User Guide

for the

OpenSSL FIPS Object Module v2.0

(including v2.0.1, v2.0.2, v2.0.3, v2.0.4, v2.0.5, v2.0.6, v2.0.7,2.0.8,2.0.9)

OpenSSL Software Foundation

September 1, 2015

Copyright and Trademark Notice

Sponsored by:

Defense Advanced Research Projects Agency (DARPA)
Transformative Apps Program

Intersoft International, Inc.

opengear
Advanced Console Management

PKWARE®

PSW GROUP

Department of Homeland Security
Science and Technology Directorate

Innominate
Security Technologies

Sponsored by:

Dell Inc.

sponsor of Beaglebone Black platforms

Acknowledgments

The OpenSSL Software Foundation (OSF) serves as the "vendor" for this validation. Project management coordination for this effort was provided by:

Steve Marquess The OpenSSL Software Foundation 1829 Mount Ephraim Road Adamstown, MD 21710 USA	+1 877-673-6775 marquess@openssl.com

with technical work by:

Dr. Stephen Henson 4 Monaco Place, Westlands, Newcastle-under-Lyme Staffordshire. ST5 2QT. England, United Kingdom	 shenson@openssl.com shenson@drh-consultancy.co.uk http://www.drh-consultancy.co.uk/
Andy Polyakov Chalmers University of Technology SE-412 96 Gothenburg Sweden	 appro@openssl.org appro@fy.chalmers.se
Tim Hudson P.O. Box 6389 Fairfield Gardens 4103 Australia	 tjh@cryptsoft.com http://www.cryptsoft.com/

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

Validation testing was performed by Infogard Laboratories. For information on validation or revalidations of software contact:

Marc Ireland FIPS Program Manager, CISSP InfoGard Laboratories 709 Fiero Lane, Suite 25 San Luis Obispo, CA 93401	805-783-0810 tel 805-783-0889 fax mireland@infogard.com http://www.infogard.com/

Revision History

This document will be revised over time as new information becomes available; check http://www.openssl.org/docs/fips/ for the latest version. Suggestions for additions, corrections, or improvement are welcome and will be gratefully acknowledged; please send document error reports or suggestions to userguide@openssl.com.

Date	Description
2015-06-09	Update team GPG/PGP keys in Appendix A, noted new 2.0.8, 2.0.9 platforms in section 2.7
2015-04-16	Multiple typographical corrections (thanks to Mike Carden *mike.carden@au.ngc.com)*
2014-09-02	Fixed typo in Section 4.3.3, added new platforms in Section 3
2014-07-21	Reference the 2.0.6 and 2.0.7 revisions
2013-12-04	Appendix B: Updated footnote referencing special cases in fips_algvs
2013-11-01	Added Citrix acknowledgment
2013-10-31	Update URL in section 5.6 (thanks to *mscriven@sdisw.com*)
2013-09-29	Fixed typo in section 6 (thanks to *karanpopali@gmail.com*)
2013-09-13	Added Cryptsoft acknowledgment, update for 2.0.5, note effective disabling of Dual EC DRBG
2013-02-02	Documented FIPSDIR in Section 4.2
2013-01-24	Fixed issue with iOS and VALID_ARCHS vs ARCHS
2013-01-10	Clarified iOS procedures
2013-01-09	Added information on FIPS_module_mode()
2013-01-08	Spelling corrections and flow improvements
2012-12-02	Changed "vendor affirmed" references to "user affirmed"
2012-11-29	Corrections to instructions for iOS building
2012-11-01	Additions to section 6
2012-10-25	Additions to section 5.3, new Appendic E.3
2012-09-07	Added new section on GMAC
2012-07-17	Added iOS to Appendix E
2012-07-03	Correct typographical errors, update acknowledgment
2012-06-28	Update with certificate number
2012-05-15	Discussion of the new "secure installation" requirement.
2012-04-09	Updated and rename the "fips_hmac" sample application; added section 6.5
2012-03-15	Platform list and cross-reference, and additional discussion of platform issues
2012-02-21	Additional discussion of cross-compilation
2011-09-07	Initial draft for openssl-fips-2.0.tar.gz

Table of Contents

1. Introduction

This document is a guide to the use of the OpenSSL FIPS Object Module, a software component intended for use with the OpenSSL cryptographic library and toolkit. It is a companion document to the *OpenSSL FIPS 140-2 Security Policy* document submitted to NIST as part of the FIPS 140-2 validation process. It is intended as a technical reference for developers using, and system administrators installing, the OpenSSL FIPS software, for use in risk assessment reviews by security auditors, and as a summary and overview for program managers. It is intended as a guide for annotation and more detailed explanation of the *Security Policy*, and *not* as a replacement. In the event of a perceived conflict or inconsistency between this document and the *Security Policy* the latter document is authoritative as only it has been reviewed and approved by the Cryptographic Module Validation Program (CMVP), a joint U.S. - Canadian program for the validation of cryptographic products (http://csrc.nist.gov/cryptval/).

