OpenSSL FIPS
Runtime Module
Version 1.2
By the
Open Source Software Institute
http://www.oss-institute.org/

OpenSSL FIPS 140-2 Security Policy

Version 1.2

March 11, 2009

Copyright Notice

Sponsored by

U.S. Department of Defense
Offices of Advanced Systems and Concepts
Open Technology Development program

Acknowledgments

The Open Source Software Institute (OSSI) serves as the "vendor" for this validation. Project management coordination for this effort was provided by the OSSI:

Steve Marquess Project Manager	301-524-9915 marquess@oss-institute.org
John Weathersby Executive Director	601-427-0152 office/601-818-7161 cell jmw@oss-institute.org
Open Source Software Institute Administrative Office P.O. Box 547 Oxford, MS 38655	601-427-0156 fax http://oss-institute.org/

with technical work by:

Stephen Henson 4 Monaco Place, Westlands, Newcastle-under-Lyme Staffordshire. ST5 2QT. England, United Kingdom	shenson@drh-consultancy.co.uk http://www.drh-consultancy.co.uk/
Andy Polyakov Chalmers University of Technology SE-412 96 Gothenburg Sweden	appro@fy.chalmers.se

in coordination with the OpenSSL Team at www.openssl.org.

Validation testing was performed by Aspect Labs, a division of BKP Security, Inc. For information on validation or revalidations of software contact:

Aspect Labs 3080 Olcott Street, Suite 110-A Santa Clara, CA 95054-3221	888-347-7140 info@aspectlabs.com http://www.aspectlabs.com/

Table of Contents

1. Introduction

This document is the non-proprietary security policy for the OpenSSL FIPS Runtime Module. This document was prepared as part of the Federal Information Processing Standard (FIPS) 140-2 Level 1 validation process.

FIPS 140-2, *Security Requirements for Cryptographic Modules*, describes the requirements for cryptographic modules. For more information about the FIPS 140-2 standard and the cryptographic module validation process see http://csrc.nist.gov/cryptval/.

2. Module Specification

The OpenSSL FIPS Runtime Module (hereafter referred to as the Module) is a software library supporting FIPS-approved cryptographic algorithms. For the purposes of the FIPS 140-2 level 1 validation, the OpenSSL FIPS Runtime Module v1.2 is a single shared library module file[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 *logical* cryptographic boundary of the Module is the shared library file itself. The *physical* cryptographic boundary of the Module is the enclosure of the computer system on which it is executing. The Module performs no communications other than with the process that calls it. It makes no network or interprocess connections and creates no files.

The Module was tested on the following platforms:

1) Windows XP Service Pack 2
2) Fedora Linux 9

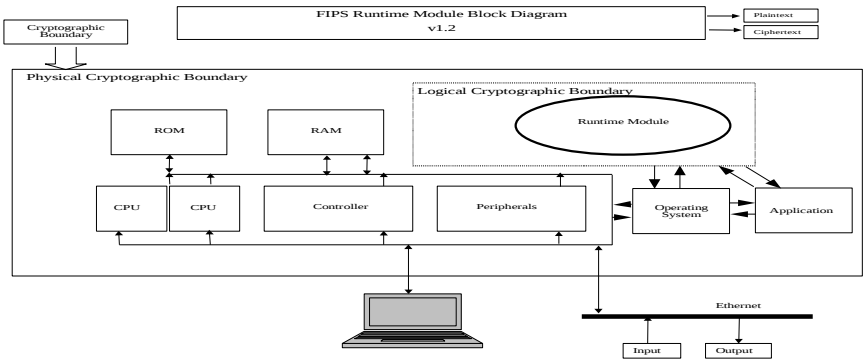

Figure 2

1 `libfips.so.0.9.8` (Linux/Unix) or `libosslfips.dll` (Windows)

2.1 Roles and Services

The Module meets all FIPS 140-2 level 1 requirements for Roles and Services, implementing both Crypto-User and Crypto-Officer roles. As allowed by FIPS 140-2, the Module does not support user authentication for those roles. Only one role may be active at a time and the Module does not allow concurrent operators.

The User and Crypto Officer roles are implicitly assumed by the entity accessing services implemented by the Module. The Crypto Officer can install and initialize the Module. The Crypto Officer role is implicitly entered when installing the Module or performing system administration functions on the host operating system.

