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

OpenSSL FIPS 140-2 Security Policy

Version 1.2

August 8, 2008

Copyright Notice

Sponsored by:

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 The DOMUS IT Security Laboratory. For information on validation or revalidations of software contact:

Christian Brych FIPS 140 Program Manager DOMUS IT Security Laboratory 2650 Queensview Drive Suite 100 Ottawa, Ontario K2B 8H6	613-726-5091 office 613-867-1241 cell cbrych@nuvo.com http://www.domusitsl.com/

Table of Contents

1. Introduction

This document is the non-proprietary security policy for the OpenSSL FIPS Object 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 Object 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 Object Module v1.2 is a single object module file named *fipscanister.o* (Linux®[1]/Unix®[2]) or *fipscanister.lib* (Microsoft Windows®[3]). 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 *fipscanister* object module. 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:

```
U1 Linux x86 no-asm     Linux.2.6.18_i686_gcc-4.1.2 (OpenSuSE 10.2) no-asm
U2 Linux x86-64 no-asm  Linux.2.6.20_x86-64_gcc-4.1.2 (OpenSuSE 10.2)
U3 Linux x86 asm        Linux.2.6.18_i686_gcc-4.1.2 (OpenSuSE 10.2)
U4 Linux x86-64 asm     Linux.2.6.20_x86-64_gcc-4.1.2 (OpenSuSE 10.2)
W1 Windows x86 no-asm   WinXP.SP2_i386_MSVC.8.0 no-asm
W2 Windows x64 no-asm   WinXP.SP2_x86-64_MSVC.8.0 no-asm
W3 Windows x86 asm      WinXP.SP2_i386_MSVC.8.0 NASM, SSE2
W4 Windows x64 asm      WinXP.SP2_x86-64_MSVC.8.0
```

The "asm" designation means that assembler language optimizations were enabled when the binary code was built, "no-asm" means that only C language code was compiled.

1 Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
2 UNIX is a registered trademark of The Open Group
3 Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

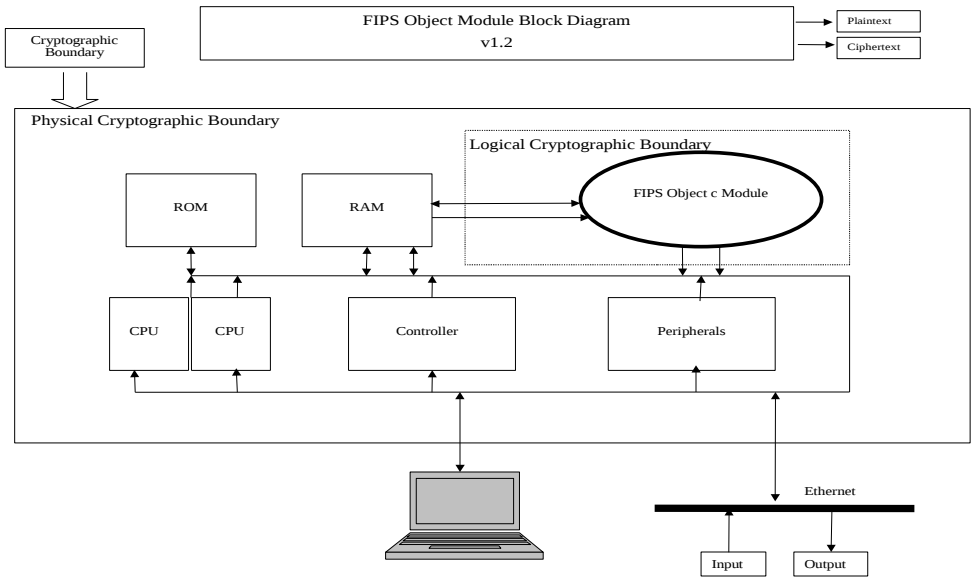

Figure 2

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, Crypto Officer	symmetric key AES, TDES	read/write/execute

Service	*Role*	*CSP*	*Access*
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
Asymmetric key generation	User, Crypto Officer	asymmetric private key RSA, DSA	read/write/execute
Keyed Hash (HMAC)	User, Crypto Officer	HMAC SHA-1 key HMAC-SHA-1	read/write/execute
Message digest (SHS)	User, Crypto Officer	none SHA-1, SHA-2	read/write/execute
Random number generation	User, Crypto Officer	seed key seed AES	read/write/execute
Show status	User, Crypto Officer	none	execute
Module initialization[1]	User, Crypto Officer	none	execute
Self test	User, Crypto Officer	none	execute
Zeroize	User, Crypto Officer	symmetric key, asymmetric key, HMAC-SHA-1 key, seed key AES	

Table 2.1

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	Ethernet ports	API input parameters
Data Output	Ethernet ports	API output parameters

1 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; see section 2.3.

