SUSE Linux Enterprise OpenSSL Cryptographic Module

version 3.1

FIPS 140-2 Non-Proprietary Security Policy

Doc version 3.1.3

Last update: 2021-09-03

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Table of contents

©9/3/21 SUSE, LLC / atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

1 Cryptographic Module Specification

This document is the non-proprietary security policy for the SUSE Linux Enterprise OpenSSL Cryptographic Module version 3.1. 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 OpenSSL Cryptographic Module version 3.1.

1.1 Module Overview

The SUSE Linux Enterprise 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

©9/3/21 SUSE, LLC / atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Table 2 lists the software components of the cryptographic module, which defines its logical boundary.

Component	Description
/lib64/libcrypto.so.1.0.0	Shared library for cryptographic algorithms.
/lib64/libssl.so.1.0.0	Shared library for TLS/DTLS network protocols.
/lib64/.libcrypto.so.1.0.0.hmac	Integrity check HMAC value for the libcrypto shared library.
/lib64/.libssl.so.1.0.0.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.

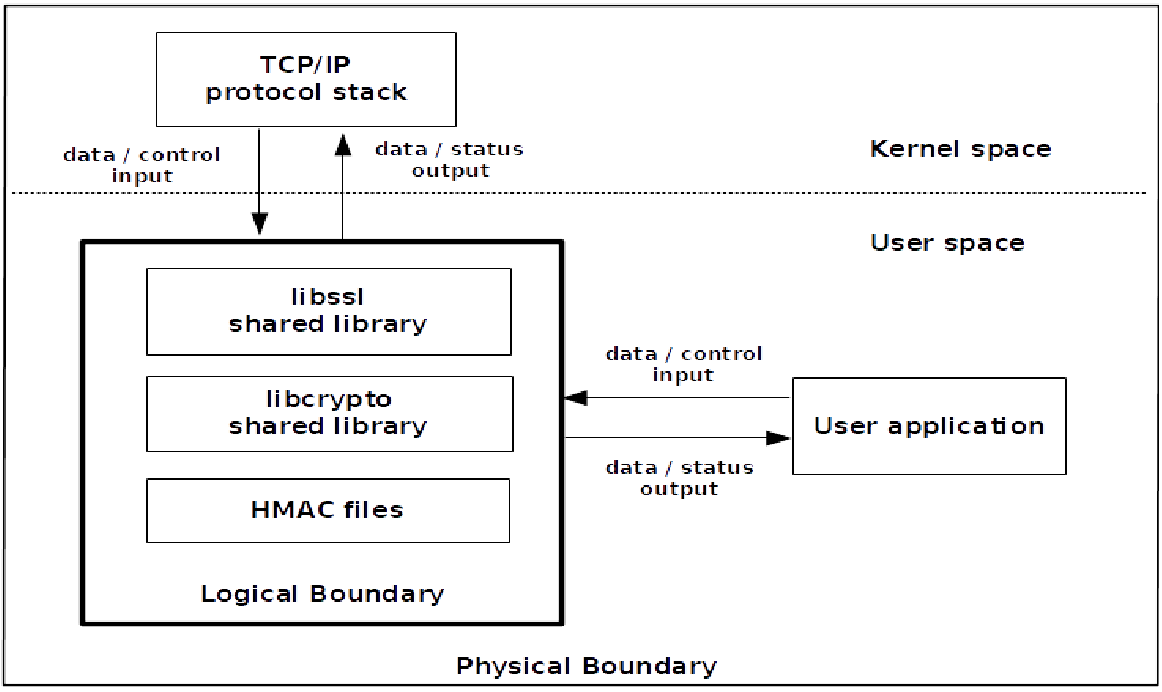

Figure 1: Software Block Diagram

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
FUJITSU Server PRIMERGY RX4770 M5	Intel Cascade Lake Xeon Platinum 8268	SUSE Linux Enterprise Server 12 SP5 with and without AES-NI (PAA)
IBM z13	z13	SUSE Linux Enterprise Server 12 SP5 with and without CPACF (PAI)

Table 3: Tested Platforms

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

©9/3/21 SUSE, LLC / atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

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.

