SUSE Linux Enterprise Server 12 - OpenSSL Module v2.0

FIPS 140-2 Non-Proprietary Security Policy

Version 1.0

Last Update: 2015-07-01

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 12 - 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 12 - OpenSSL Module (hereafter referred to as the “Module”) is a software library supporting FIPS 140-2 Approved cryptographic algorithms. The current version of the Module is 2.0. An earlier version of this Module has gone through FIPS 140-2 validation under certificate #1930.

This Module provides a C language application program interface (API) for use by other processes that require cryptographic functionality.

For FIPS 140-2 purposes, the Module is classified as a multi-chip standalone module. The Module's logical cryptographic boundary is the shared library files and their integrity check HMAC files that are delivered with the RPM packages. The dracut-fips RPM package is only used for the configuration of the Module in every boot. This code is not active when the Module is operational and does not provide any services to users interacting with the Module. Configuration and installation of the Module requires following RPM packages:

- The configuration of the FIPS Approved mode is provided by the dracut-fips package delivered in dracut-fips-037-37.2.x86_64.rpm.
- 64-bit libssl and libcrypto shared libraries (libssl.so.1.0.0 and libcrypto.so.1.0.0) delivered in libopenssl1_0_0-1.0.1i-17.1.x86_64.rpm – note that the RPM also delivers other shared libraries implementing the OpenSSL engines which are not part of the Module and are inaccessible in FIPS Approved mode.
- 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-1.0.1i-17.1.x86_64.rpm.

The Module's physical cryptographic boundary is the enclosure of the computer system on which it is executing.

Security Component	FIPS 140-2 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 has been tested in the following software configurations:

- 64-bit x86_64 with AES-NI enabled
- 64-bit x86_64 with AES-NI disabled

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

Manufacturer	Model	O/S & Ver.
HP	ProLiant DL320e Gen8	SUSE Linux Enterprise Server 12

Table 2: Tested Platforms

1.2 Description of the FIPS Approved Mode

When the Module is powered-on, the FIPS_mode_set() function is executed automatically without any operator intervention. The Module verifies the integrity of the runtime executable using a HMAC-SHA-256 digest computed at build time. If the digests match, the power-up self-test is then performed. If the power-up self-test is successful, FIPS_mode_set() sets the FIPS_mode flag to 1 and the Module is in FIPS Approved mode.

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

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

Algorithm	Validation Certificate	Standards/Usage	Keys/CSPs
AES using AES-NI processor instructions (ECB, CBC, OFB, CFB 1, CFB 8, CFB 128, CTR, CCM, GCM and XTS)	Cert. #3197	FIPS 197 AES SP 800-38A SP 800-38C CCM SP 800-38D GCM SP 800-38E XTS Encryption and Decryption	AES keys 128 bits, 192 bits (except XTS) and 256 bits
AES using SSSE3 assembler (ECB, CBC, OFB, CFB 1, CFB 8, CFB 128, CTR, CCM, GCM and XTS)	Cert. #3199		
AES using straight assembler (ECB, CBC, OFB, CFB 1, CFB 8, CFB 128, CTR, CCM, GCM and XTS)	Cert. #3198		
Triple-DES (ECB, CBC, CFB 1, CFB 8, CFB 64, OFB, CTR)	Cert. #1823	SP 800-67 SP 800-38A Encryption and Decryption	Triple-DES keys 168 bits
DSA	Cert. #915	FIPS 186-4	DSA keys 2048 and 3072 bits

Algorithm	Validation Certificate	Standards/Usage	Keys/CSPs
• SHA-224 • SHA-256 • SHA-384 • SHA-512		Domain Parameter Generation, Domain Parameter Verification, Key Pair Generation and Signature Generation	
DSA • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512		Signature Verification	DSA keys 1024, 2048 and 3072 bits Note: 1024 bits DSA key is legacy support only.
RSA Key Pair Generation	Cert. #1628	FIPS 186-4 Appendix B.3.3	RSA keys 2048 and 3072 bits
RSA (X9.31) Signature Generation • SHA-256 • SHA-384 • SHA-512		FIPS 186-4	RSA keys 2048 and 3072 bits
RSA (X9.31) Signature Verification • SHA-1 • SHA-256 • SHA-384 • SHA-512			RSA keys 1024, 2048 and 3072 bits
RSA (PKCS #1 v1.5 and PSS) Signature Generation • SHA-224 • SHA-256 • SHA-384 • SHA-512			RSA keys 2048 and 3072 bits
RSA (PKCS #1 v1.5 and PSS) Signature Verification • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512			RSA keys 1024, 2048 and 3072 bits
ECDSA	Cert. #586	FIPS 186-4 Key Pair Generation and	ECDSA keys P-256, P-384 and P-521

