SUSE Linux Enterprise Server 11 SP2 - OpenSSL Module v0.9.8j

FIPS 140-2 Security Policy

Version 1.6

Last Update: 2013-04-01

Contents

1 Cryptographic Module Specification

This document is the non-proprietary security policy for the 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 OpenSSL Module (hereafter referred to as the “Module”) is a software library supporting FIPS 140-2 -approved cryptographic algorithms. For the purposes of the FIPS 140-2 level 1 validation, the OpenSSL Module is SUSE Linux OpenSSL Module Version 0.9.8j-0.44.1

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 provided with the following RPM packages:

- 64 bit shared libraries delivered in libopenssl0_9_8 0.9.8j-0.44.1.x86_64.rpm – note the shared libraries delivered with the RPM and implementing the OpenSSL engines are not part of the module and are inaccessible in FIPS mode.
- 32 bit shared libraries delivered in libopenssl0_9_8-32bit-0.9.8j-0.44.1.x86_64.rpm – note the shared libraries delivered with the RPM and implementing the OpenSSL engines are not part of the module and are inaccessible in FIPS mode.
- HMAC integrity verification files for the 64 bit shared libraries delivered in libopenssl0_9_8-hmac-0.9.8j-0.44.1.x86_64.rpm.
- HMAC integrity verification files for the 32 bit shared libraries delivered in libopenssl0_9_8-hmac-32bit-0.9.8j-0.44.1.x86_64.rpm. The Module's physical cryptographic boundary is the enclosure of the computer system on which it is executing.

The FIPS_mode_set() function 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 TRUE and the Module is in FIPS mode.

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

Security Component	FIPS 140-2 Security Level
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:

- 32 bit x86_64 with Intel Westmere processor AES-NI enabled
- 32 bit x86_64 with Intel Westmere processor AES-NI disabled
- 64 bit x86_64 with Intel Westmere processor AES-NI enabled
- 64 bit x86_64 with Intel Westmere processor AES-NI disabled

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

Manufacturer	Model	O/S & Ver.
HP	ProLiant DL380	SUSE Linux Enterprise Server 11 SP2 on Intel Xeon E5620

Table 2: Tested platforms

1.2 Description of the Approved Mode

The Module supports the following FIPS 140-2 approved algorithms:

Algorithm	Validation Certificate	Usage	Keys/CSPs
AES (Software-only) (ECB, CBC, OFB, CFB 1, CFB 8, CFB 128)	Certs #2053, #2055	encrypt/decrypt	AES keys 128, 192, 256 bits
AES (using AES-NI processor instructions)	Certs #2052, #2054	encrypt/decrypt	AES keys 128, 192, 256 bits
Triple-DES (ECB, CBC, CFB, OFB)	Certs #1323, #1324	encrypt/decrypt	Triple-DES keys 168 bits
DSA	Certs #650, #651	sign, verify, and keygen	DSA keys 1024 bits
ANSI X9.31PRNG	Certs #1073, #1074	random number generation	PRNG seed value and seed key 128 bits
SHA-1	Certs #1797, #1798	hashing	N/A
SHA-224	Certs #1797, #1798	hashing	N/A
SHA-256	Certs #1797, #1798	hashing	N/A
SHA-384	Certs #1797, #1798	hashing	N/A
SHA-512	Certs #1797, #1798	hashing	N/A
HMAC-SHA-1	Certs #1249, #1250	message integrity	HMAC Key
HMAC-SHA224	Certs #1249, #1250	message integrity	HMAC Key
HMAC-SHA256	Certs #1249, #1250	message integrity	HMAC Key

Algorithm	Validation Certificate	Usage	Keys/CSPs
HMAC-SHA384	Certs #1249, #1250	message integrity	HMAC Key
HMAC-SHA512	Certs #1249, #1250	message integrity	HMAC Key
RSA (X9.31, PKCS #1.5, PSS)	Certs #1069, #1070	sign, verify, and keygen	RSA keys 1024 to 16384 bits

Table 3: Approved algorithms

Please note that AES cipher is implemented with common C code in the module. If the AES-NI instruction set is supported by the processor and it is enabled in the operational environment, the module uses the AES-NI instruction set implementation as the AES cipher. If the AES-NI instruction set is not supported or is disabled in the operational environment, the module uses the C implementation as the AES cipher. This applies to both 32-bit and 64-bit of the module.