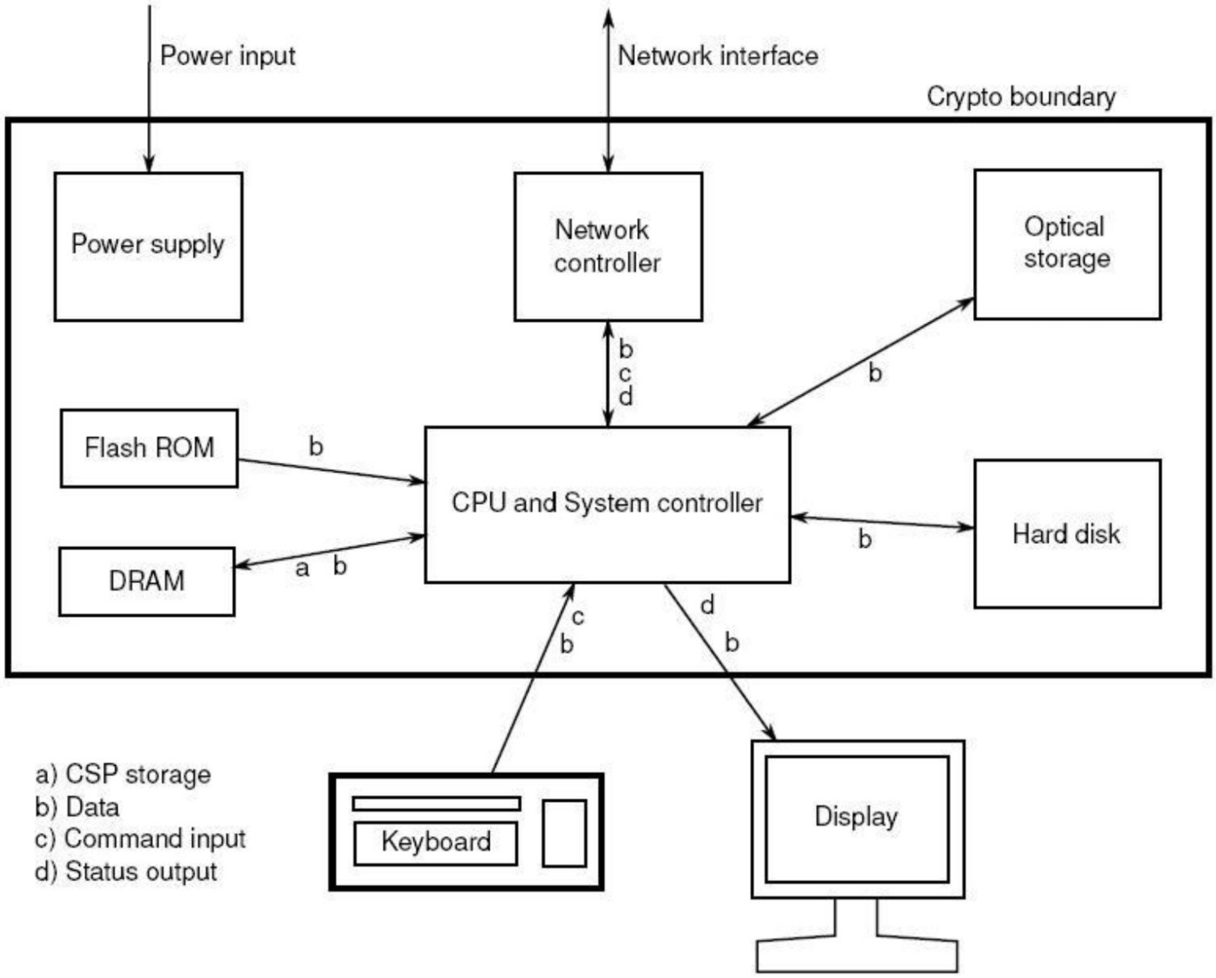

Figure 2: Hardware Block Diagram

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.

©9/3/21 SUSE, LLC / atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

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, error messages.
Power Input	PC Power Supply Port	N/A

Table 4: Ports and Interfaces

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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 and verification	DSA, SHS	User	None	n/a

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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	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	KDF TLS	User	Shared secret	Read
			Derived key	Create
	KDF PBKDF (Vendor Affirmed)	User	Password/passphrase	Read
			Derived key	Create
Network Protocol Services				
Transport Layer Security (TLS) network protocol v1.0, v1.1 and v1.2	Supported cipher suites in FIPS mode (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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service	Algorithm	Role	Keys/CSPs	Access
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, KDF TLS	User	None	N/A
Module installation	N/A	Crypto Officer	None	N/A
Module configuration	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	Blowfish, Camellia, CAST, CAST5, DES, IDEA, RC2, RC4, RC5 and SEED	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
Domain Parameter Generation and Verification	DSA restrictions listed in Table 9	User	None	N/A
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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Service	Algorithm / Modes	Role	Keys	Access
Message digest	GHASH, GOST, MD4, MD5, MDC2, HMAC-MD5, RMD160	User	None	N/A
Message authentication code (MAC)	HMAC and CMAC restrictions listed in Table 9	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 TLS v1.3	User	Shared secret	Read
			Derived key	Create
	KDF PBKDF using non-approved message digest.	User	Password/passphrase	Read
			Derived key	Create
Random Number Generation	ANSI X9.31 RNG	User	None	N/A
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 for the different processor architectures:

