Red Hat Enterprise Linux 6.6 OpenSSL Module v3.0

FIPS 140-2 Non-proprietary Security Policy

version 2.15

Last Update: 2015-08-28

Contents

1.Cryptographic Module Specification 3
1.1.Description of the Module 3
1.2.Description of the Approved Modes 4
1.3.Cryptographic Boundary....7
1.3.1.Hardware Block Diagram....8
1.3.2.Software Block Diagram....9
2.Cryptographic Module Ports and Interfaces 10
3.Roles, Services and Authentication....11
3.1.Roles....11
3.2.Services....11
3.3.Operator Authentication....12
3.4.Mechanism and Strength of Authentication....12
4.Physical Security 13
5.Operational Environment 14
5.1.Policy 14
6.Cryptographic Key Management....15
6.1.Random Number Generation....15
6.2.Key/Critical Security Parameter (CSP) Authorized Access and Use by Role and Service/Function 15
6.3.Key/CSP Storage 15
6.4.Key/CSP Zeroization....15
7.Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC) 16
8.Self-Tests 17
8.1.Power-Up Tests....17
8.2.Conditional Tests....17
9.Guidance....19
9.1.Crypto Officer Guidance 19
9.2.User Guidance....20
9.2.1.TLS and Diffie-Hellman....20
9.2.2.AES XTS Guidance....20
9.2.3.Random Number Generator....21
9.2.4.AES GCM IV....21
9.2.5.RSA and DSA Keys....21
9.3.Handling Self-Test Errors 21
10.Mitigation of Other Attacks....22
11.Glossary and Abbreviations....23
12.References....24
Appendix A: Code for Invoking the OpenSSL Module....25

1. Cryptographic Module Specification

This document is the non-proprietary security policy for the Red Hat Enterprise Linux 6.6 OpenSSL Module v3.0, 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 Red Hat Enterprise Linux 6.6 OpenSSL Module (hereafter referred to as the “Module”) is a software library supporting FIPS 140-2 Approved cryptographic algorithms. The code base of the Module is formed in a combination of standard OpenSSL shared library, OpenSSL FIPS Object Module and development work by Red Hat. The Module provides a C language application program interface (API) for use by other processes that require cryptographic functionality. For the purpose of the FIPS 140-2 validation, its embodiment type is defined as multi-chip standalone.

The following table shows the security level for each of the eleven sections of the validation.

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 on the following multi-chip standalone platforms:

Manufacturer	Model	O/S & Ver.
HP (Hewlett-Packard)	ProLiant DL380p Gen8	Red Hat Enterprise Linux 6.6 (single operator)
IBM (International Business Machines)	System x3500 M4	Red Hat Enterprise Linux 6.6 (single operator)

Table 2: Tested Platforms

On each of the tested platforms listed above, the Module has been tested for the following configurations:

- 32-bit x86_64 with and without AES-NI enabled
- 64-bit x86_64 with and without AES-NI enabled

To operate the Module, the operating system must be restricted to a single operator mode of operation. (This should not be confused with single user mode which is runlevel 1 on Red Hat Enterprise Linux (RHEL). This refers to processes having access to the same cryptographic instance which RHEL ensures this cannot happen by the memory management hardware.)

1.2. Description of the Approved Modes

If the file '/proc/sys/crypto/fips_enabled' exists and contains a numeric of '1', the Module is invoked into FIPS Approved mode at initialization time.

The Module verifies the integrity of the runtime executable using a HMAC-SHA-256 digest computed at build time. If the digests matched, the power-up self-test is then performed. If the power-up self-test is successful, the FIPS_mode flag is set to TRUE and the Module is in FIPS Approved mode.