Familiarity with the OpenSSL distribution and library API (Application Programming Interface) is assumed. This document is not a tutorial on the use of OpenSSL and it only covers issues specific to the FIPS 140-2 validation. For more information on the use of OpenSSL in general see the many other sources of information such as http://openssl.org/docs/ and *Network Security with OpenSSL* (Reference 4).

The *Security Policy* document (Reference 1) is available online at the NIST Cryptographic Module Validation website, http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1747.pdf.

For more information on the OpenSSL Software Foundation see http://openssl.com/. For more information on the OpenSSL project see http://openssl.org/. For more information on NIST and the cryptographic module validation program, see http://csrc.nist.gov/cryptval/.

For information and announcements regarding current and future OpenSSL related validations see http://openssl.org/docs/fips/fipsnotes.html. That web page also has a very quick introduction extracted here:

1.1 FIPS What? Where Do I Start?

Ok, so your company needs FIPS validated cryptography to land a big sale, and your product currently uses OpenSSL. You haven't worked up the motivation to wade through the entire User Guide and want the quick "executive summary". Here is a grossly oversimplified account:

OpenSSL itself is not validated,and never will be. Instead a carefully defined software component called the OpenSSL FIPS Object Module has been created. The Module was designed for compatibility with the OpenSSL library so products using the OpenSSL library and API can be converted to use FIPS 140-2 validated cryptography with minimal effort.

The OpenSSL FIPS Object Module validation is unique among all FIPS 140-2 validations in that the product is "delivered" in source code form, meaning that if you can use it exactly as is and can build it for your platform according to a very specific set of instructions, then you can use it as validated cryptography[3].

The OpenSSL library is also unique in that you can download and use it for free.

If you require source code or build process changes for your intended application, then you cannot use the open source based validated module – you must obtain your own validation. This situation is common; see "Private Label" validation, below.

New FIPS 140-2 validations (of any type) are slow (6-12 months is typical), expensive (US$50,000 is typical for an uncomplicated validation), and unpredictable (completion dates are not only uncertain when first beginning a validation, but remain so during the process).

Note that FIPS 140-2 validation is a complicated topic that the above summary does not adequately address. You have been warned!

1.2 "Change Letter" Modifications

If the existing validated OpenSSL FIPS Object Module is *almost* what you need, but some minor modifications are necessary for your intended use, then it *may* be possible to retroactively modify the original validation to include those necessary changes. The process by which this is done is known as the "maintenance letter" or "change letter" process. A change letter can be substantially faster and less expensive than obtaining a new, independent validation.

Modifications to the FIPS module to support a new platform (operating system or compiler) are often compatible with the change letter process.

1.3 The "Private Label" Validation

The OSF would prefer to work on open source based validations which benefit the OpenSSL user community at large. However, we understand not all work can benefit the community. We refer to validations based directly on the OpenSSL FIPS Object Module but not available to the community as "private label" validations. They are also sometimes referred to as "cookie cutter" validations.

Many ISVs and vendors are interested in private label validations, and the OSF will assist in such efforts with a priced engagement. An ISV or vendor usually obtains a private label validation for marketing or risk management purposes. For example, a company may choose to privately retain its validation to ensure its competitive advantage, or a company might modify the sources and choose to keep the changes private.

[3] Either directly or via "User Affirmation" which is discussed in §5.5.

OSF has performed numerous private validations for desktop, server, and mobile platforms with very competitive pricing. Often, the pricing is less than the account setup fee for closed sourced and locked-in solution. Trivial and uncomplicated validations can often be performed using fixed rate contracts to assure cost constraints.

2. Background

For the purposes of FIPS 140-2 validation, the OpenSSL FIPS Object Module v2.0 is defined as a specific discrete unit of binary object code (the "*FIPS Object Module*") generated from a specific set and revision level of source files embedded within a source distribution. These platform portable source files are compiled to create the object code in an isolated and separate form. That object code is then used to provide a cryptographic services to external applications. The terms *FIPS Object Module* and *FIPS Module* elsewhere in this document refer to this *OpenSSL FIPS Object Module* object code.