- For the Intel Xeon processor architecture:
  - use of AES-NI, SSSE3 and strict assembler instructions for AES implementations;
  - use of AVX2, AVX, SSSE3 and strict assembler instructions for SHA implementations;

This document can be reproduced and distributed only whole and intact, including this copyright notice.

  - use of the CLMUL instruction set and strict assembler for GHASH that is used in GCM mode.
- For the IBM z13 processor architecture:
  - use of the CPACF and strict assembler for AES, SHA and GHASH implementations[1].

The module uses the most efficient implementation based on the processor's capability. Notice that only one algorithm implementation can be executed in runtime.

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.

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, CBC, CFB1, CFB8, CFB128, OFB, CTR	128, 192, 256	Data Encryption and Decryption	FIPS197, SP800-38A	A786 A792 A795
	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	128, 192, 256	Key Wrapping and Unwrapping	SP800-38F	
	GCM	128, 192, 256	Data Encryption and Decryption	SP800-38D	A783 A784 A785 A787 A788 A789 A790 A791 A794
DRBG	CTR_DRBG: AES-128, AES-192, AES-256 with/without DF, with/without PR	N/A	Deterministic Random Bit Generation	SP800-90A	A786 A792 A795 A797
	Hash_DRBG: SHA-1, SHA-224 SHA-256, SHA-384, SHA-512 with/without PR	N/A	Deterministic Random Bit Generation	SP800-90A	A797

1 Only algorithm implementations using CPACF are approved.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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	A797
DSA		L=2048, N=224 L=2048, N=256 L=3072, N=256	Key Pair Generation	FIPS186-4	A779 A780 A781 A782
	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-224	L=2048, N=224	Domain Parameter Verification		
	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	A779 A780 A781 A782
	SHA-224, SHA-256, SHA-384, SHA-512	P-224, P-256, P-384, P-521	Digital Signature Generation		
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	P-224, P-256, P-384, P-521	Digital Signature Verification		
HMAC	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	112 or greater	Message authentication code	FIPS198-1	A779 A780 A781 A782
KAS-ECC-SSC	ECC Ephemeral Unified Scheme	P-224, P-256, P-384, P-521	EC Diffie-Hellman Key Agreement	SP800-56Arev3	A779 A780 A781 A782
KAS-FFC-SSC	dhEphem scheme with safe prime groups.	2048, 3072, 4096, 6144, 8192	Diffie-Hellman Key Agreement	SP800-56Arev3	A807

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
Safe Primes Key Generation and Verification	Safe Prime Groups: ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	2048, 3072, 4096, 6144, 8192	Diffie-Hellman Key Agreement	SP800-56Arev3	A807
KDF PBKDF (vendor affirmed)[2]	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512		Key Derivation	SP800-132	Please see footnote below
KDF TLS (CVL)	TLS v1.0, v1.1, v1.2 SHA-256, SHA-384		Key Derivation	SP800-135Rev1	A779 A780 A781 A782
RSA	B.3.3	2048, 3072, 4096	Key Pair Generation	FIPS186-4	A779 A780 A781 A782
	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			
SHS	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Message Digest	FIPS180-4	A779 A780 A781 A782
Triple-DES	ECB, CBC, CFB1, CFB8, CFB64, OFB	192 (two-key Triple-DES)	Data Decryption	SP800-67	A793

2 Even though the PBKDF has CAVP certificates (#A779, #A780, #A781 and #A782), the corresponding selftests are not implemented.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
		192 (three-key Triple-DES)	Data Encryption and Decryption	SP800-38A	
	CMAC	192	MAC Generation and Verification	SP800-67 SP800-38B	
KTS	AES KW	128, 192, 256	Key Wrapping	SP800-38F	A786 A792 A795
	AES CCM	128, 192, 256			
	AES GCM	128, 192, 256			A783 A784 A785 A787 A788 A789 A790 A791 A794
	AES CBC and HMAC	128, 256			A779 A780 A781 A782 A786 A792 A795
	Triple-DES CBC and HMAC	192			A779 A780 A781 A782 A793

Table 7: Approved Cryptographic Algorithms

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
RSA Key Generation, Digital Signature Generation and Digital Signature Verification with key size > 4096 bits.	Digital Signature; allowed in [SP800-131A]
MD5	Pseudo-random function (PRF) in TLS v1.0 and v1.1; allowed per [SP800-52]
NDRNG	The module obtains the entropy data from a NDRNG to seed the DRBG.