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)	Certs. #3106, #3113, #3114, #3119	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-AES) and 256 bits
AES using SSSE3 assembler (ECB, CBC, OFB, CFB 1, CFB 8, CFB 128, CTR, CCM, GCM and XTS)	Certs. #3107, #3108, #3109, #3112		AES keys 128 bits, 192 bits and 256 bits Notes: XTS-AES only supports 128 bits and 256 bits AES key. AES with ECB mode only supports 128 bits AES key.
AES using straight assembler (ECB, CBC, OFB, CFB 1, CFB 8, CFB 128, CTR, CCM, GCM and XTS)	Certs. #3104, #3105, #3110, #3111		
Triple-DES (ECB, CBC, OFB, CFB 1, CFB 8, CFB 64, CTR)	Certs. #1784, #1785, #1786, #1790	SP 800-67 Encryption and Decryption	Triple-DES keys 168 bits
AES CMAC	Certs. #3104, #3105, #3106, #3107, #3108, #3109, #3110, #3111, #3112, #3113, #3114, #3119	SP 800-38B Message Integrity Generation	AES keys 128, 192, 256 bits
Triple-DES CMAC	Certs. #1784, #1785, #1786, #1790		Triple-DES keys 168 bits
DSA • SHA-224 • SHA-256	Certs. #897, #898, #899, #903	FIPS 186-4 Domain Parameters Generation and Verification, Key	DSA keys L=2048, N=224 L=2048, N=256 L=3072, N=256

Algorithm	Validation Certificate	Standards/Usage	Keys/CSPs
		Generation, Signature Generation	
DSA Signature Verification • SHA-1 • SHA-224 • SHA-256		FIPS 186-4 Signature Verification	DSA keys L=1024, N=160 L=2048, N=224 L=2048, N=256 L=3072, N=256 Note: 1024 bits DSA key is legacy support only.
RSA	Certs. #1583, #1584, #1586, #1590	FIPS 186-4 Appendix B.3.3 Key Generation	RSA keys 2048, 3072 bits
RSA (X9.31) Signature Generation • SHA-256 • SHA-384 • SHA-512		FIPS 186-4 Signature Generation and Verification	RSA keys 2048, 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, 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	Certs. #560, #561, #562, #564	FIPS 186-4 Key Pair Generation and Public Key Verification	ECDSA keys based on P-256, P-384, or P-521 curve

Algorithm	Validation Certificate	Standards/Usage	Keys/CSPs
ECDSA Signature Generation • SHA-224 • SHA-256 • SHA-384 • SHA-512		FIPS 186-4 Signature Generation	
ECDSA Signature Verification • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512		FIPS 186-4 Signature Verification	
CTR-based DRBG • AES 128 • AES 192 • AES 256	Certs. #612, #613, #614, #615, #616, #621, #622, #623, #624, #625, #626, #631	SP 800-90A Random Number Generation	Entropy input string, seed, V and Key
Hash-based DRBG • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512	Certs. #610, #611, #614, #617, #618, #619, #620, #625, #626, #629, #630, #631		Entropy input string, seed, V and C
HMAC-based DRBG • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512	Certs. #610, #611, #614, #617, #618, #619, #620, #625, #626, #629, #630, #631		Entropy input string, seed, V and Key
SHA-1 (using AVX + SSSE3 assembler)	Certs. #2547, #2569, #2570, #2577	FIPS 180-4 Hashing	N/A
SHA-1 (using SSSE3 assembler)	Certs. #2563, #2564, #2565, #2566		
SHA-1 SHA-224 SHA-256 SHA-384 SHA-512 (using straight assembler)	Certs. #2567, #2568, #2574, #2575		
HMAC-SHA-1 (using AVX + SSSE3 assembler)	Certs. #1931, #1950, #1951, #1958	FIPS 198-1 Message Integrity	At least 112 bits HMAC Key
HMAC-SHA-1 (using SSSE3 assembler)	Certs. #1944, #1945, #1946, #1947		

Algorithm	Validation Certificate	Standards/Usage	Keys/CSPs
HMAC-SHA-1 HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512 (using straight assembler)	Certs. #1948, #1949, #1955, #1956		
SP 800-56A DLC primitive Diffie-Hellman	CVL Certs. #374, #376, #377, #380	SP 800-56A Key Agreement and Establishment	public key size 2048 bits or larger and private key size 224 bits or 256 bits
SP 800-56A DLC primitive EC Diffie-Hellman			P-256, P-384, P-521 curve
SP 800-135 Section 4.2 Key Derivation in TLS v1.0, v1.1 and v1.2	CVL Certs. #375, #381	SP800-135 Key Derivation in TLS	None

Table 3: Approved Algorithms

The Module supports AES-NI, straight AES assembler and AES SSSE3 assembler implementation for AES. Different implementations can be set by the environment variable. The Module also supports SHA AVX+SSSE3 assembler, SHA SSSE3 assembler and straight SHA assembler implementation for SHA. All these implementations and the related algorithms have been CAVS tested. Although the Module implements different AES and SHA implementations, only one will ever be available at runtime.