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

Algorithm	Validation Certificate	Usage	Keys/CSPs
Diffie-Hellman	N/A, see caveat below	Key agreement and establishment	None
RSA (encrypt, decrypt)	N/A, see caveat below	Key agreement and establishment	RSA keys 1024 to 16384 bits
MD5	N/A, see caveat below	Message digest	N/A
HMAC-MD5	N/A, see caveat below	Message digest	N/A

Table 4: Non-Approved algorithms

CAVEATS:

1) Diffie-Hellman (key agreement; key establishment methodology provides between 80 and 160 bits of encryption strength)

2) RSA (key wrapping; key establishment methodology provides between 80 and 160 bits of encryption strength)

3) MD5 and HMAC-MD5 for use in TLS only

4) The module generates cryptographic keys whose strengths are modified by available entropy – 160 bits

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

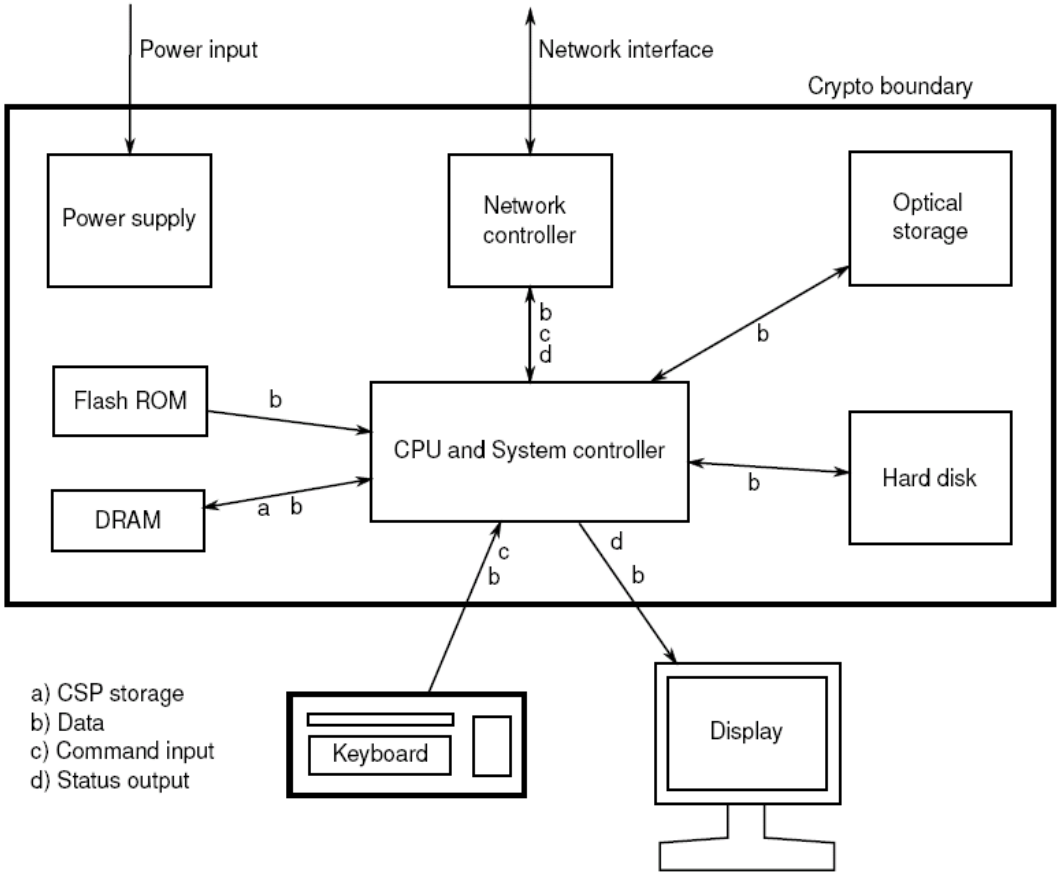

Figure 1. Hardware Block Diagram

1.3.2 Software Block Diagram

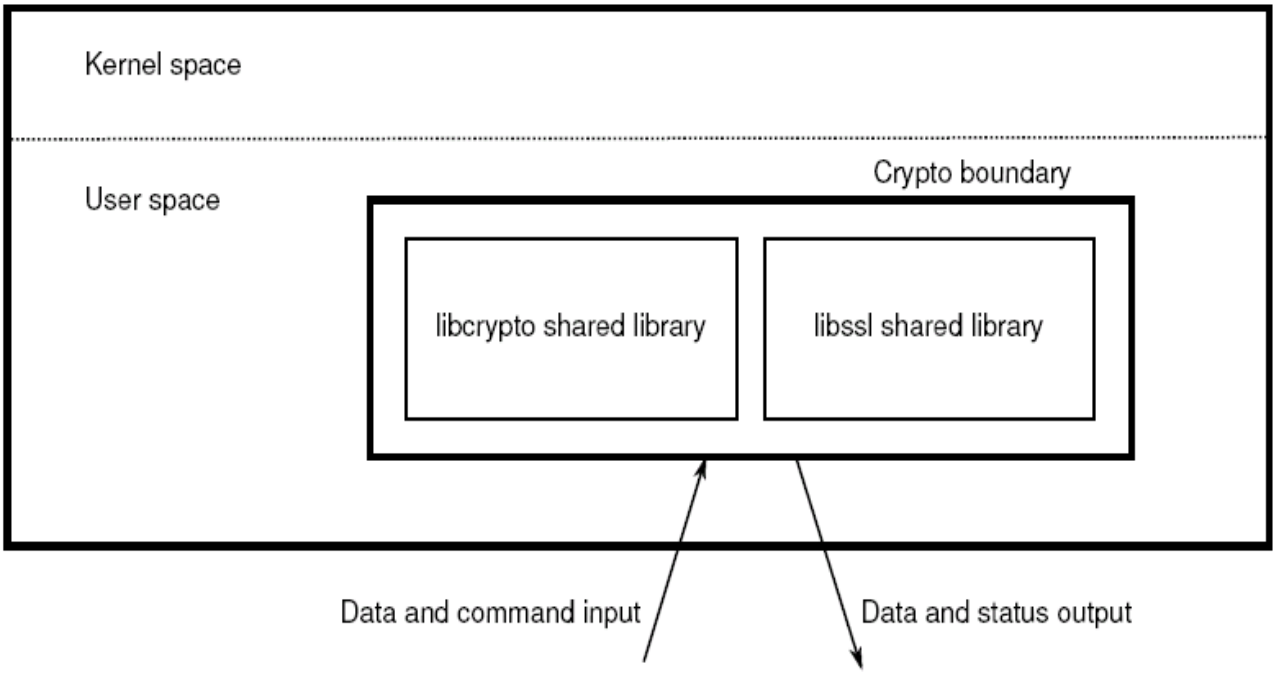

Figure 2. Software Block Diagram

1.4 SUSE Linux Cryptographic Modules and FIPS 140-2 Validation

1.4.1 FIPS Approved Mode

The 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 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 configuration files on filesystem
Status Output	Serial port, Ethernet port	API
Power Input	PC Power Supply Port	N/A

Table 5: 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, which are identified along with their allowed services in Table 5 (and the services are further detailed in Table 6).

Role	Services (see list below)
User	Encryption, Decryption (symmetric and public/private), Random Numbers
Crypto Officer	Configuration of FIPS 140-2 validated mode, Encryption, Decryption (symmetric and public/private), Random Numbers

Table 6: 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 shows the services available to the various roles and the access to cryptographic keys and CSPs resulting from services.

Service	Role	Algorithms and CSPs	Access
Symmetric encryption/decryption	User, Crypto Officer	symmetric key AES, TDES	read/write/execute
Key transport	User, Crypto Officer	asymmetric private key RSA	read/write/execute
Digital signature	User, Crypto Officer	asymmetric private key RSA, DSA	read/write/execute
Symmetric key generation	User, Crypto Officer	symmetric key AES, TDES	read/write/execute
TLS	User, Crypto Officer	symmetric key AES, TDES asymmetric public/private key RSA, HMAC Key	read/write/execute
TLS Key Agreement	User, Crypto Officer	symmetric key	read/write/execute