Table 8: Non-Approved but Allowed Algorithms

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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
Blowfish, Camellia, CAST, CAST5, DES, IDEA, RC2, RC4, RC5 and SEED	Data Encryption and Decryption.
2-key Triple-DES	Data Encryption.
GHASH, GOST, MD2, MD4, MD5, MDC-2, HMAC-MD5, RMD160	Message Digest.
HMAC with less than 112-bit keys	Message Authentication Code.
CMAC with 2-key Triple-DES	Message Authentication Code.
SRP	Key Agreement.
SHA-1	Digital Signature Generation for DSA, ECDSA and RSA, 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 and Public Key Validation.
ECDSA with P-192 curve, K curves, B curves and non-NIST curves.	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.
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.
ANSI X9.31 RNG	Random Number Generation

Table 9: Non-Approved Cryptographic Algorithms

This document can be reproduced and distributed only whole and intact, including this copyright notice.

4 Physical Security

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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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.

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.

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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 generated 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_cleanup()
Triple-DES keys			EVP_CIPHER_CTX_free(), EVP_CIPHER_CTX_cleanup()
HMAC keys			HMAC_CTX_cleanup()
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 SP 800-56Arev3 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, 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 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	Derived from entropy	N/A	FIPS_drbg_free()

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Name	Generation	Entry and Output	Zeroization
state: V value, C value, key (if applicable)	input as defined in SP800-90A		
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] (vendor affirmed).

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

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

The public and private keys used in the Diffie-Hellman and EC Diffie-Hellman key agreement schemes are generated internally by the module using the same DSA and ECDSA key generation compliant with [FIPS186-4] and [SP800-56Arev3]. The Diffie-Hellman key agreement scheme generates keys using safe primes defined in RFC7919 and RFC3526, 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 and shared secret computation schemes. The key agreement schemes are also used as part of the TLS protocol key exchange. The module now exclusively supports SP 800-56Arev3 key agreement schemes in FIPS mode of operation. For Diffie-Hellman, the module supports the use of safe primes from RFC7919 for domain parameters and key generation that is used by the TLS key agreement implemented by the module. The module also supports the use of safe primes from RFC3526 that can be used by the IKE key agreement implemented in the OpenSSL module. Note that the current module only implements the shared secret computation of safe primes used in IKE RFC3526 and not the entire IKE key agreement:

- TLS (RFC7919)
  - ffdhe2048 (ID = 256)
  - ffdhe3072 (ID = 257)
  - ffdhe4096 (ID = 258)
  - ffdhe6144 (ID = 259)
  - ffdhe8192 (ID = 260)
- 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, AES-CCM, AES-GCM.
- Key wrapping using 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, CCM, GCM provides between 128 and 256 bits of encryption strength.
- AES key wrapping using AES in CBC mode and HMAC, provides 128 or 256 bits of encryption strength.
- Triple-DES key wrapping using HMAC provides 112 bits of encryption strength.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

- RSA key wrapping[3] provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman shared secret computation provides between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman shared secret computation provides between 112 and 256 bits of encryption strength.
- Diffie-Hellman key agreement provides between 112 and 200 bits of encryption strength.
- EC Diffie-Hellman key agreement provides between 112 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.

The module also supports password-based key derivation (PBKDF), as a vendor-affirmed security function. 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.

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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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.

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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
RSA	KAT RSA PKCS#1 v1.5 scheme with 2048-bit key and SHA-224, SHA-256, SHA-384 and SHA-512 signature generation (separately tested); KAT RSA PKCS#1 v1.5 scheme with 2048-bit key and SHA1, SHA-224, SHA-256, SHA-384 and SHA-512 signature verification (separately tested); KAT RSA PSS scheme with 2048-bit key and SHA224, SHA-256, SHA-384, SHA-512 signature generation (separately tested); KAT RSA PSS scheme with 2048-bit key and SHA1, SHA224, SHA-256, SHA-384, SHA-512 signature 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
TLS KDF (SP 800-135Rev1)	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.