The Module supports the following non-Approved algorithms but allowed in FIPS Approved mode:

Algorithm	Usage	Keys/CSPs
RSA (encrypt, decrypt) with key size equal or larger than 2048 bits	Key Wrapping	RSA private keys
Diffie-Hellman with public key size 2048 bits or larger and private key size 224 bits or 256 bits	Key Agreement	Diffie-Hellman private keys
EC Diffie-Hellman with key sizes according to P-256, P-384 and P-521 NIST curves	Key Agreement	EC Diffie-Hellman private keys
MD5	Message Digest used only in TLS	N/A

Table 4. Non-Approved but allowed Algorithms

According to Table 2: Comparable strengths in NISP SP 800-57 Part1 (dated on March 8, 2007), the key sizes of RSA, Diffie-Hellman and EC Diffie-Hellman provides the following security strength for

the corresponding key establishment method shown below:

1. *RSA (key wrapping; key establishment methodology provides 112 or 128 bits of encryption strength; non-compliant less than 112 bits of encryption strength)*
2. *Diffie-Hellman (key agreement; key establishment methodology provides 112 or 128 bits of encryption strength; non-compliant less than 112 bits of encryption strength)*
3. *EC Diffie-Hellman (key agreement; key establishment methodology provides between 112 and 256 bits of encryption strength)*

However, the size alone does not determine the security strength of the RSA, Diffie-Hellman and EC Diffie-Hellman keys. Since the seed source for key generation is outside the logical boundary of the module, the following caveat is applicable:

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

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

The Module supports the following non-FIPS 140-2 Approved algorithms, which shall not be used in the FIPS Approved mode. Any use of the non-Approved functions will cause the Module to operate in the non-FIPS mode implicitly:

Algorithm	**Usage**	**Keys/CSPs**
RSA (encrypt, decrypt) with key size smaller than 2048 bits	key wrapping	RSA keys
RSA with key sizes not listed in Table 3	sign, verify, and key generation	RSA keys
DSA with key sizes not listed in Table 3	sign, verify, and key generation	DSA keys
Diffie-Hellman with key sizes not listed in Table 4	key agreement and establishment	Diffie-Hellman keys
ANSI X9.31 RNG (with AES-128 core)	random number generation	PRNG seed value and seed key 128 bits
Camellia	Encryption/decryption	Symmetric key
CAST	Encryption/decryption	Symmetric key
DES	Encryption/decryption	Symmetric key
IDEA	Encryption/decryption	Symmetric key
J-PAKE	Password Authenticated Key Exchange	Password
MD2	Hash function	N/A
MD4	Hash function	N/A
MDC2	Hash function	N/A
RC2	Encryption/decryption	Symmetric key
RC4	Encryption/decryption	Symmetric key
RC5	Encryption/decryption	Symmetric key

Algorithm	Usage	Keys/CSPs
RIPEMD	Hash function	N/A
Whirlpool	Hash function	N/A

Table 5. Non-Approved Algorithms

Note: ANSI X9.31 RNG will become a non-Approved algorithm on January 1, 2016.

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 cryptographic boundary are the shared library files and their integrity check HMAC files, which are delivered through Red Hat Package Manager (RPM) as listed below:

- Red Hat Enterprise Linux 6.6 OpenSSL Module RPM file with version 1.0.1e-30.el6_6.5, which contains the following files:
  - /usr/lib64/.libcrypto.so.1.0.1e.hmac (64 bits)
  - /usr/lib64/.libssl.so.1.0.1e.hmac (64 bits)
  - /usr/lib64/libcrypto.so.1.0.1e (64 bits)
  - /usr/lib64/libssl.so.1.0.1e (64 bits)
  - /usr/lib/.libcrypto.so.1.0.1e.hmac (32 bits)
  - /usr/lib/.libssl.so.1.0.1e.hmac (32 bits)
  - /usr/lib/libcrypto.so.1.0.1e (32 bits)
  - /usr/lib/libssl.so.1.0.1e (32 bits)
- The configuration of the FIPS mode is provided by the dracut-fips package with the version of the RPM file of 004-356.el6_6.1.