- User Role: Loading the Module and calling any of the API functions. This role has access to all of the services provided by the Module.
- Crypto-Officer Role: Installation of the Module on the host computer system. This role is assumed implicitly when the system administrator installs the Module library file.

Service	*Role*	*CSP*	*Access*
Symmetric encryption/decryption	User	AES and TDES symmetric keys	read/write/execute
Key wrapping for key transport	User	RSA public/private key pairs	read/write/execute
DH primitives	User	Diffie-Hellman keys	read/write/execute
Digital signature	User	RSA and DSA asymmetric keys	read/write/execute
Symmetric key generation	User	AES, TDES and HMAC symmetric keys	read/write/execute
Asymmetric key generation	User	RSA, DSA and Diffie-Hellman keys	read/write/execute
Keyed Hash (HMAC)	User	HMAC keys	read/write/execute
Message digest (SHS)	User	none	read/write/execute
Random number generation (ANSI X9.31)	User	RNG seed and seed key	read/write/execute
Show status	User	none	execute
Module initialization	User	none	execute
Self test	User	Integrity-check HMAC key	execute
Zeroize	User	all symmetric and	write

Service	*Role*	*CSP*	*Access*
		asymmetric keys, as well as parameters other than those used by Data Input and Output Interfaces	

Table 2.1

Note that only the private key components of public/private key pairs are CSPs. The public keys are assumed to be publicly visible.

2.2 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.

FIPS Interface	*Physical Port*	*Module Interface*
Data Input	Physical Ports of a GPC	API input parameters
Data Output	Physical Ports of a GPC	API output parameters
Control Input	Physical Ports of a GPC	API function calls and parameters, other than those used by Data Input and Output interfaces
Status Output	Physical Ports of a GPC	API return codes
Power Input	Power Port of a GPC	N/A

Table 2.2

2.3 Self Tests

The Module performs both power-up self tests at module initialization and conditional tests during operation. Input, output, and cryptographic functions cannot be performed while the Module is in a self-test or error state as 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 fail and thus no further cryptographic operations are possible.

Power-Up Self Tests

Algorithm	Test
AES	KAT
Triple-DES	KAT
DSA signatures	pairwise consistency test, sign/verify
RSA signatures	KAT
ANSI X9.31 PRNG	KAT
HMAC-SHA-1	KAT
HMAC-SHA-224	KAT
HMAC-SHA-256	KAT
HMAC-SHA-384	KAT
HMAC-SHA-512	KAT
SHA-1	KAT[2]
SHA-224	KAT[2]
SHA-256	KAT[2]
SHA-384	KAT[2]
SHA-512	KAT[2]
module integrity check	HMAC-SHA-1

Table 2.3a

Conditional Self Tests

Algorithm	Test
DSA key pair generation	pairwise consistency
RSA key pair generation	pairwise consistency
PRNG	continuous RNG test

Table 2.3b

A single initialization call, `FIPS_mode_set,` is required to initialize the Module for operation in the FIPS 140-2 Approved mode. When the Module is in FIPS mode all security functions and cryptographic algorithms are performed in Approved mode.

2 Tested as part of the HMAC known answer tests.

The FIPS mode initialization is performed when the application invokes the `FIPS_mode_set` call which returns a "1" for success and "0" for failure. Interpretation of this return code is the responsibility of the host application. Prior to this invocation the Module is uninitialized in the non-FIPS mode by default.

The `FIPS_mode_set` function verifies the integrity of the runtime executable using a HMAC-SHA-1 digest computed at build time. If this computed HMAC-SHA-1 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. If any component of the power-up self-test fails an internal global error flag is set to prevent subsequent invocation of any cryptographic function calls. Any such power-up self test failure is a hard error that can only be recovered by reinstalling the Module[3]. If all components of the power-up self-test are successful then the Module is in FIPS mode. The power-up self-tests may be performed at any time by restarting the module.

A power-up self-test failure can only be cleared by a successful `FIPS_mode_set` invocation.

2.4 Mitigation of Other Attacks

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

2.5 Physical Security

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

3 The FIPS_mode_set() function could be re-invoked but such re-invocation does not provide a means from recovering from an integrity test or known answer test failure.