FIPS Interface	*Physical Port*	*Module Interface*
Control Input	Keyboard, Serial port, Ethernet port	API function calls
Status Output	Keyboard, Serial port, Ethernet port	API return codes
Power Input	PCI Compact Power Connector	N/A

Table 2.2

2.3 Self Tests

The Module performs both power-up self tests at module initialization[2] 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 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	pairwise consistency test, sign/verify
RSA	KAT
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[3]
SHA-224	KAT[1]
SHA-256	KAT[1]

2 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; see section 2.3.

3 Tested as part of the HMAC known answer tests.

Algorithm	Test
SHA-384	KAT[1]
SHA-512	KAT[1]
module integrity	HMAC-SHA-1

Table 2.3a

Conditional Self Tests

Algorithm	Test
DSA	pairwise consistency
RSA	pairwise consistency
PRNG	continuous 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.

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[4]. 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 with a separate function call, `FIPS_selftest().` This function call also returns a "1" for success and "0" for failure, and interpretation of this return code is the responsibility of the host application.

A power-up self-test failure can only be cleared by a successful `FIPS_mode_set()` invocation. No operator intervention is required during the running of the self-tests.

2.4 Mitigation of Other Attacks

The Module does not contain additional security mechanisms beyond the requirements for FIPS 140-2

4 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.

level 1 cryptographic modules.

2.5 Physical Security

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

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 Module is installed using one of the set of 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[5]. The HMAC-SHA-1 of the Module distribution file as tested by the CMT Laboratory and listed in Appendix A is verified during installation of the Module file as described in Appendix A.

Upon initialization[6] of the Module, the module will run its power-up self tests. Successful completion of the power-up self tests[7] ensures that the module is operating in the FIPS mode of operation.

As the Module has no way of managing keys, any keys that are input or output from applications utilizing the module must be input or output in encrypted form using FIPS approved algorithms.

The self-tests can be called on demand by reinitializing the module using the `FIPS_mode_set()` function call, or alternatively using the `FIPS_selftest()` function call.

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

6 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; see section 2.3.

7 The power-up self tests are performed as part of the FIPS mode initialization process; see section 2.3.

4. Cryptographic Key Management

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.

The developer shall use entropy sources that contain at least 128 bits of entropy to seed the RNG as the module is not capable of detecting randomness or quality of the seeding material provided.

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 Crypto-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. Zeroization of sensitive data is performed automatically by API function calls for intermediate data items, and on demand by the calling process using Module provided API function calls provided for that purpose.

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.

The calling process can perform key zeroization of keys by calling an API function.

4.5 Cryptographic Algorithms

The Module supports the following FIPS approved or allowed algorithms:

Algorithm	*Validation Certificate*	*Usage*	*Keys/CSPs*
AES	#695	encrypt/decrypt	AES keys 128, 192, 256 bits
TDES	#627	encrypt/decrypt	Triple-DES keys 168 bits
DSA	#264	sign and verify	DSA keys 1024 bits
PRNG (ANSI X9.31 Appendix	#407	random number generation	PRNG seed value is 128 bits; seed key values are 128 bits,

Algorithm	*Validation Certificate*	*Usage*	*Keys/CSPs*
A.2.4 using AES)			192 bits, and 256 bits
RSA (X9.31, PKCS #1.5, PSS)	#323	sign and verify	RSA keys 1024 to 16384 bits
RSA encrypt/decrypt	(allowed in FIPS mode, see caveat below)	key wrapping	RSA (key wrapping; key establishment methodology provides 80 to 256 bits of encryption strength
SHA-1	#723	hashing	N/A
SHA-224	#723	hashing	N/A
SHA-256	#723	hashing	N/A
SHA-384	#723	hashing	N/A
SHA-512	#723	hashing	N/A
HMAC-SHA-1	#373	message integrity	HMAC key
HMAC-SHA224	#373	message integrity	HMAC key
HMAC-SHA256	#373	message integrity	HMAC key
HMAC-SHA384	#373	message integrity	HMAC key
HMAC-SHA512	#373	message integrity	HMAC key

Table 4.5a

DSA supports a key size of less than 1024 bits except when not in FIPS mode.

RSA (key wrapping; key establishment methodology provides between 80 and 256 bits of encryption strength).