The OpenSSL RPM package of the Module includes the binary files, integrity check HMAC files, Man Pages and the OpenSSL Engines provided by the standard OpenSSL shared library. The OpenSSL Engines and their shared object files are not part of the Module, and therefore they must not be used when operating the Module in FIPS Approved mode.

The Module shall be installed and instantiated by the dracut-fips package with the RPM file version 004-356.el6_6.1. The dracut-fips RPM package is only used for the installation and instantiation of the Module. This code is not active when the Module is operational and does not provide any services to users interacting with the Module. Therefore the dracut-fips RPM package is outside the Module's logical boundary.

1.3.1. Hardware Block Diagram

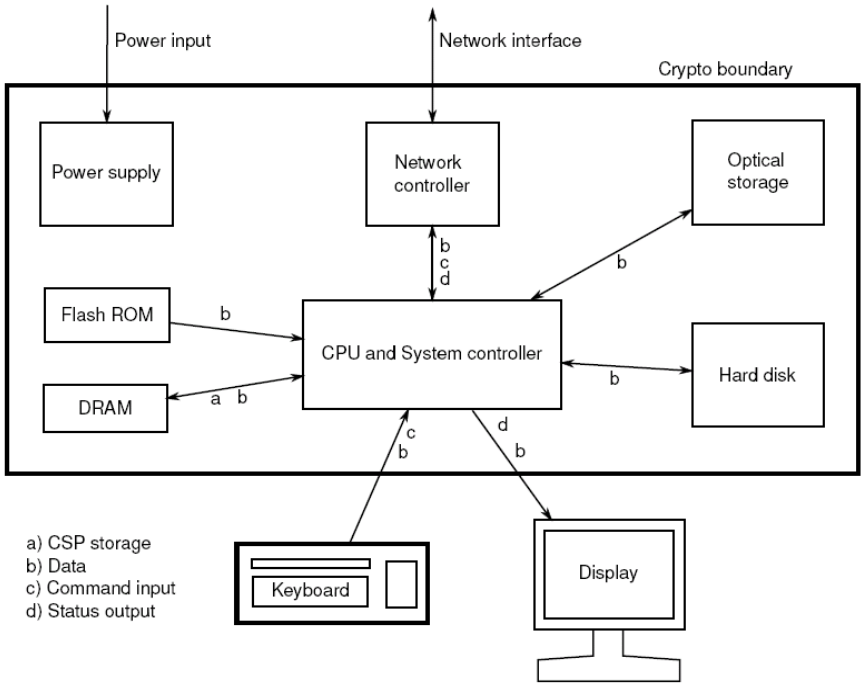

Figure 1. Hardware Block Diagram

1.3.2. Software Block Diagram

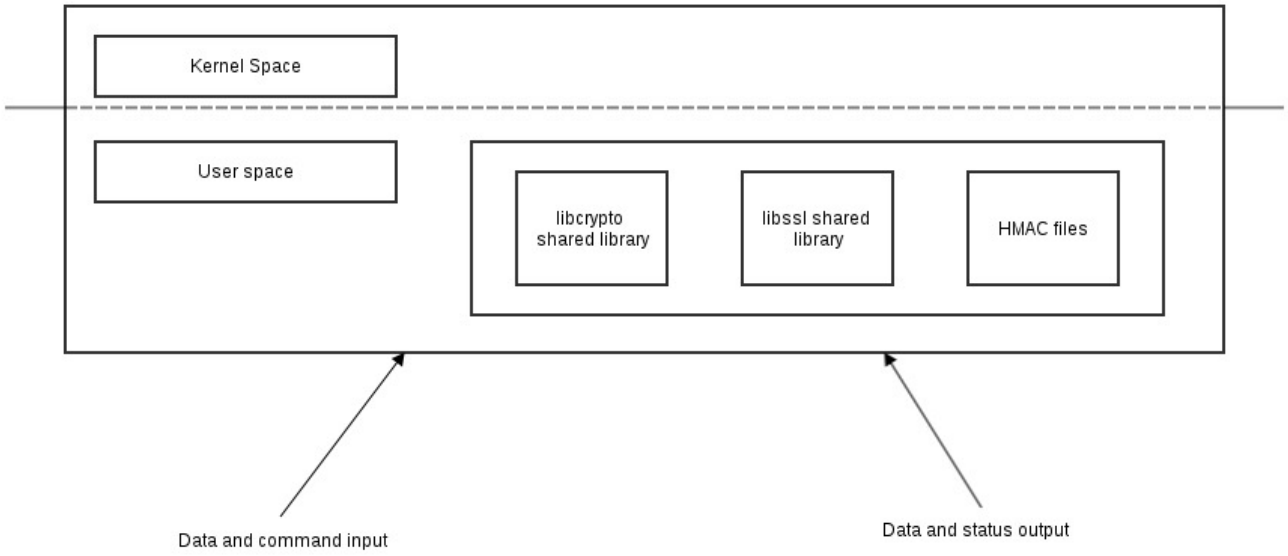

Figure 2. Software Block Diagram
(the cryptographic boundary includes the HMAC integrity files)

2. Cryptographic Module Ports and Interfaces

The physical ports of the Module are the same as the computer system on which it executes. 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, Network	API function calls, or configuration files on filesystem
Status Output	Serial port, Ethernet port, Network	API
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

There are two users of the Module:

- User
- Crypto Officer

The User and Crypto Officer roles are implicitly assumed by the entity accessing services implemented by the Module. Installation of the Module is only done by the Crypto Officer. For User documentation, please refer to the man pages of ssl(3), crypto(3) as an entry into the Module's API documentation for SSL/TLS and generic crypto support.

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 Module’s user documentation. The following table shows the services available to the various roles and the access to cryptographic keys and CSPs resulting from services.

The following table lists the Approved services available in FIPS Approved mode. Please refer to Table 3 and 4 for the Approval key size of each algorithm used in the services.

Service	Role	CSPs	Access
Symmetric encryption/decryption	User	AES and Triple-DES key	read/write/execute
Asymmetric key generation	User	RSA, DSA and ECDSA private key	read/write/execute
Digital signature generation and verification	User	RSA, DSA and ECDSA 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, Premaster Secret, Master Secret, Diffie-Hellman Private Components and EC Diffie-Hellman Private Components	read/write/execute