The FIPS Object Module provides an API for invocation of FIPS approved cryptographic functions from calling applications, and is designed for use in conjunction with standard OpenSSL 1.0.1 distributions. These standard OpenSSL 1.0.1 source distributions support the original non-FIPS API as well as a *FIPS Mode* in which the FIPS approved algorithms are implemented by the FIPS Object Module and non-FIPS approved algorithms are *disabled* by default. These non-validated algorithms include, but are not limited to, Blowfish, CAST, IDEA, RC-family, and non-SHA message digest and other algorithms.

The FIPS Object Module was designed and implemented to meet FIPS 140-2, Level 1 requirements. There are no special steps required to ensure FIPS 140-2 compliant operation of the FIPS Object Module, other than building, loading, and initializing the FIPS approved and HMAC-SHA-1 digest verified source code. This process of generating the application executable object from source code for all supported platforms[1] is documented in detail at §4 and §5.

The FIPS Object Module provides confidentiality, integrity signing, and verification services. The FIPS Object Module supports the following algorithms: Triple DES, AES, CMAC, CCM, RSA (for digital signatures), DH, DSA/DSA2, ECDSA/ECDSA2, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, and HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512. The FIPS Object Module supports SP 800-90 and ANSI X9.31 compliant pseudo-random number generators.

The FIPS Object Module supports the Suite B cryptographic algorithms and can be used with Suite B cryptography exclusively. Suite B requires 128-bit security levels and forbids use of TLS lesser than 1.2 (TLS 1.0 and 1.1 use MD5 as a PRF during key agreement).

[1]By definition, for all platforms to which the validation can be extended. Per the requirements of the *Security Policy*, any change to the documented build process renders the result non-FIPS approved.

The FIPS Object Module v2.0 is similar in many respects to the earlier OpenSSL FIPS Object Module v1.2.x. The v1.2.4 was originally validated in late 2008 with validation certificate #1051; that original validation has been extended several times to incorporate additional platforms.

The v1.2.x Module is only compatible with OpenSSL 0.9.8 releases, while the v2.0 Module is compatible with OpenSSL 1.0.1 and later releases. The v2.0 Module is the best choice for all new software and product development.

2.1 Terminology

2.1.1 FIPS 140-2 Specific Terminology

During the course of multiple validations it became clear that some terminology was interpreted differently by OpenSSL developers, cryptographers, the CMVP and FIPS 140-2 specialists. In this section some of the potential confusions in terminology are discussed.

Approved Mode

The FIPS 140-2 Approved Mode of Operation is the operation of the FIPS Object Module when all requirements of the *Security Policy* have been met and the software has successfully performed the power-up and self test operation (invocation of the **`FIPS_mode_set()`** function call). In this document this Approved Mode is referred to simply as *FIPS mode*.

Crypto Officer

System administrator. The FIPS 140-2 Crypto Officer[4] is the person having the responsibility and access privileges to install, configure, and initialize the cryptographic software.

HMAC-SHA-1 digest

A HMAC-SHA-1 digest of a file using a specific HMAC key (the ASCII string "**`etaonrishdlcupfm`**"). Such digests are referred to in this document as "digests" or "fingerprints". The digests are used for integrity checking to verify that the software in question has not been modified or corrupted from the form originally used as the basis of the FIPS 140-2 validation.

Note that the PGP or GPG signatures traditionally used to check the integrity of open source software distributions are *not* a component of any of the FIPS 140-2 integrity checks.

Module

[4]The term "Officer" does not imply a requirement for a military or government official, although some military or government organizations may choose to restrict the performance of this system administration role to certain official capacities.

The concept of the cryptographic module is important for FIPS 140-2, and it has subtle nuances in this context. Conceptually the Module is the binary object code and data in the FIPS Object Module for a running process.

The "cryptographic module" is often referred to simply as "module". That term is capitalized in this document as a reminder that it has a somewhat different meaning than assumed by software developers outside of a FIPS 140-2 context.

Note that traditionally the executable (or shared library) file on disk corresponding to this Module as a running process is also considered to be a Module[5] by the CMVP. An integrity check of the entire executable file on disk prior to memory mapping is considered acceptable as long as that executable file does not contain any extraneous[6] software. In this traditional case the specific executable file is submitted for testing and thus the precise content (as a bit string) is known in advance.

In the case of the FIPS Object Module only source code is submitted for validation testing, so the bit string value of the binary object code in memory cannot be known in advance. A chain of checks beginning with the source code and extending through each step in the transformation of the source code into a running process was established to provide a check equivalent to that used by more traditional object based validations.

