RNG	User
Message digest (MD2, MD4, MD5, MDC-2, HMAC-MD5, RIPEMD160)	User
Symmetric encryption/decryption (Blowfish, Camellia, CAST, DES, IDEA, RC2, RC4, RC5, SEED)	User
Key agreement by using JPAKE	User
TLS-SRP key exchange	User
Whirlpool hash function	User

Table 6: Non-Approved Service Details

3.3 Operator Authentication

At security level 1, authentication is neither required nor employed. The role is implicitly assumed based on the service requested.

3.4 Algorithms

The Module provides multiple implementations of algorithms. Different implementations can be invoked by setting the environment variable. Please note that only one implementation will be available at runtime. For TLS protocol, only the key derivation function (KDF) has been tested by the CAVP.

3.4.1 Running on Intel Xeon Processor

On the platform that runs Intel Xeon processor, the module supports the use of AES-NI (by default), SSSE3 and generic assembler for AES implementation, the use of AVX2, AVX, SSSE3 and generic assembler for SHA implementation, and the use of CLMUL instruction set and generic assembler for GHASH that is used for GCM mode. Each implementation is determined by the environment variable OPENSSL_ia32cap.

The following table shows the CAVS certificates and their associated information of the cryptographic implementation in FIPS mode.

Algorithm	CAVS Cert	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
AES	#4588 (using AES assembler for AES, and CLMUL for GHASH) #4594 (using SSSE3 assembler for AES, and CLMUL for GHASH) #4595 (using AES-NI for AES, and CLMUL for GHASH)	FIPS197, SP800-38A	ECB, CBC, CFB1, CFB8, CFB128, OFB, 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
		SP800-38D	GCM	128, 192, 256	Data Encryption and Decryption
		SP800-38E	XTS	128, 256	Data Encryption and Decryption for Data Storage
		SP800-38F	KW	128, 192, 256	Key Wrapping and Unwrapping
	#4645 (using AES-NI for AES, and assembler for GHASH) #4646 (using AES assembler for AES, and assembler for GHASH) #4647 (using SSSE3 assembler for AES, and assembler for GHASH)	SP800-38D	GCM	128, 192, 256	Data Encryption and Decryption
Diffie-Hellman	CVL #1263	SP800-56A (All except KDF)	FCC dhEphem scheme	p=2048, q=224; p=2048, q=256	Diffie-Hellman Key Agreement
EC Diffie-Hellman	CVL #1263	SP800-56A (All except KDF)	ECC Ephemeral Unified scheme	P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement
ECC CDH Primitive	CVL #1263	SP800-56A Section 5.7.1.2	N/A	P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement

Algorithm	CAVS Cert	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
TLS KDF	CVL #1264	SP800-135	TLS v1.0, v1.1 and v1.2	N/A	Key Derivation
DRBG	#1536 (using AVX2 for SHA) #1537 (using AVX for SHA) #1538 (using SSSE3 for SHA[1]) #1539 (using SHA assembler)	SP800-90A	**Hash_DRBG**: **HMAC_DRBG**: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Deterministic Random Bit Generation
	#1531 (using AES assembler for AES) #1535 (using SSSE3 for AES) #1540 (using AES-NI for AES)		**CTR_DRBG**: AES-128, AES-192, AES-256		
DSA	#1220	FIPS186-4	SHA-1[2], SHA-224, SHA-256, SHA-384, SHA-512	L=1024, N=160[3]; L=2048, N=224; L=2048, N=256; L=3072, N=256	Key Pair Generation, Domain Parameter Generation and Verification, Digital Signature Generation and Verification
ECDSA	#1127	FIPS186-4	SHA-1[4], SHA-224, SHA-256, SHA-384, SHA-512	P-192[5], P-224, P-256, P-384, P-521	Key Pair Generation, Public Key Validation, Digital Signature Generation and Verification

[1] The module only supports SHA-1, SHA-224 and SHA-256 when using SSSE3 implementation for SHA.

[2] SHA-1 is only allowed and CAVS tested in DSA Domain Parameter Verification and DSA Signature Verification for legacy use.