RSA key wrapping	User	RSA, private key	read/write/execute
Certificate Management/ Handling	User	RSA, DSA or ECDSA private key parts of certificates	read/write/execute
Keyed Hash (HMAC)	User	HMAC Key	read/write/execute
Keyed Hash (CMAC)	User	CMAC key	read/write/execute
Message digest (SHS)	User	none	read/write/execute
Random number generation (SP800-90A DRBG), including symmetric key generation	User	Entropy input string and seed	read/write/execute

Service	Role	CSPs	Access
Show status	User	none	execute
Module initialization	User	none	execute
Self-test	User	none	read/execute
Zeroize	User	All aforementioned CSPs	read/write/execute
Module installation	Crypto Officer	none	read/write

Table 7: Approved Service Details

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

Service	Role	Access
Asymmetric encryption/decryption using non-Approved RSA key size	User	read/write/execute
Symmetric encryption/decryption using non-Approved algorithms	User	read/write/execute
Digital signature generation and verification using non-Approved RSA and DSA private key	User	read/write/execute
TLS connection using keys established by Diffie-Hellman with non-Approved key sizes	User	read/write/execute
Asymmetric key generation using non-Approved RSA and DSA key size	User	read/write/execute
Random number generation using ANSI X9.31 RNG	User	read/write/execute

Table 8. Non-Approved Service Details

Note:
The module does not share CSPs between an Approved mode of operation and a non-Approved mode of operation. All cryptographic keys used in the FIPS-Approved mode of operation must be generated in the FIPS-Approved mode or imported while running in the FIPS-Approved mode. If the DRBG is used for key generation for non-Approved services in non-FIPS mode, reseeding the DRBG before and after the key generation is mandatory.

More information about the services and their associated APIs can be found in the Man Pages included in the rpm packages. The evp(3) is the starting point of the Man Pages.

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 comprises 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 Module is the single user of the 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 not be used. In addition, other tracing mechanisms offered by the Linux environment, such as ftrace or systemtap, shall not be used.

6. Cryptographic Key Management

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 deallocated.