The chain of checks works backwards from the software as resident in memory for a process to the executable program file from which the process was created (the existing precedent), then to the FIPS Object Module used to link the program file, and finally to the original source files used to create the FIPS Object Module. Each of those stages can be thought of as antecedents of the Module, and the integrity of each needs to be verified to assure the integrity of the Module.

2.1.2 General Glossary

ABI	Application Binary Interface
AES	Advanced Encryption Standard
AES-NI	AES New Instructions
ARM	a processor instruction set architecture developed by ARM Holdings
API	Application Programming Interface
Blowfish	A cryptographic algorithm not allowed in FIPS mode
CAST	A cryptographic algorithm not allowed in FIPS mode
CC	Common Criteria

[5]Presumably because the transformations of the disk resident file contents performed by the runtime loader are considered to be well understood and sufficiently minimal.
[6]The definition of what constitutes "extraneous" is not formally specified and subject to interpretation.

CCM	Combined Cipher Machine, a mode of operation for cryptographic block ciphers
CDH	Cofactor Diffie-Hellman, a Discrete Logarithm Cryptography (DLC) primitive, see SP 800-56A
CAVP	Cryptographic Algorithm Validation Program, see http://csrc.nist.gov/groups/STM/cavp/
CMAC	Cipher-based MAC, a block cipher-based message authentication code algorithm
CMVP	Cryptographic Module Validation Program, see http://csrc.nist.gov/groups/STM/cmvp/
CTR	DRBG flavor
DH	Diffie-Hellman, a FIPS approved cryptographic algorithm
DLL	Dynamic Link Library, a shared library for the Microsoft Windows OS
DRBG	Deterministic Random Bit Generator, see SP 800-90
DSA	Digital Signature Algorithm, a FIPS approved cryptographic hash function
DSA2	DSA as defined in FIPS 186-3
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography (see EC)
ECDH	Elliptic Curve Diffie–Hellman, a variant of Diffie–Hellman used as an anonymous key agreement protocol
ECDSA	Elliptic Curve Digital Signature Algorithm, a variant of DSA which uses ECC
ECDSA2	ECDSA as defined in FIPS 186-3
ELF	Executable and Linkable Format, the standard binary file format for Unix-like systems on x86
ENGINE	An OpenSSL mechanism for interfacing with external cryptographic implementations
EVP	ENVelope encryption, an OpenSSL API that provides a high-level interface to cryptographic functions
FIPS	Federal Information Processing Standards, see http://www.itl.nist.gov/fipspubs/
FIPS 140-2	See http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

FIPS Object Module	the special monolithic object module built from the special source distribution[7] identified in the *Security Policy*
GCM	Galois/Counter Mode, a mode of operation for symmetric key cryptographic block ciphers
GPG	See PGP
GUI	Graphical User Interface
HMAC	Hash Message Authentication Code, a mechanism for message authentication using cryptographic hash functions
IA	Information Assurance
IDEA	A cryptographic algorithm not allowed in FIPS mode
IKE	Internet Key Exchange, a protocol for exchanging information required for secure communication.
IP	Internet Protocol, a network communications protocol
IPsec	Internet Protocol Security, a protocol suite for securing IP communications by authenticating and encrypting each IP packet
IT	Information Technology
IUT	Implementation Under Test
KAT	Known Answer Test
MASM	The Microsoft assembler, no longer supported by OpenSSL
MD2	A cryptographic algorithm not allowed in FIPS mode
NEON	an architecture extension for ARM Cortex™-A series processors,
NASM	the open source Netwide ASseMbler, see http://www.nasm.us/
NID	Name IDentifier for extracting information from a certificate Distinguished Name.
NIST	National Institute of Science and Technology, see http://www.nist.gov/
OE	See Operational Environment
Operational Environment	The FIPS 140-2 term for "platform"
OS	Operating System
OSF	The OpenSSL Software Foundation
PCLMULQDQ	an instruction for x86 processors which performs carry-less multiplication of two 64-bit operands
PGP	Pretty Good Privacy, an encrypted E-mail program

[7] Roughly speaking, this special source distribution was created from the **`OpenSSL-fips-2_0-stable`** branch in the CVS source code repository with the command **`make VERSION=fips-2.0 TARFILE=openssl-fips-2.0.tar -f Makefile.fips dist`**.