3. Secure Operation

The tested operating systems segregate user processes into separate process spaces. Each process space is an independent virtual memory area that is logically separated from all other processes by the operating system software and hardware. The Module functions entirely within the process space of the process that invokes it, and thus satisfies the FIPS 140-2 requirement for a single user mode of operation. The "single user mode" means that for each spawned instance of a cryptographic module, only one operator may access the module at a time.

The Module is installed using the instructions in Appendix A appropriate to the target system. A complete revision history of the source code from which the Module was generated is maintained in a version control database[4].

Upon initialization of the Module by invocation of the `FIPS_mode_set` call the module will run its power-up self tests. Successful completion of the power-up self tests as indicated by a return value of "1" from the `FIPS_mode_set` call ensures that the module is operating in the FIPS mode of operation.

The self-tests can be called on demand by restarting the module (i.e., reloading the module and re-invoking the FIPS mode initialization API call).

4 See http://cvs.openssl.org/

4. Cryptographic Key Management

For each API call the calling application provides public and private keys (if any) specified in the API call.

4.1 Key Generation

The Module supports generation of DH, DSA, and RSA public-private key pairs. The Module employs an ANSI X9.31 compliant random number generator for creation of asymmetric and symmetric keys as well as FIPS 186-2 compliant DSA key pair generation algorithm.

4.2 Key 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.

4.3 Key Access

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

4.4 Key Protection and Zeroization

Keys residing in internally allocated data structures can only be accessed using the Module defined API. The operating system protects memory and process space from unauthorized access.

Only the process that creates or imports keys can use or export them. No persistent storage of key data is performed by the Module. All API functions are executed by the invoking process in a non-overlapping sequence such that no two API functions will execute concurrently.

Rebooting of the system will zeroize any keys present in volatile RAM.

4.5 Cryptographic Algorithms

The Module supports the following FIPS approved or allowed algorithms:

Algorithm	*Validation Certificate*	*Usage*
AES	#681, #682	encrypt/decrypt
TDES	#623, #624	encrypt/decrypt
Diffie-Hellman	(allowed in FIPS mode, see caveat below)	DH primitives
DSA	#257, #258	sign and verify
PRNG	#397, #398	random number generation
RSA (X9.31, PKCS #1.5, PSS)	#318, #319	sign and verify

Algorithm	*Validation Certificate*	*Usage*
RSA encrypt/decrypt	(allowed in FIPS mode, see caveat below)	key wrapping
SHA-1	#711, #712	hashing
SHA-224	#711, #712	hashing
SHA-256	#711, #712	hashing
SHA-384	#711, #712	hashing
SHA-512	#711, #712	hashing
HMAC-SHA-1	#362, #363	message integrity
HMAC-SHA224	#362, #363	message integrity
HMAC-SHA256	#362, #363	message integrity
HMAC-SHA384	#362, #363	message integrity
HMAC-SHA512	#362, #363	message integrity

Table 4.5a

Diffie-Hellman (provides between 80 and 256 bits of encryption strength).

Diffie-Hellman, DSA, and RSA do not permit a key size of less than 1024 bits when in FIPS mode.

RSA (key wrapping; key establishment methodology provides between 80 and 150 bits of encryption strength).

The module supports the DES encryption algorithm, which shall not be used in the Approved mode of operation.

Appendix A Installation Instructions

Installation consists of copying the shared library file to the appropriate location where it can be referenced by the host operating system at application runtime.

For Unix and Linux systems the shared library file is named *libfips.so.0.9.8*. For Microsoft Windows the shared library file is named *libosslfips.dll*.

Installation instructions:

1. Copy the shared library file to the appropriate location on the host system.
2. As appropriate define or register the shared library for reference by the operating system (O/S) run-time loader. This step will vary depending on the O/S and whether the shared library is to be installed for global access by all users or only for use by a specific application. For Windows simply placing the shared library in the same directory as the calling application suffices for use by that application. For Unix/Linux the LD_LIBRARY_PATH environment variable (or possibly others such as LD_PRELOAD) can be defined, or the *ldconfig* command can be used to configure the system-wide cache of shared libraries listed in the */etc/ld.so.conf* file.
3. The module is now available for use.