The Module supports the following non-FIPS approved algorithms:

Algorithm	*Usage*	*Keys/CSPs*
Diffie-Hellman	key establishment	Diffie-Hellman keys

Table 4.5b

The Module only provides functions that implement Diffie-Hellman primitives. The shared secret provides between 80 and 219 bits of encryption strength.

Appendix A Installation Instructions

The eight test platforms represent different combinations of installation instructions and "code paths" -- the distinct set of object code executed depending on the host hardware and operating system platform.

Platform	Code Path	Installation Instruction Set
1	pure C 32 bit	U1
2	pure C 32 bit	W1
3	pure C 64 bit	U1
4	pure C 64 bit	W1
5	x86 asm	U2
6	x86 asm	W2
7	x86-64 asm	U2
8	x84-64 asm	W2

The command sets U1, W1, ... are:

```
U1:
        ./config fipscanisterbuild no-asm
        make
        make install

U2:
        ./config fipscanisterbuild
        make
        make install

W1:
        ms\do_fips no-asm

W2:
        ms\do_fips
```

Each of these command sets are relative to the top of the directory containing the uncompressed and expanded contents of the distribution file *openssl-fips-1.2.tar.gz*.

Installation instructions

1. Download and copy the distribution file *openss-fips-1.2.tar.gz* to the target system. This distribution file can be downloaded from *http://www.openssl.org/source/*.

2. Verify that the SHA-1 HMAC digest of the distribution file (see Appendix B). Note that if a suitable utility to generate SHA1 HMAC digests is not available, this check will need to be deferred until the *openssl* command is generated in the following step.
3. Execute one of the installation command sets U1, W1, U2, W2 as shown above. No other command sets shall be used.
4. The resulting *fipscanister.o* or *fipscanister.lib* file is now available for use.
5. The calling application enables FIPS mode by calling the *FIPS_mode_set()* function.

Note that failure to use one of the specified commands sets exactly as shown will result in module that cannot be considered compliant with FIPS 140-2.

Linking the Runtime Executable Application

Note that applications interfacing with the FIPS Object Module are outside of the cryptographic boundary. When linking the application with the FIPS Object Module two steps are necessary:

1. The HMAC-SHA-1 digest of the FIPS Object Module file must be calculated and verified against the installed digest to ensure the integrity of the FIPS object module.

2. A HMAC-SHA1 digest of the FIPS Object Module must be generated and embedded in the FIPS Object Module for use by the `FIPS_mode_set()` function at runtime initialization.

The OpenSSL distribution contains a reference utility[8] which can be used to perform the verification of the FIPS Object Module and to generate the new HMAC-SHA-1 digest for the runtime executable application. Failure to embed the digest in the executable object will prevent initialization of FIPS mode.

At runtime the `FIPS_mode_set()` function compares the embedded HMAC-SHA-1 digest with a digest generated from the FIPS Object Module object code. This digest is the final link in the chain of validation from the original source to the runtime executable application file.

8 This utility is the "*openssl sha"* command with the *-hmac* option switch. It is included in the FIPS Object Module distribution and also in all recent OpenSSL distributions. The version of this utility generated from the FIPS Object Module distribution can be used to check the validity of the distribution tarball digest after the fact. Note that in principle a software distribution could be corrupted in such a way as to incorrectly return the expected digest. This risk is present for all validated products, of course, and would be even harder to detect without visible source code.

Appendix B Controlled Distribution File Fingerprint

The *OpenSSL FIPS Object Module v1.2* consists of the FIPS Object Module (the *fipscanister.o* or *fipscanister.lib* contiguous unit of binary object code) generated from the specific source files found in the specific special OpenSSL distribution o*penssl-fips-1.2.tar.gz* with HMAC-SHA-1 digest of

```
79193087e8115df76d3de1f346f7410df79cf6e0
```

at http://www.openssl.org/source/openssl-fips-1.2.tar.gz. This digest can be calculated and displayed with the command

```
openssl sha1 -hmac etaonrishdlcupfm openssl-fips-1.2.tar.gz
```

The set of files specified in this tar file constitutes the complete set of source files of this module. There shall be no additions, deletions, or alterations of this set as used during module build. The OpenSSL distribution tar file shall be verified using the above HMAC-SHA-1 digest.

The arbitrary 16 byte key of:

```
65 74 61 6f 6e 72 69 73 68 64 6c 63 75 70 66 6d
```

(equivalent to the ASCII string "`etaonrishdlcupfm`") is used to generate the HMAC-SHA-1 value for the FIPS Object Module integrity check.