PKCS#1	Public-Key Cryptography Standard #1
PKCS#3	Public-Key Cryptography Standard #3
POST	Power Up Self Test, an initialization process required by FIPS 140-2
PRNG	Pseudo-Random Number Generator
RNG	Random Number Generator
PSS	Probabilistic Signature Scheme, a provably secure way of creating signatures with RSA
RSA	Rivest-Shamir-Adleman, a public key cryptographic algorithm
SHA	Secure Hash Algorithm, a cryptographic hash function
SSE2	Streaming SIMD Extension 2, an extension of the x86 instruction set
SSH	Secure SHell, a network protocol for secure data communication
SSL	Secure Socket Layer, a predecessor to the TLS protocol
SSSE3	Supplemental Streaming SIMD Extensions 3, an extension of the x86 instruction set
Suite B	a set of cryptographic algorithms created by the National Security Agency
TLS	Transport Layer Security, a cryptographic protocol providing communication security over IP connections
VMS	Virtual Memory System, an operating system that runs on VAX, Alpha and Itanium-based families of computers (now obsolete)
x86	a family of instruction set architectures originally defined by Intel
XTS	XEX Tweakable Block Cipher with Ciphertext Stealing
XTS-AES	a cryptographic algorithm specified in SP 800-38E

2.2 The FIPS Module and Integrity Test

The FIPS Object Module is generated in binary file format, with an embedded pre-calculated HMAC-SHA-1 digest covering the module[8] as it is loaded into application address space. The Module integrity check consists of recalculating that digest from the memory areas and comparing it to the embedded value which resides in an area not included in the calculated digest[9]. This “in-core hashing” integrity test is designed to be both executable format independent and fail-safe.

[8]Specifically, the text and read-only data segments which constitute the initialized components of the module.
[9]If the digest value resided in the data area included in the calculation of that digest, the calculated value of the digest would itself be an input into that calculation.

For this scenario the Module is the text and data segments as mapped into memory for the running application.

The term Module is also used, less accurately, to designate the antecedent of that memory mapped code and data, the FIPS Object Module file residing on disk.

The FIPS Object Module is generated from source code, so the integrity of that source must also be verified. The single runtime digest check typical of pre-built binary files is replaced by a chain of digest checks in order to validate that the running code was in fact generated from the original source code. As before the term Module properly designates the text and data segments mapped into memory, but is also more loosely used to reference several levels of antecedents. These levels are discussed below.

2.3 The FIPS Integrity Test

The FIPS 140-2 standard requires an integrity test of the Module to verify its integrity at initialization. In addition to the requirement that the integrity test validate that the FIPS Object Module code and data have not changed, two additional implicit requirements for the integrity test were identified during the validation process.

2.3.1 Requirement for Exclusive Integrity Test

An integrity test that is merely guaranteed to fail if any of the cryptographic module software changes is not sufficient. It is also necessary that the integrity test *not* fail if the cryptographic module software is not directly corrupted, even though the application referencing the cryptographic module may be damaged with unpredictable consequences for the correct functioning of that application. Another way of looking at this is that as application failures are out of scope of the integrity test there needs to be some level of assurance that changes to application software do not affect the cryptographic module integrity test[10].

This requirement is met with an in-core integrity test that carefully excludes any extraneous[11] object code from the digest calculation and verification.

2.3.2 Requirement for Fixed Object Code Order

The relative order of all object code components within the module must be fixed and invariant. The usual linking process does not care about the relative order of individual object modules, e.g. both

```
gcc –o runfile alpha.o beta.o gamma.o
```

[10]This assurance was given by showing during testing that corruption of code or data outside of the memory area containing the FIPS Object Module did not result in an integrity test failure.
[11]The definition of what constitutes "extraneous" is not formally specified and thus subject to interpretation.

and

```
gcc –o runfile beta.o alpha.o gamma.o
```

produce functionally identical executable files. Likewise, the order of object modules in a static link library is irrelevant:

```
ar r libxxx.a alpha.o beta.o gamma.o
```

and

```
ar r libxxx.a beta.o alpha.o gamma.o
```

produce interchangeable link libraries, and a given application may not incorporate all of the object modules contained with the link library when resolving references. For the FIPS Object Module it was required that any such omission or rearrangement of the Module object modules during the application creation process not occur. This requirement is satisfied by simply compiling all the source code into a single monolithic object module:

```
ld –r –o fipscanister.o fips_start.o ... fips_end.o
```

with all the object modules between the `fips_start.o` and `fips_end.o` modules that define the low and high boundaries of a monolithic object module. All subsequent reference to this monolithic object module will preserve the relative order, and presence, of the original object code components.

2.4 The File Integrity Chain

Most validated products consisting of a pre-built binary executable implement the module integrity check as a digest check over portions of that executable file or the corresponding memory mapped image. For the FIPS Object Module the module integrity check instead takes the form of a chain of digest checks beginning with the source files used for the CMVP validation testing. Note that while this chain of checks is more complex, it provides much more visibility for independent verification compared to the case of validated pre-built binary executables. With the FIPS Object Module the prospective user can independently verify that the runtime executable does indeed directly derive from the same source that was the basis of the validation.