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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_0_0-1.0.2p-3.36.1.x86_64.rpm libopenssl1_0_0-hmac-1.0.2p-3.36.1.x86_64.rpm
IBM z13	libopenssl1_0_0-1.0.2p-3.36.1.s390x.rpm libopenssl1_0_0-hmac-1.0.2p-3.36.1.s390x.rpm

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"
```


©9/3/21 SUSE, LLC / atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

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

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

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

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

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

9.2.2 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

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

- 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 self-tests KATs, with the exception of the RSA and DRBG algorithms, 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.

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.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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.

©9/3/21 SUSE, LLC / atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

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.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Appendix A - TLS Cipher Suites

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

Cipher Suite	Reference
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	RFC2246
TLS_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_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_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_RSA_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	RFC5246
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	RFC5246
TLS_DHE_RSA_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_DHE_DSS_WITH_AES_128_GCM_SHA256	RFC5288
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	RFC5288
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_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

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

Cipher Suite	Reference
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	RFC5289
TLS_ECDHE_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_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	RFC5289
TLS_ECDHE_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_RSA_WITH_AES_128_CCM	RFC6655
TLS_RSA_WITH_AES_256_CCM	RFC6655
TLS_DHE_RSA_WITH_AES_128_CCM	RFC6655
TLS_DHE_RSA_WITH_AES_256_CCM	RFC6655
TLS_RSA_WITH_AES_128_CCM_8	RFC6655
TLS_RSA_WITH_AES_256_CCM_8	RFC6655
TLS_DHE_RSA_WITH_AES_128_CCM_8	RFC6655
TLS_DHE_RSA_WITH_AES_256_CCM_8	RFC6655

Table 16: TLS Cipher Suites

©9/3/21 SUSE, LLC / atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.

Appendix B - CAVP certificates

The tables below show the certificates obtained from the CAVP for all the target platforms included in Table 3. The CAVP certificates validate all algorithm implementations used as approved or allowed security functions in FIPS mode of operation. The tables include the certificate number, the label used in the CAVP certificate for reference and a description of the algorithm implementation.

Cert#	CAVP Label	Algorithm Implementation
A779	SHA_ASM	All algorithms impacted by SHA using assembler implementation.
A780	SHA_SSSE3	All algorithms using SHA with SSSE3 instruction.
A781	SHA_AVX	All algorithms using SHA with AVX instruction.
A782	SHA_AVX2	All algorithms using SHA with AVX2 instruction.
A783	BAES_CTASM_ASM	AES-GCM using SSSE3 instruction for Constant Time assembler and Bit Slice, and assembler implementation for multiplication and GHASH.
A784	BAES_CTASM_CLMULNI	AES-GCM using SSSE3 instruction for Constant Time assembler and Bit Slice, and PCLMULQDQ instruction for multiplication and GHASH.
A785	BAES_CTASM_AVX	AES-GCM using SSSE3 instruction for Constant Time assembler and Bit Slice AES, and AVX instruction for multiplication and GHASH.
A786	BAES_CTASM	AES using SSSE3 instruction for Constant Time assembler and Bit Slice AES.
A787	AESASM_CLMULNI	AES-GCM using assembler implementation, and PCLMULQDQ instruction for multiplication and GHASH.
A788	AESASM_AVX	AES-GCM using assembler implementation, and AVX instruction for multiplication and GHASH.
A789	AESNI_ASM	AES-GCM using AESNI, and assembler implementation for multiplication and GHASH.
A790	AESNI_CLMULNI	AES-GCM using AESNI instructions, and PCLMULQDQ instruction for multiplication and GHASH.
A791	AESNI_AVX	AES-GCM using AESNI instructions, and AVX instruction for multiplication and GHASH.
A792	AESNI	AES using AESNI instructions.
A793	TDES_C	Triple-DES C implementation.
A794	AESASM_ASM	AES-GCM using assembler implementation.
A795	AESASM	AES assembler implementation.
A797	DRBG_10X	Generic DRBG implementation with all types of DRBG.
A807	FFC_DH	Generic C non-optimized DH implementation.

Table 17: Implementations for CAVP certificates

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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** August 12, 2020 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
FIPS202	**SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions** https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.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
RFC3526	**More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)** https://tools.ietf.org/html/rfc3526
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

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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

RFC7919 **Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)**
https://tools.ietf.org/html/rfc7919

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

©9/3/21 SUSE, LLC / atsec information security.

This document can be reproduced and distributed only whole and intact, including this copyright notice.

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

©9/3/21 SUSE, LLC / atsec information security.
This document can be reproduced and distributed only whole and intact, including this copyright notice.