6.1. Random Number Generation

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

The Module provides /dev/urandom from the Operational Environment as a source of random numbers for DRBG seeds and entropy input string. The Module initializes this pseudo device at system startup.

The Module performs continuous self-tests on the output of SP800-90A DRBG to ensure that consecutive random numbers do not repeat.

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

An authorized application as user (i.e., 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 API. The destruction functions then overwrite the memory occupied by keys with pre-defined values 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)

Product Name and Model: HP ProLiant DL380p Gen8
Regulatory Model Number: HSTNS-5163
Product Options: All
EMC: Class A

Product Name and Model: IBM System x3500 M4
Regulatory Model Number: 7383-AC1
Product Options: All
EMC: Class A

8. Self-Tests

FIPS 140-2 requires that the Module perform 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. All of these tests are listed and described in this section. No operator intervention is required during the running of the self-tests.
See section 9.3 for descriptions of possible self-test errors and recovery procedures.

8.1. Power-Up Tests

The Module performs 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, sign and verify
RSA	KAT, signature generation and verification are tested separately
ECDSA	PCT, sign and verify
Diffie-Hellman	Primitive "Z" Computation KAT
EC Diffie-Hellman	Primitive "Z" Computation KAT
SP 800-90A CTR_DRBG	KAT
SP 800-90A Hash_DRBG	KAT
SP 800-90A DRBG_HMAC	KAT
HMAC-SHA-1, -244, -256, -384, -512	KAT
SHA-1, -224, -256, -384, -512	KAT
CMAC	KAT
Module integrity	HMAC-SHA-256

Table 9: Module Self-Tests

8.2. Conditional Tests

Algorithm	Test
DSA	Pairwise consistency test: signature generation and verification
ECDSA	Pairwise consistency test: signature generation and verification

Algorithm	Test
RSA	Pairwise consistency test: signature generation and verification, encryption and decryption
DRBG SP800-90A	Continuous Random Number Generation Test

Table 10: Module Conditional Tests

9. Guidance

9.1. Crypto Officer Guidance

The version of the RPMs containing the FIPS validated Module is stated in section 1 above. The OpenSSL FIPS 140-2 module referenced in section 1 must be installed according to its Security Policy. The OpenSSL static libraries libcrypto.a and libssl.a in openssl-static package are not approved to be used in the FIPS Approved mode. The applications must be linked dynamically to run the OpenSSL in the FIPS Approved mode.
The RPM package of the Module can be installed by standard tools recommended for the installation of RPM packages on a Red Hat Enterprise Linux system (for example, yum, rpm, and the RHN remote management tool).
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.
Operators must first invoke OPENSSL_init(void) before using the Module.
Only the cipher types listed in section 1.2 are allowed to be used.
To bring the Module into FIPS Approved mode, perform the following:

1. Install the dracut-fips package:
   ```
   # yum install dracut-fips
   ```
2. Recreate the INITRAMFS image:
   ```
   # dracut -f
   ```

After regenerating the initramfs, the Crypto Officer has to append the following string to the kernel command line by changing the setting in the boot loader:

```
fips=1
```

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. Two alternative mechanisms are provided to switch the Module into this mode:

- 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 the application requests FIPS Approved mode using the FIPS_mode_set() function call

while passing a 1 as its parameter. This must be done prior to any initialization.
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 the version listed in chapter 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.
Either of the following two methods may be used to initialize the Module in FIPS Approved mode:

- Explicitly invoke the Module in the FIPS Approved mode by calling the FIPS_mode_set() function, which returns a “1” for success or a “0” for failure.
- Implicitly initialize the Module in the FIPS Approved mode by calling OpenSSL_add_all_algorithms() and/or SSL_library_init() functions. These functions query the file /proc/sys/crypto/fips_enabled. If the file contains 1, the Module implicitly calls FIPS_mode_set(1) which ensures that the Module will operate in the FIPS Approved mode. 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(). See Appendix A for code examples.

Interpretation of the return code is the responsibility of the host application. Prior to invocation, the Module is uninitialized in non-FIPS Approved mode by default.
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).
See Appendix A for skeleton code that illustrates how to invoke the Module.