2.4.1 Source File (Build Time) Integrity

“Build time” is when the FIPS Object Module is created from the OpenSSL FIPS source distribution, in accordance with the *Security Policy*.

The first file integrity check occurs at build time when the HMAC-SHA-1 digest of the distribution file is calculated and compared to the stored value published in the *Security Policy (Appendix B)*.

Because the source files reside in this specific distribution and cannot be modified these source files are referred to as *sequestered* files.

Note that a means to calculate the HMAC-SHA-1 digest is required in order to perform this integrity check. A "bootstrap" standalone HMAC-SHA-1 utility, **`fips_standalone_sha1`**, is included in the distribution. This utility is generated first before the sequestered files are compiled in order to perform the integrity check. Appendix C gives an example of an equivalent utility.

2.4.2 Object Module (Link Time) Integrity

"Link time" is when the application is linked with the previously built and installed FIPS Object Module to generate an executable program.

The build process described in the *Security Policy* results in the creation of an object module, **`fipscanister.o`**, and a matching digest file, **`fipscanister.o.sha1`**. This FIPS Object Module contains the object code corresponding to the sequestered source files (object code for FIPS specific functions such as **`FIPS_mode_set()`** and for the algorithm implementations).

The link time integrity check occurs when the FIPS Object Module is used to create an application executable object (binary executable or shared library). The digest stored in the installed file **`fipscanister.o.sha1`** must match the digest calculated for the **`fipscanister.o`** file.

Note that except in the most unusual circumstances the FIPS Object Module itself (**`fipscanister.o`**) is not linked directly with application code. Instead the FIPS Object Module is embedded in the OpenSSL libcrypto library (libcrypto.a/libcrypto.so) which is then referenced in the usual way by the application code. That combination is known as a "FIPS capable" OpenSSL library and is discussed in more detail in section 2.5.

2.4.3 Application Executable Object (Run Time) Integrity

Application "run time" occurs when the previously built and installed application program is invoked. Unlike the previous step this invocation is usually performed repeatedly.

The runtime integrity check occurs when the application attempts to enable FIPS mode via the **`FIPS_mode_set()`** function call. The digest embedded within the object code from **`fipscanister.o`** must match the digest calculated for the memory mapped text and data areas.

2.5 Relationship to the OpenSSL API

The FIPS Object Module is designed for indirect use via the OpenSSL API. Applications linked with the "FIPS capable" OpenSSL libraries can use both the FIPS validated cryptographic functions of the FIPS Object Module and the high level functions of OpenSSL. The FIPS Object Module should not be confused with OpenSSL library and toolkit or any specific official OpenSSL distribution release.

A version of the OpenSSL product that is suitable for use with the FIPS Object Module is a *FIPS Compatible OpenSSL*.

When the FIPS Object Module and a FIPS compatible OpenSSL are separately built and installed on a system, with the FIPS Object Module embedded within the OpenSSL library as part of the OpenSSL build process, the combination is referred to as a *FIPS capable OpenSSL*.

Summary of definitions
The *FIPS Object Module* is the FIPS 140-2 validated module described in the *Security Policy*
A *FIPS compatible OpenSSL* is a version of the OpenSSL product that is designed for compatibility with the FIPS Object Module API
A *FIPS capable OpenSSL* is the combination of the separately installed *FIPS Object Module* along with a *FIPS compatible OpenSSL*.

Table 2.5

The OpenSSL libraries, when built from a standard OpenSSL distribution with the "**`fips`**" configuration option for use with the FIPS Object Module, will contain the usual non-FIPS algorithms and non-cryptographic supporting functions, and the non-FIPS algorithm disabling restrictions.

Note that use of individual object modules comprising the monolithic FIPS Object Module is specifically forbidden by FIPS 140-2 and the CMVP[12]. In the absence of that restriction the individual object modules would just be incorporated directly in the OpenSSL **`libcrypto.a`** library. The monolithic FIPS Object Module must be used in its entirely and cannot be edited to accommodate size constraints.