Service	Role	Algorithms and CSPs	Access
		AES, TDES asymmetric public/private key RSA, HMAC Key, Premaster Secret, Master Secret and DH Secret	
Certificate Management/ Handling	User, Crypto Officer	Certificates	read/write/execute
Asymmetric key generation	User, Crypto Officer	asymmetric private key RSA, DSA	read/write/execute
Keyed Hash (HMAC)	User, Crypto Officer	HMAC Key, HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512	read/write/execute
Message digest (SHS)	User, Crypto Officer	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	read/write/execute
Random number generation	User, Crypto Officer	PRNG Seed and Seed Key	read/write/execute
Show status	User, Crypto Officer	none	execute
Module initialization	User, Crypto Officer	none	execute
Self test	User, Crypto Officer	HMAC SHA-256 key	read/execute
Zeroize	User, Crypto Officer	symmetric key, asymmetric key, HMAC key, Seed and Seed key	read/write/execute

Table 7: 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 deallocated.

6.1 Random Number Generation

The Module employs an ANSI X9.31-compliant random number generator for creation of asymmetric and symmetric keys.

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

The kernel performs continual tests on the random numbers it uses, to ensure that the seed and seed key input to the Approved RNG do not have the same value. The kernel also performs continual tests on the output of the approved RNG 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 (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)

Product Name and Model: HP ProLiant DL380 G7
Regulatory Model Number: HSTNS-5141
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
DSA	Pairwise consistency
Triple-DES	KAT
AES	KAT
AES-NI	KAT
ANSI X9.31PRNG	KAT
RSA	KAT
SHA-1	KAT
SHA-256	KAT
SHA-512	KAT
SHA-224	Tested with SHA-256 KAT
SHA-384	Tested with SHA-512 KAT
HMAC-SHA-1	KAT
HMAC-SHA-224	KAT
HMAC-SHA-256	KAT
HMAC-SHA-384	KAT
HMAC-SHA-512	KAT
HMAC-SHA-256	Module integrity

Table 8: Module self tests

8.2 Conditional Tests

Algorithm	Test
DSA	Pairwise consistency
RSA	Pairwise consistency
PRNG	Continuous RNG test

Table 9: 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. Operators must first invoke OPENSSL_init(void) before using the module.

ENGINE_register_* and ENGINE_set_default_* function calls are prohibited while in the Approved mode. Furthermore, once the Approved mode is entered, it must not be exited, which prohibits calls to FIPS_mode_set(0).

Only the cipher types listed in section 1.2 are allowed to be used.

To bring the module into FIPS mode, perform the following:

Edit the bootloader configuration:
vi /boot/grub/menu.lst

Add the following kernel commandline parameter to the bootloader configuration entry for your installation:
fips=1

Reboot to apply this setting. After the reboot, you can check the content of the file /proc/sys/crypto/fips_enabled to see if the FIPS-approved mode is enabled on your system.

The Module allows linking of 32 bit, as well as, 64 bit binaries without modifying the installation base. Both, the 32 bit and the 64 bit binary RPM packages can be installed concurrently. For each word size version, SLES provides an individual development RPM package that includes the header files.

Please note that only one of these two development packages can be installed at one time. If the 64 bit development package is installed, users can only compile 64 bit binaries. Contrary, if the 32 bit development package is installed, only 32 bit binaries can be compiled. Any attempt to compile a different word size than what is provided with the development package will ultimately fail during link time. In addition, several data structures will have different types which have an impact on the stability of the compiled software. The crypto officer may change the installed development package as necessary without impacting the FIPS 140-2 certified module configuration.

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 Approved mode:

- Explicitly invoke the module in the 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().

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

The Module performs the self tests described in section 8. See section 9.3 for descriptions of possible self test errors and recovery procedures.

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

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 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, you must reinstall the Module. If you downloaded the software, verify the package hash to confirm a proper download.

10 Mitigation of Other Attacks

The Module does not contain additional security mechanisms beyond the requirements for FIPS 140-2 level 1 cryptographic modules.

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 explicitely 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
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
FSM	Finite State Model
HMAC	Hash Message Authentication Code
LDAP	Lightweight Directory Application Protocol
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 180-3 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 186-3 Digital Signature Standard (DSS), http://csrc.nist.gov/publications/PubsFIPS.html

[8] 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.