[3] 1024-bit key is only allowed and CAVS tested in DSA Domain Parameter Verification and DSA Signature Verification for legacy use.

[4] SHA-1 is only allowed and CAVS tested in ECDSA Public Key Validation and ECDSA Signature Verification for legacy use.

[5] P-192 curve is only allowed and CAVS tested in ECDSA Public Key Validation and ECDSA Signature Verification for legacy use.

Algorithm	CAVS Cert	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
HMAC	#3042 (using AVX2 for SHA) #3043 (using AVX for SHA) #3044 (using SSSE3 for SHA[1]) #3045 (using SHA assembler)	FIPS198-1	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code
RSA	#2505	FIPS186-4	**X9.31** SHA-1[6], SHA-256, SHA-384, SHA-512 **PKCS#1v1.5** SHA-1[6], SHA-224, SHA-256, SHA-384, SHA-512 **PSS** SHA-1[6], SHA-224, SHA-256, SHA-384, SHA-512	1024[7], 2048, 3072, 4096[8]	Key Pair Generation, Digital Signature Generation and Verification
SHS	#3768 (using AVX2 for SHA) #3769 (using AVX for SHA) #3770 (using SSSE3 for SHA[1]) #3771 (using SHA assembler)	FIPS180-4	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest
Triple-DES	#2439	SP800-67, SP800-38A	ECB, CBC, CFB1, CFB8, CFB64, OFB, CTR	192	Data Encryption and Decryption

[6] SHA-1 is only allowed and CAVS tested in RSA Signature Verification for legacy use.
[7] 1024-bit key is only allowed and CAVS tested in RSA Signature Verification for legacy use.
[8] 4096-bit key is only CAVS tested for RSA Signature Generation.

Algorithm	CAVS Cert	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
		SP800-67, SP800-38B	CMAC	192	MAC Generation and Verification

Table 7 Cryptographic Algorithms for Intel Xeon Processor

3.4.2 Running on z13 Processor

On the platform that runs z system, the module supports the use of CPACF (by default) or generic assembler for AES, SHA and GHASH implementations. Each implementation is determined by the environment variable OPENSSL_s390xcap.

The following table shows the CAVS certificates and their associated information of the cryptographic implementation in FIPS mode.

Algorithm	CAVS Cert	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
AES	#4622 (using AES assembler) #4623 (using AES and GHASH from CPACF)	FIPS197, SP800-38A	ECB, CBC, CFB1, CFB8, CFB128, OFB, 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
		SP800-38D	GCM	128, 192, 256	Data Encryption and Decryption
		SP800-38E	XTS	128, 256	Data Encryption and Decryption for Data Storage
		SP800-38F	KW	128, 192, 256	Key Wrapping and Unwrapping
Diffie-Hellman	CVL #1276	SP800-56A (All except KDF)	FCC dhEphem scheme	p=2048, q=224; p=2048, q=256	Diffie-Hellman Key Agreement

Algorithm	CAVS Cert	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
EC Diffie-Hellman	CVL #1276	SP800-56A (All except KDF)	ECC Ephemeral Unified scheme	P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement
ECC CDH Primitive	CVL #1276	SP800-56A Section 5.7.1.2	N/A	P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement
TLS KDF	CVL #1359	SP800-135	TLS v1.0, v1.1 and v1.2	N/A	Key Derivation
DRBG	#1552 (using SHA and AES assembler) #1553 (using SHA and AES from CPACF)	SP800-90A	**Hash_DRBG:** **HMAC_DRBG:** SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 **CTR_DRBG:** AES-128, AES-192, AES-256	N/A	Deterministic Random Bit Generation
DSA	#1221	FIPS186-4	SHA-1[2], SHA-224, SHA-256, SHA-384, SHA-512	L=1024, N=160[3]; L=2048, N=224; L=2048, N=256; L=3072, N=256	Key Pair Generation, Domain Parameter Generation and Verification, Digital Signature Generation and Verification
ECDSA	#1131	FIPS186-4	SHA-1[4], SHA-224, SHA-256, SHA-384, SHA-512	P-192[5], P-224, P-256, P-384, P-521	Key Pair Generation, Public Key Validation, Digital Signature Generation and Verification