Algorithm	Validation Certificate	Standards/Usage	Keys/CSPs
		Public Key Verification	
ECDSA • SHA-224 • SHA-256 • SHA-384 • SHA-512		Signature Generation	
ECDSA • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512		Signature Verification	
SHA-1 (using AVX + SSSE3 assembler)	Cert. #2645	FIPS 180-4 Hashing	N/A
SHA-1 (using SSSE3 assembler)	Cert. #2648		
SHA-1 SHA-224 SHA-256 SHA-384 SHA-512 (using straight assembler)	Cert. #2646		
HMAC-SHA-1 (using AVX + SSSE3 assembler)	Cert. #2014	FIPS 198-1 Message Integrity	At least 112 bits HMAC Key
HMAC-SHA-1 (using SSSE3 assembler)	Cert. #2016		
HMAC-SHA-1 HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512 (using straight assembler)	Cert. #2015		
Hash-based DRBG	Certs. #674, #675, #676	SP 800-90A Random Number Generation	Entropy input string, seed, V and C
HMAC-based DRBG			Entropy input string, seed, V and Key
CTR-based DRBG			Entropy input string, seed, V and Key
AES CMAC	Certs. #3197, #3198, #3199	SP 800-38B Message Integrity Generation	AES keys 128, 192 and 256 bits
Triple-DES CMAC	Cert. #1823		Triple-DES keys 168 bits

Algorithm	Validation Certificate	Standards/Usage	Keys/CSPs
TLS v1.0, v1.1 and v1.2	CVL Cert. #430	Key Deviation Function	N/A
Diffie-Hellman • SHA-224 • SHA-256	CVL Cert. #431	SP 800-56A Key agreement	Ephemeral keys 2048 and 3072 bits, and Shared secret
EC Diffie-Hellman • SHA-256 • SHA-384 • SHA-512	CVL Cert. #431	SP 800-56A Key agreement	Ephemeral keys P-256, P-384 and P-521, and Shared secret

Table 3: Approved Algorithms

The Module provides multiple implementations of AES and SHA. Different implementations can be set by an environment variable. The implementation is selected by the Module based on properties of the underlying hardware. Thus, only one implementation will ever be available at runtime. All these implementations and the related algorithms have been CAVS tested.

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

Algorithm	Validation Certificate	Standards/Usage	Keys/CSPs
RSA (encrypt, decrypt)	N/A	Key wrapping	RSA keys 2048 and 3072 bits

Table 4: Non-Approved but Allowed Algorithms

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

Algorithm	Standards/Usage
Diffie-Hellman	Key agreement, domain parameters with size smaller than 2048 bits
EC Diffie-Hellman	Key agreement, domain parameters with size smaller than P-256
RSA (encrypt, decrypt)	Key wrapping, key size smaller than 2048 bits or greater than 3072 bits
ANSI X9.31 RNG	Random Number Generation
MD2	Message digest
MD4	Message digest
MD5	Message digest
MDC-2	Message digest
HMAC-MD5	Message digest
DSA	Domain Parameters Generation, Key Generation and Signature Generation, key size smaller than 2048 bits or greater than 3072 bits
RSA	Key Generation and Signature Generation, key size smaller than 2048 bits or greater than 3072 bits
RSA	Signature Verification, key size smaller than 1024 bits and greater than 3072 bits

Algorithm	Standards/Usage
Blowfish	Encryption and Decryption
Camellia	Encryption and Decryption
CAST	Encryption and Decryption
DES	Encryption and Decryption
IDEA	Encryption and Decryption
JPAKE	Key Agreement
RC2	Encryption and Decryption
RC4	Encryption and Decryption
RC5	Encryption and Decryption
RIPEMD160	Message digest
SEED	Encryption and Decryption
TLS-SRP	Key exchange
Whirlpool	Hash

Table 5: 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.