Various non-FIPS algorithms such as Blowfish, IDEA, CAST, MD2, etc. are included in the OpenSSL libraries (depending on the **`./config`** options specified in addition to **`fips`**). For applications that do not utilize FIPS 140-2 cryptography, the resulting libraries are drop-in compatible with the libraries generated without the **`fips`** option (a deliberate design decision to encourage wider availability and use of FIPS 140-2 validated algorithms). The converse is not true:

[12]Actually, to encourage use of **`fipscanister.o`** even in non-FIPS mode applications, a copy is incorporated into **`libcrypto.a`**, but special care is taken to preclude its usage in FIPS enabled applications. The *fipsld* utility provided in the FIPS compatible OpenSSL distributions prevents that usage as follows. In static link context that is achieved by referencing the official **`fipscanister.o`** first on the command line., and in dynamic link context by temporarily removing it from **`libcrypto.a`**. This removal is necessary because dynamic linking is commonly accompanied by **`-whole-archive`**, which would force both copies of **`fipscanister.o`** into the shared library. Note the integrity check is designed as a failsafe precaution in the event of link errors -- even if two copies are included into the application in error, the integrity check will prevent the use of one copy for the integrity test and the other for the actual implementation of cryptography. In other words, if both the official **`fipscanister.o`** and the unvalidated version that is embedded in **`libcrypto.a`** both end up in an executable binary, and if **`FIPS_mode_set()`** returns success, the unvalidated copy will not be used for cryptography.

a non-FIPS OpenSSL library cannot be substituted for the FIPS Compatible library because the FIPS specific function calls will not be present (such as **FIPS_mode_set()**).

2.6 FIPS Mode of Operation

Applications that utilize FIPS mode must call the **FIPS_mode_set()** function. After successful FIPS mode initialization, the non-FIPS algorithms will be disabled by default.
The FIPS Object Module together with a compatible version of the OpenSSL product can be used in the generation of both FIPS mode and conventional applications. In this sense, the combination of the FIPS Object Module and the usual OpenSSL libraries constitutes a “FIPS capable API”, and provide both FIP approved algorithms and non-FIPS algorithms.

2.6.1 FIPS Mode Initialization

Only one initialization call, **FIPS_mode_set()**, is required to operate the FIPS Object Module in a FIPS 140-2 Approved mode, referred to herein as "FIPS mode". When the FIPS Object Module is in FIPS mode all security functions and cryptographic algorithms are performed in Approved mode. Use of the **FIPS_mode_set()** function call is described in §5.

A power-up self-test is performed automatically by the **FIPS_mode_set()** call, or optionally at any time by the **FIPS_selftest()** call (see Appendix D). If any power-up self-test fails the internal global error flag **FIPS_selftest_fail** is set and subsequently tested to prevent invocation of any cryptographic function calls.

The internal global flag **FIPS_mode** is set to FALSE indicating 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 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 FIPS Object Module is in FIPS mode.

2.6.2 Algorithms Available in FIPS Mode

Only the algorithms listed in tables 4a and 4b of the Security Policy are allowed in FIPS mode. Note that Diffie-Hellman and RSA are *allowed* in FIPS mode for key agreement and key establishment even though they are “Non-Approved” for that purpose. RSA for sign and verify is “Approved” and hence also allowed, along with all the other Approved algorithms listed in that table.

The OpenSSL library attempts to disable non-FIPS algorithms. when in FIPS mode. The disabling occurs on the **EVP_*** APIs and most low level function calls. Failure to check the return code from low level functions could result in unexpected behavior. Note also that sufficiently creative or unusual use of the API may still allow the use of non-FIPS algorithms. The non-FIPS algorithm

disabling is intended as an aid to the developer in preventing the accidental use of non-FIPS algorithms in FIPS mode, and not as an absolute guarantee. It is the responsibility of the application developer to ensure that only FIPS algorithms are used when in FIPS mode.

OpenSSL provides mechanisms for interfacing with external cryptographic devices, such as accelerator cards, via “ENGINES.” This mechanism is not disabled in FIPS mode. In general, if a FIPS validated cryptographic device is used with OpenSSL in FIPS mode so that all cryptographic operations are performed either by the device or the FIPS Object Module, then the result is still FIPS validated cryptography.

However, if any cryptographic operations are performed by a non-FIPS validated device, the result is use of non-validated cryptography. It is the responsibility of the application developer to ensure that ENGINES used during FIPS mode of operation are also FIPS validated.

2.7 Revisions of the 2.0 Module

Existing FIPS 140-2 validations can be retroactively modified, within defined limits, via the "maintenance letter" or "change letter" process. Change letter modifications are typically done to correct minor "non-cryptographically significant" bugs or, most commonly, to add support for new platforms. Change letter actions are usually less expensive and faster than a full validation; and are an attractive option to the software vendor desiring to use the FIPS module for a platform not currently covered by the validation.