Algorithm	CAVS Cert	Standard	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use
HMAC	#3059 (using SHA assembler) #3060 (using SHA from CPACF)	FIPS198-1	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code
RSA	#2519	FIPS186-4	**X9.31** SHA-1[6], SHA-256, SHA-384, SHA-512 **PKCS#1v1.5** SHA-1[6], SHA-224, SHA-256, SHA-384, SHA-512 **PSS** SHA-1[6], SHA-224, SHA-256, SHA-384, SHA-512	1024[7], 2048, 3072, 4096[8]	Key Pair Generation, Digital Signature Generation and Verification
SHS	#3788 (using SHA assembler) #3789 (using SHA from CPACF)	FIPS180-4	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest
Triple-DES	#2455	SP800-67, SP800-38A	ECB, CBC, CFB1, CFB8, CFB64, OFB, CTR	192	Data Encryption and Decryption
		SP800-67, SP800-38B	CMAC	192	MAC Generation and Verification

Table 8: Cryptographic Algorithms for z Systems z13 Processor

3.4.3 Non-Approved Algorithms

The Module supports the following FIPS 140-2 non-Approved algorithms which are allowed for use in FIPS Approved mode:

Algorithm	Usage
RSA Key Encapsulation with Encryption and Decryption Primitives and key size >= 2048-bit	Key Establishment; allowed in [FIPS140-2_IG] D.9
Diffie-Hellman with key size >= 2048-bit (CVL certs #1263, #1276)	Key Agreement; allowed in [FIPS140-2_IG] D.8
EC Diffie-Hellman with P-224, P-256, P-384, P-521 curves (CVL certs #1263, #1276)	Key Agreement; allowed in [FIPS140-2_IG] D.8
RSA Key Generation and Digital Signature Verification with key size > 3072 bits, and Digital Signature Generation with key size > 4096-bit	Digital Signature; allowed in [SP800-131A]
DSA Key Generation, Domain Parameter Generation and Verification, Digital Signature Generation and Verification with key size > 3072-bit	Digital Signature; allowed in [SP800-131A]
MD5[9]	Pseudo-random function (PRF) in TLS v1.0 and v1.1; allowed in [SP800-52]
SHA-1 used in the Digital Signature Generation[10]	Digital Signature Generation in TLS; allowed in [SP800-52]
NDRNG	The module obtains the entropy data from NDRNG to seed the DRBG.

Table 9: Non-Approved but Allowed Algorithms

The Module supports the following FIPS 140-2 non-Approved algorithms:

Algorithm	Usage
Diffie-Hellman with key size < 2048-bit	Key Agreement with non-Approved key size
RSA with key size < 2048-bit	Key Pair Generation, Digital Signature Generation, Key Encapsulation with non-Approved key size
RSA with key size < 1024-bit	Digital Signature Verification with non-Approved key size
DSA with key size < 2048-bit	Key Pair Generation, Domain Parameters Generation, Digital Signature Generation with non-Approved key size
DSA with key size < 1024-bit	Digital Signature Verification with non-Approved key size
ANSI X9.31 RNG	non-Approved Random Number Generation
MD2, MD4, MD5, MDC-2, HMAC-MD5,	non-Approved Message Digest

[9] 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].

[10] According [SP800-52], SHA-1 is disallowed for Key Pair Generation and Digital Signature Generation, with the exception of digital signatures on ephemeral parameters in TLS.

RIPEMD160	
Blowfish, Camellia, CAST, DES, IDEA, RC2, RC4, RC5, SEED	non-Approved Data Encryption / Decryption
JPAKE	non-Approved Key Agreement
TLS-SRP	non-Approved Key Exchange
Whirlpool	non-Approved Hash

Table 10: Non-Approved Algorithms

The non-Approved algorithms shall not be used in the FIPS Approved mode. Any use of these non-Approved algorithm functions will cause the Module to operate in the non-Approved mode implicitly.

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

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 platforms specified in Table 2.

5.2 Policy

The operating system is restricted to a single operator mode of operation (i.e., concurrent operators are explicitly excluded).