Notes:

1) Diffie-Hellman (CVL Cert. #431, key agreement; key establishment methodology provides 112 or 128 bits of encryption strength)
2) EC Diffie-Hellman (CVL Cert. #431, key agreement; key establishment methodology provides between 128 and 256 bits of encryption strength)
3) RSA (key wrapping; key establishment methodology provides 112 or 128 bits of encryption strength)
4) MD5 and HMAC-MD5 for use in TLS only

Caveat:

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

1.3 Cryptographic Boundary

The Module's physical boundary is the surface of the case of the platform (depicted in the hardware block diagram). The Module's logical boundary is depicted in the software block diagram.

1.3.1 Hardware Block Diagram

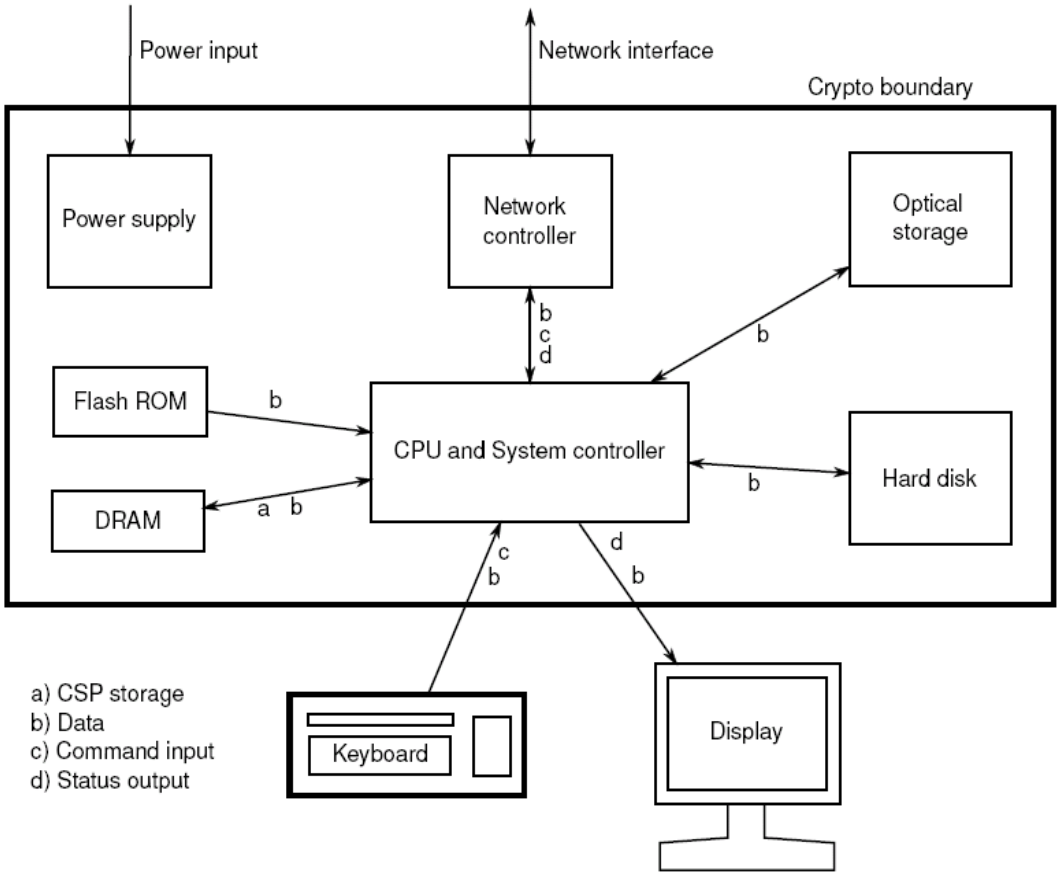

Figure 1. Hardware Block Diagram

1.3.2 Software Block Diagram

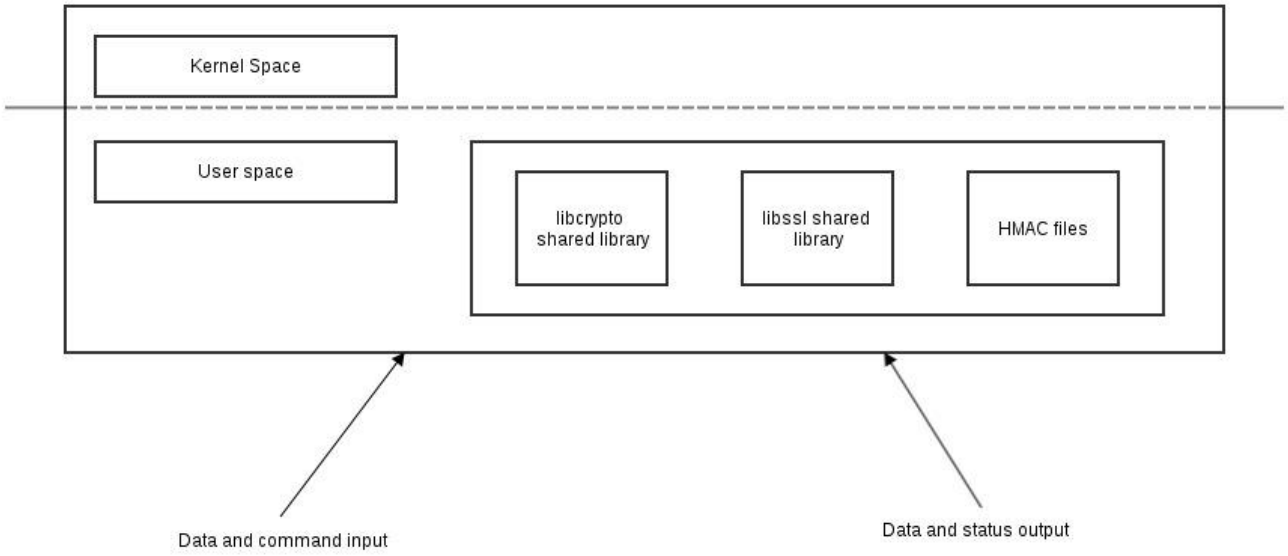

Figure 2. Software Block Diagram

1.4 SUSE Linux Cryptographic Modules and FIPS 140-2 Validation

1.4.1 FIPS Approved Mode

The FIPS Approved mode ensures that FIPS required self tests are executed and that ciphers are restricted to those that have been FIPS validated by the CMVP.

The FIPS Approved mode for a Module becomes effective as soon as the Module power on self tests complete successfully and the Module loads into memory.

2 Cryptographic Module Ports and Interfaces

The physical ports of the Module are the same as the computer system on which it is executing. The logical interface is a C-language application program interface (API).

The Data Input interface consists of the input parameters of the API functions. The Data Output interface consists of the output parameters of the API functions. The Control Input interface consists of the actual API functions. The Status Output interface includes the return values of the API functions. The ports and interfaces are shown in the following table.

FIPS Interface	Physical Port	Module Interface
Data Input	Ethernet ports	API input parameters, kernel I/O – network or files on filesystem
Data Output	Ethernet ports	API output parameters, kernel I/O – network or files on filesystem
Control Input	Keyboard, Serial port, Ethernet port	API function calls, or /proc/sys/crypto/fips_enabled