9.2.1. TLS and Diffie-Hellman

The TLS protocol implementation provides both, the server and the client sides. 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 a single data unit encrypted with the 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-FIPS Approved SSLeay Deterministic Random Number Generator when using the RAND_* API calls.

9.2.4. AES GCM IV

In case the module's power is lost and then restored, the key used for the AES GCM encryption/decryption shall be re-distributed.

9.2.5. RSA and DSA Keys

The Module allows the use of 1024 bit RSA and DSA keys for legacy purposes, including signature generation.
As per SP800-131A, RSA and DSA must be used with 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 in FIPS Approved mode.
Application can enforce the key generation bit length restriction for RSA and DSA keys by setting the environment variable OPENSSL_ENFORCE_MODULUS_BITS. This environment variable ensures that 1024 bit keys cannot be generated.

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 power-up self-test (consisting of the algorithm-specific Pairwise Consistency and Known Answer Tests) is performed.
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 known answer test failed
- FIPS_R_SELFTEST_FAILED - a known answer test failed
- FIPS_R_TEST_FAILURE – a known answer test failed (RSA); pairwise consistency test failed (DSA)
- FIPS_R_PAIRWISE_TEST_FAILED – a pairwise consistency test during DSA or RSA key generation failed
- FIPS_R_FIPS_MODE_ALREADY_SET - the application initializes the FIPS mode when it is already initialized
- RAND_R_PRNG_STUCK – the random number generator generated two same consecutive 128 bit values

These errors are reported through the regular ERR interface of the Module and can be queried by functions such as ERR_get_error(). See the OpenSSL manual page for the function description.
A fatal error occurs only 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, the Module must be reinstalled. If downloading the software, make sure to 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
CVT	Component Verification Testing
DES	Data Encryption Standard
DRBG	Deterministic Random Bit Generator
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
RHEL	Red Hat Enterprise Linux
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] OpenSSL man pages where crypto(3) provides the introduction and link to all OpenSSL APIs regarding the cryptographic operation and ssl(3) to all OpenSSL APIs regarding the SSL/TLS protocol family
[2] FIPS 140-2 Standard, http://csrc.nist.gov/groups/STM/cmvp/standards.html
[3] FIPS 140-2 Implementation Guidance, http://csrc.nist.gov/groups/STM/cmvp/standards.html
[4] FIPS 140-2 Derived Test Requirements,http://csrc.nist.gov/groups/STM/cmvp/standards.html
[5] FIPS 197 Advanced Encryption Standard, http://csrc.nist.gov/publications/PubsFIPS.html
[6] FIPS 180-4 Secure Hash Standard, http://csrc.nist.gov/publications/PubsFIPS.html
[7] FIPS 198-1 The Keyed-Hash Message Authentication Code (HMAC), http://csrc.nist.gov/publications/PubsFIPS.html
[8] FIPS 186-4 Digital Signature Standard (DSS), http://csrc.nist.gov/publications/PubsFIPS.html
[9] ANSI X9.52:1998 Triple Data Encryption Algorithm Modes of Operation, http://webstore.ansi.org/FindStandards.aspx?Action=displaydept&DeptID=80&Acro=X9&DpName=X9,%20Inc.
[10] 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 800-56A, Recommendation for Pair-Wise Key Establishment Schemes using Discrete Logarithm Cryptography (Revised), http://csrc.nist.gov/publications/PubsFIPS.html
[14] NIST SP 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, http://csrc.nist.gov/publications/PubsFIPS.html

Appendix A: Code for Invoking the OpenSSL Module

The following code snippets demonstrate how to initialize the OpenSSL Module FIPS mode features:

```
/* Initialize the library */
OpenSSL_add_all_algorithms();

/* Optionally initialize also the SSL library */
/* SSL_library_init(); */

/* Optionally call this code to force the FIPS mode regardless
of the system-wide settings */
/* if (!FIPS_mode()) {
if (!FIPS_mode_set(1)) {
/* FIPS_mode_set() failed */
}
/* } */

/* To query, whether the FIPS mode is active */
if (FIPS_mode()) {
/* Active */
} else {
/* Inactive */
}

/* To query, whether some of the FIPS selftests failed and the module is in
the error state */
if (FIPS_selftest_failed() {
/* Failed, error state */
} else {
/* FIPS mode is either inactive or FIPS selftests succeeded */
}
```