The application that makes calls to the cryptographic Module is the single user of the cryptographic Module, even when the application is serving multiple clients.

The ptrace(2) system call, the debugger (gdb(1)), and strace(1) shall not be used.

6 Cryptographic Key Management

The application that uses the Module is responsible for appropriate destruction and zeroization of the key material. The library provides functions for key allocation and destruction, which overwrites the memory that is occupied by the key information with "zeros" before it is reallocated.

The management of all keys/CSPs used by the Module is summarized in the table below.

Key/CSP	Generation	Entry/Output	Zeroization
AES keys	N/A.	The key is passed into the module via API input parameters in plaintext.	EVP_CIPHER_CTX_cleanup ()
Triple-DES keys			EVP_CIPHER_CTX_cleanup ()
HMAC key			HMAC_CTX_cleanup()
RSA public-private keys	The public-private keys are generated using FIPS 86-4 Key Generation method, and the random value used in the key generation is generated using SP800-90A DRBG.	The key is passed into the module via API input parameters in plaintext. The key is passed out of the module via API output parameters in plaintext.	RSA_free()
DSA public-private keys			DSA_free()
ECDSA public-private keys			EC_KEY_free()
Diffie-Hellman domain parameters	The domain parameters used in Diffie-Hellman and the components to generate the public-private keys used in EC Diffie-Hellman are generated using SP800-90A DRBG.	The key is passed into the module via API input parameters in plaintext. The key is passed out of the module via API output parameters in plaintext.	DH_free()
EC Diffie-Hellman public-private keys			EC_KEY_free()
TLS Pre-Master Secret and Master Secret	Generated during the TLS handshake.	None	SSL_free() and SSL_clear()
Entropy input string	Obtained from NDRNG.	None	FIPS_drbg_free()
DRBG internal state (V, C, Key)	During DRBG initialization.	None	FIPS_drbg_free()

Table 11: Key Management Details

6.1 Random Number Generation

The Module employs a SP 800-90A DRBG as random number generator for creation of asymmetric and 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 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 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 module performs conditional self-tests on the output of NDRNG to ensure that consecutive random numbers do not repeat, and performs DRBG health tests as defined in section 11.3 of [SP800-90A].

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

6.2 Key Generation

For generating HMAC keys and symmetric keys, the module does not provide any dedicated key generation service. However, the Random Number Generation service can be called by the user to obtain random numbers which can be used as key material for symmetric algorithms or HMAC. The key material of HMAC keys and symmetric keys may also be generated during the Diffie-Hellman or EC Diffie-Hellman key agreement.

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 also provides key wrapping using the AES with KW mode and 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.

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 provides between 128 and 256 bits of encryption strength.
- RSA key encapsulation provides between 112 and 256 bits of encryption strength.

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

6.4 Key Entry / Output

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

6.5 Key / CSP Storage

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

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

6.6 Key / CSP Zeroization

The memory occupied by keys is allocated by regular memory allocation operating system calls. The application is responsible for calling the appropriate zeroization functions from the OpenSSL Module API. The zeroization functions then overwrite the memory occupied by keys with "zeros" and deallocates the memory. In case of abnormal termination, or swap in/out of a physical memory page of a process, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

7 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The test 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 the error state.

No operator intervention is required during the running of the self-tests.

See section 9.3 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 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 - 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	Test
AES	KAT, encryption and decryption are tested separately
Triple-DES	KAT, encryption and decryption are tested separately
DSA	PCT, signature generation and signature verification are

Algorithm	Test
	tested separately
RSA	KAT, signature generation and signature verification are tested separately
ECDSA	PCT, signature generation and signature verification are tested separately
Diffie-Hellman	Primitive "Z" Computation KAT
EC Diffie-Hellman	Primitive "Z" Computation KAT
CTR_DRBG	KAT
Hash_DRBG	KAT
HMAC_DRBG	KAT
SHA-1	KAT
SHA-224	Tested as part of SHA-256 KAT
SHA-256	KAT
SHA-384	Tested as part of SHA-512 KAT
SHA-512	KAT
HMAC-SHA-1	KAT
HMAC-SHA-224	KAT
HMAC-SHA-256	KAT
HMAC-SHA-384	KAT
HMAC-SHA-512	KAT

Table 12: Module 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 3.4, only one AES or SHA implementation is available at run-time.