Status Output	Serial port, Ethernet port	API return codes and status parameters
Power Input	PC Power Supply Port	N/A

Table 6: 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 7 (and the services are further detailed in Table 8 and Table 9).

Role	Descriptions
User	Perform general security services, including Approved and non-Approved security functions. (Please see Table 8 and Table 9 for more details)
Crypto Officer	Perform Module installation, configuration and initialization

Table 7: 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 for the crypto operations are crypto(3) 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 3 and Table 4 for the Approved key size of each algorithm used in the services.

Service	Role	CSPs	Access
Symmetric encryption/decryption	User	AES or Triple-DES key	read/write/execute
Asymmetric key generation	User	RSA, DSA or ECDSA private key	read/write/execute
Digital signature generation/verification	User	RSA, DSA or ECDSA private key	read/write/execute
Random number generation (SP 800-90A DRBG)	User	Seed, entropy input string and intermediate values	read/write/execute
RSA key wrapping	User	RSA private key	read/write/execute
TLS network protocol	User	AES or Triple-DES key, HMAC key	read/write/execute
TLS Key Agreement	User	AES or Triple-DES key, RSA, DSA or ECDSA private key, HMAC key, Pre-Master Secret, Master Secret, Diffie-Hellman or EC Diffie-Hellman private components	read/write/execute
Certificate Management/ Handling	User	RSA, DSA or ECDSA private key of certificates	read/write/execute
Keyed Hash (HMAC)	User	HMAC key	read/write/execute

Service	Role	CSPs	Access
Keyed Hash (CMAC)	User	CMAC key	read/write/execute
Message digest (SHS)	User	none	read/write/execute
Show status	User	none	execute
Module installation	Crypto Officer	none	read/write
Module initialization	Crypto Officer	none	execute
Self test	User	HMAC-SHA-256 key for integrity test	read/execute
Zeroize	User	All aforementioned CSPs	read/write/execute

Table 8: Approved Service Details

The following table lists the non-Approved services available in non-Approved mode. Please refer to Table 5 for the non-Approved key size or algorithm.

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

Table 9: Non-Approved Service Details

3.3 Operator Authentication

At security level 1, authentication is neither required nor employed. The role is implicitly assumed on entry.

3.4 Mechanism and Strength of Authentication

At security level 1, authentication is not required.

4 Physical Security

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

5 Operational Environment

This Module operates in a modifiable operational environment per the FIPS 140-2 definition.

5.1 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.

In FIPS Approved mode, the ptrace(2) system call, the debugger (gdb(1)), and strace(1) shall be not 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	Storage	Zeroization API
AES or Triple-DES key	SP 800-90A DRBG	Ephemeral	EVP_CIPHER_CTX_cleanup()
AES GCM IV	SP 800-90A DRBG	Ephemeral	aes_gcm_cleanup()
DSA, ECDSA or RSA private and public keys	SP 800-90A DRBG	Ephemeral	DSA_free(), EC_KEY_free() or RSA_free()
HMAC keys	SP 800-90A DRBG or established during the TLS handshake	Ephemeral	HMAC_CTX_cleanup()
CMAC keys	SP 800-90A DRBG	Ephemeral	CMAC_CTX_cleanup()
Diffie-Hellman or EC Diffie-Hellman private and public components	SP 800-90A DRBG	Ephemeral	DH_free() or EC_KEY_free()
DRBG seed and entropy string	Obtained from /dev/urandom	Ephemeral	Automatic zeroized when seeding operation completes
DRBG intermediate values	Derived from the entropy string as defined in SP 800-90A	Ephemeral	Automatic zeroized when freeing DRBG handle
TLS Pre-Master Secret and Master Secret	Established during the TLS handshake	Ephemeral	SSL_free() and SSL_clear()

Table 10: 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.

The Linux kernel provides /dev/urandom as a source of random numbers for DRBG seeds. The Linux kernel initializes this pseudo device at system startup.

The Module performs Continuous Random Number Generation Test (CRNGT) on the output of the SP 800-90A DRBG to ensure that consecutive random numbers do not repeat. The CRNGT on the random numbers for seeding the DRBG is performed by the kernel.

6.2 Key/Critical Security Parameter (CSP) Authorized Access and Use by Role and Service/Function

An authorized application as user (the User role) has access to all key data generated during the operation of the Module.

6.3 Key/CSP Storage

Public and private keys are provided to the Module by the calling process, and are destroyed when released by the appropriate API function calls. The Module does not perform persistent storage of keys.

6.4 Key/CSP Zeroization

The memory occupied by keys is allocated by regular libc malloc/calloc() calls. The application is responsible for calling the appropriate destruction functions from the OpenSSL Module API. The destruction functions then overwrite the memory occupied by keys with “zeros” and deallocates the memory with the free() call. 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 platform that runs the Module meets the requirements of 47 CFR FCC PART 15, Subpart B, Class A (Business use).

8 Self Tests

FIPS 140-2 requires that the Module performs self-tests to ensure the integrity of the Module and the correctness of the cryptographic functionality at start up. In addition, some functions require continuous verification of function, such as the random number generator. These tests consist of Known Answer Test (KAT), Pairwise Consistency Test (PCT) and Continuous Random Number Generation Test (CRNGT) as listed and described in this section.

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

See section 9.2 for descriptions of possible self-test errors and recovery procedures.

8.1 Power-Up Tests

The Module performs both power-up self-tests (at Module initialization) and continuous condition tests (during operation). Input, output, and cryptographic functions cannot be performed while the Module is in a self-test or error state because the Module is single-threaded and will not return to the calling application until the power-up self-tests are complete. If the power-up self-tests fail, subsequent calls to the Module will also fail - thus no further cryptographic operations are possible.

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 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
SP800-90A CTR DRBG	KAT
SP800-90A hash-based DRBG	KAT
SP800-90A HMAC-based 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

Algorithm	Test
HMAC-SHA-384	KAT
HMAC-SHA-512	KAT
CMAC with AES and Triple-DES	KAT, encryption and decryption are tested separately
HMAC-SHA-256	Module integrity

Table 11: Module Self Tests

8.2 Conditional Tests

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
ANS X9.31 RNG	CRNGT

Table 12: Module Conditional Tests

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 RPM package of the Module can be installed by standard tools recommended for the installation of RPM packages on a SUSE Linux system (for example, rpm, yast and yast online_update).