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

8.2 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	Test
DSA	PCT for Key Pair Generation
RSA	PCT for Key Pair Generation
ECDSA	PCT for Key Pair Generation
SP 800-90A DRBG	CRNGT

Table 13: Module Conditional Tests

8.3 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.

9 Guidance

Password-based encryption and password-based key generation do not provide sufficient strength to satisfy FIPS 140-2 requirements. As a result, data processed with password-based encryption methods are considered to be unprotected.

9.1 Crypto Officer Guidance

The Module is delivered as a binary object file packaged in an RPM. The integrity of the RPM is automatically verified during the installation and the Crypto officer shall not install the RPM file if the RPM tool indicates an integrity error. The versions of the RPMs containing the validated Module are listed in Section 1.1.

The RPM package of the Module can be installed by standard tools recommended for the installation of RPM packages on a SUSE Linux system.

For proper operation of the in-Module integrity verification, the prelink has to be disabled. This can be done by setting PRELINKING=no in the /etc/sysconfig/prelink configuration file. If the libraries were already prelinked, the prelink should be undone on all the system files using the 'prelink -u -a' command.

ENGINE_register_* and ENGINE_set_default_* function calls are prohibited. Furthermore, FIPS_mode_set( ) with a parameter of zero (0) is prohibited.

The library can be configured to support FIPS in either of the following ways:

Option 1: Boot the system with the kernel command line opetion “fips=1”.

Option 2: Set the environment variable OPENSSL_FORCE_FIPS_MODE to “1”.

Option 1 effects the whole system, including other libraries, and all instances of the OpenSSL module that may be initialized.

Option 2 only effects the OpenSSL module in an environment where the variable is set. Note that some implementations clear the execution environment for child processes, or daemons spawned by other users.

For Option 1, the following steps shall be performed with root privilege:

1. Install the dracut-fips package:

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

2. Recreate the INITRAMFS image:

```
# dracut -f
```

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
```

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 to the kernel command line:

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

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 verified that 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.

If an application that uses the Module for its cryptography is put into a chroot environment, the Crypto Officer must ensure one of the above methods is available to the Module from within the chroot environment to ensure FIPS operation is enabled.

9.2 User Guidance

The Module must be operated in FIPS Approved mode to ensure that FIPS 140-2 validated cryptographic algorithms and security functions are used.

The application can query whether the FIPS operation is active by calling FIPS_mode() and it can query whether an integrity check or KAT self test failed by calling FIPS_selftest_failed().

The Module performs the self tests described in section 8.1. See section 9.3 for descriptions of possible self test errors and recovery procedures.

9.2.1 TLS and Diffie-Hellman

The TLS protocol implementation provides both the server and the client side. As required by SP800-131A, Diffie-Helllman 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 cryptographic Module accepts Diffie-Hellman key sizes smaller than 2048 bits offered by the TLS server.

The TLS server implementation of the cryptographic Module 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:

- when the Module is used as a TLS server, the Diffie-Hellman parameters (dh argument) of the aforementioned API call must be 2048 bits or larger;
- when 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 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.

9.2.3 Random Number Generator

The OpenSSL API call of RAND_cleanup() must not be used. This call will cleanup the internal DRBG state. This call also replaces the DRBG instance with the non-Approved deterministic random number generator when using the RAND_* API calls.

9.2.4 AES GCM IV

AES_GCM is used in version 1.2 or higher of the TLS protocol. The module is compliant with SP 800-52 and the mechanism for IV generation is compliant with RFC 5288. The operations of one of the two parties involved in the TLS key establishment scheme are performed entirely within the cryptographic boundary of the Module.

In case of power loss from the Module, the AES GCM key will be re-negotiated. No IV is stored in memory.

Whenever the nonce_explicit part of the IV has been exhausted, the module will abort the TLS session and re-perform a handshake to establish new keying material."

9.2.5 RSA and DSA Keys

The Module allows the use of 1024 bit RSA and DSA keys for legacy purposes, including signature generation.

As per SP800-131A, RSA and DSA must be used with keys greater than or equal to 2048 bits. To comply with the requirements of FIPS 140-2, a user must therefore only use keys with 2048 bits or more.

9.2.6 Triple-DES Keys

According to IG A.13, the same Triple-DES key shall not be used to encrypt more than 2^{28} 64-bit blocks of data.

9.3 Handling Self Test 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:

FIPS_R_FINGERPRINT_DOES_NOT_MATCH - the integrity verification check failed
FIPS_R_FIPS_SELFTEST_FAILED - a KAT failed for DSA, RSA, ECDSA, Diffie-Hellman or EC Diffie-Hellman
FIPS_R_SELFTEST_FAILED - a KAT failed for AES, Triple-DES, DRBG, SHA, HMAC or CMAC
FIPS_R_TEST_FAILURE – a PCT failed during public key signature test for DSA, ECDSA or RSA
FIPS_R_PAIRWISE_TEST_FAILED – a PCT failed during key generation for DSA, ECDSA or RSA
FIPS_R_DRBG_STUCK – a CRNGT failed for SP 800-90A DRBG

These errors are reported through the regular ERR interface of the shared libraries and can be queried by functions such as ERR_get_error(). See the OpenSSL Module manual page 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.

11 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

Cipher Suite	Reference
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
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

Cipher Suite	Reference
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
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 14: TLS Cipher Suites

12 Glossary and Abbreviations

AES	Advanced Encryption Specification
CAVP	Cryptographic Algorithm Validation Program
CBC	Cypher Block Chaining
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CFB	Cypher Feedback
CLMUL	Carry-less Multiplication
CMAC	Cipher-based Message Authentication Code
CPACF	CP Assist for Cryptographic Function
CSP	Critical Security Parameter
CVL	Component Verification List
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
FSM	Finite State Model
HMAC	Hash Message Authentication Code
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
OFB	Output Feedback
PAA	Processor Algorithm Acceleration
PAI	Processor Algorithm Implementation
PRNG	Pseudo Random Number Generator
RNG	Random Number Generator
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
TDES	Triple DES
TLS	Transport Layer Security
XTS	XEX-based Tweaked-codebook mode with ciphertext Stealing

13 References

[1] FIPS 140-2 Standard,

[2] FIPS 140-2 Implementation Guidance, http://csrc.nist.gov/groups/STM/cmvp/standards.html

[3] FIPS 140-2 Derived Test Requirements, http://csrc.nist.gov/groups/STM/cmvp/standards.html

[4] FIPS 197 Advanced Encryption Standard, http://csrc.nist.gov/publications/PubsFIPS.html

[5] FIPS 180-4 Secure Hash Standard, http://csrc.nist.gov/publications/PubsFIPS.html

[6] FIPS 198-1 The Keyed-Hash Message Authentication Code (HMAC), http://csrc.nist.gov/publications/PubsFIPS.html

[7] FIPS 186-4 Digital Signature Standard (DSS), http://csrc.nist.gov/publications/PubsFIPS.html

[8] NIST SP 800-67 Revision 1, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, http://csrc.nist.gov/publications/PubsFIPS.html

[9] NIST SP 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, http://csrc.nist.gov/publications/PubsFIPS.html

[10] NIST SP 800-38C, Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, http://csrc.nist.gov/publications/PubsFIPS.html

[11] NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, http://csrc.nist.gov/publications/PubsFIPS.html

[12] NIST SP 800-38E, Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices, http://csrc.nist.gov/publications/PubsFIPS.html

[13] NIST SP 800-52, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations,

[14] NIST SP 800-56A, Recommendation for Pair-Wise Key Establishment Schemes using Discrete Logarithm Cryptography (Revised), http://csrc.nist.gov/publications/PubsFIPS.html

[15] NIST SP 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, http://csrc.nist.gov/publications/PubsFIPS.html

[16] RFC 5288, AES Galois Counter mode (GCM) Cipher Suite for TLS, https://tools.ietf.org/html/rfc5288