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 while in the FIPS Approved mode. Furthermore, once the FIPS Approved mode is entered, it must not be exited, which prohibits calls to FIPS_mode_set(0).

To bring the Module into FIPS approved mode, perform the following:

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.

Because FIPS 140-2 has certain restrictions on the use of cryptography which are not always wanted, the Module needs to be put into FIPS approved mode explicitly. If the file /proc/sys/crypto/fips_enabled exists and

contains a numeric value other than 0, the Module is put into FIPS approved mode at initialization time. This is the mechanism recommended for ordinary use, activated by using the fips=1 option in the boot loader, as described above.

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 entry into FIPS approved mode. Failure to do so will not allow the application to properly enter FIPS approved mode.

Once the Module has been put into FIPS Approved mode, it is not possible to switch back to standard mode without terminating the process first.

The version of the RPM containing the validated Module is listed in Section 1.1. The integrity of the RPM is automatically verified during the installation of the Module and the Crypto Officer shall not install the RPM file if the RPM tool indicates an integrity error.

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.

Explicitly invoke the Module in the FIPS Approved mode by calling the FIPS_mode_set(1) function, which returns a “1” for success or a “0” for failure.

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

Interpretation of the return code is the responsibility of the host application. Prior to invocation, the Module is uninitialized in non-Approved mode by default.

The Module performs the self tests described in section 8.1. See section 8 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 any more.

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: in case the Module is used as TLS server, the Diffie-Hellman parameters (dh argument) of the aforementioned API call must be 2048 bits or larger; in case the Module is used as TLS client, the TLS server must be configured to only offer Diffie-Hellman keys of 2048 bits or larger.

9.2.2 AES XTS Guidance

The length of the data unit encrypted with XTS-AES shall not exceed 2^{20} AES blocks that is 16MB of data.

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 Guidance

The AES_GCM is used within the TLS protocol of version 1.2 or higher. The module is compliance with SP 800-52 and the mechanism for IV generation is compliance 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.

9.2.5 RSA and DSA Keys

The cryptographic Module of OpenSSL 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 either 2048 bit keys or 3072 bit keys. To comply with the requirements of FIPS 140-2, a user must therefore only use keys with 2048 bits or 3072 bits.

9.3 Handling Self Test Errors

The effects of self-test failures in the Module differ depending on the type of self-test that failed.

The FIPS_mode_set() function verifies the integrity of the runtime executable using a HMAC SHA-256 digest, which is computed at build time. If this computed HMAC SHA-256 digest matches the stored known digest, then the algorithm-specific power-up self-tests are performed, which consist of of Known Answer Test (KAT), Pairwise Consistency Test (PCT) and Continuous Random Number Generation Test (CRNGT).

Non-fatal self-test errors transition the Module into an error state. The application must be restarted to recover from these errors. The non-fatal self-test errors are:

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
FIPS_R_FIPS_MODE_ALREADY_SET - the application initializes the FIPS Approved mode when it is already initialized

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.

A fatal error occurs when the Module is in the error state (a self-test has failed) and the application calls a crypto function of the Module that cannot return an error in normal circumstances (void return functions). The error message 'FATAL FIPS SELFTEST FAILURE' is printed to stderr and the application is terminated with the abort() call. The only way to recover from a fatal error is to restart the application. If failures persist, you must reinstall the Module. If you downloaded the software, verify the package hash to confirm a proper download.

10 Mitigation of Other 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 API function of RSA_blinding_on turns blinding on for key rsa and generates a random blinding factor. The random number generator must be seeded prior to calling RSA_blinding_on.

Weak Triple-DES keys are detected as follows:

```
/* 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 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
CMT	Cryptographic Module Testing
CMVP	Cryptographic Module Validation Program
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
NVLAP	National Voluntary Laboratory Accreditation Program
OFB	Output Feedback
O/S	Operating System
PRNG	Pseudo Random Number Generator
RNG	Random Number Generator
RSA	Rivest, Shamir, Addleman
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SLA	Service Level Agreement
SOF	Strength of Function
SSH	Secure Shell
TDES	Triple DES
UI	User Interface

12 References

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

[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, http://csrc.nist.gov/publications/PubsFIPS.html

[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