Several change letter modifications were in process prior to the formal award of the initial OpenSSL FIPS Object Module v2.0 validation. More change letters are anticipated over the lifetime of the validation. For all past validations we have always been careful to introduce any changes in a way that will not impact any previously tested platforms, so that the most recent revision of the module can be used for new deployments on any platform.

The history of new revisions include:

2.0.1 Addition of Apple iOS 5.1 on ARMv7
2.0.1 Addition of WinCE 5.0 on ARMv7
2.0.1 Addition of Linux 2.6 on PowerPC32-e500 (PPC)
2.0.1 Addition of DSP Media Framework 1.4 on TI C64x+
2.0.1 Addition of WinCE 6.0 on ARMv7
2.0.1 Addition of Android 4.0 on OMAP 3 (ARMv7)
2.0.2 Addition of NetBSD 5.1 on PowerPC32-e500 (PPC)
2.0.2 Addition of NetBSD 5.1 on Intel Xeon 5500 (x86)
2.0.3 Addition of Win2008 on Xeon E3-1220v2 (x86)
2.0.3 Addition of RHEL 32/64 bit on Xeon E3-1220v2 (x86) under vSphere
2.0.3 Addition of Win7 on Intel Core i5-2430M (x86) with AES-NI
2.0.3 Addition of Android 4.1/4.2 on Nvidia Tegra 3 (ARMv7) with/without NEON

2.0.3 Addition of WinEC7 on Freescale i.MX53xD (ARMv7) with/without NEON
2.0.3 Addition of Android 4.0 on Qualcomm Snapdragon APQ8060 (ARMv7)
2.0.3 Addition of VMware Horizon Module on Qualcomm MSM8X60 (ARMv7)
2.0.3 Addition of Apple OS X 10.7 on Intel Core i7-3615QM (x86)
2.0.3 Addition of Apple iOS 5.0 on ARM Cortex A8 (ARMv7)
2.0.4 Addition of OpenWRT 2.6 on MIPS 24Kc
2.0.5 Addition of QNX 6.4 on Freescale i.MX25 (ARMv4)
2.0.5 Addition of Apple iOS 6.1 on Apple A6X SoC (ARMv7s)
2.0.5 Addition of eCos 3 on Freescale i.MX27 926ejs (ARMv5TEJ)
2.0.5 Addition of VMware Horizon Workspace 1.5 under vSphere on Intel Xeon E3-1220 (x86) with/without AES-NI
2.0.5 Addition of Ubuntu 13.04 on AM335x Cortex-A8 (ARMv7) with/without NEON
2.0.5 Addition of Linux 3.8 on ARM926 (ARMv5TEJ)
2.0.5 Addition of Linux 3.4 under Citrix XenServer on Intel Xeon E5-2430L (x86) with/without AES-NI
2.0.5 Addition of Linux 3.4 under VMware ESX on Intel Xeon E5-2430L (x86) with/without AES-NI
2.0.5 Addition of Linux 3.4 under Microsoft Hyper-V on Intel Xeon E5-2430L (x86) with/without AES-NI
2.0.5 Addition of Apple iOS 6.0 on Apple A5 / ARM Cortex-A9 with/without NEON
2.0.6 Removal of Dual EC DRBG (no platforms)
2.0.7 Addition of Linux 2.6 on Freescale e500v2 (PPC)
2.0.7 Addition of AcanOS 1.0 on Intel Core i7-3612QE (x86)
2.0.7 Addition of AcanOS 1.0 on Intel Core i7-3612QE (x86) with AES-NI
2.0.7 Addition of AcanOS 1.0 on Feroceon 88FR131 (ARMv5)
2.0.7 Addition of FreeBSD 8.4 on Intel Xeon E5440 (x86)
2.0.7 Addition of FreeBSD 9.1 on Xeon E5-2430L (x86)
2.0.7 Addition of FreeBSD 9.1 on Xeon E5-2430L (x86) with AES-NI
2.0.7 Addition of ArbOS 5.3 on Xeon E5645 (x86)
2.0.7 Addition of Linux ORACLESP 2.6 on ASPEED AST2100 (ARMv5)
2.0.7 Addition of Linux ORACLESP 2.6 on ServerEngines PILOT3 (ARMv5)
2.0.8 Addition of Linux ORACLESP 2.6 on ASPEED AST-Series (ARMv5)
2.0.8 Addition of Linux ORACLESP 2.6 on Emulex PILOT 3 (ARMv5)
2.0.8 Addition of FreeBSD 9.2 on Xeon E5-2430L (x86) with-without AES-NI
2.0.8 Addition of FreeBSD 10.0 on Xeon E5-2430L (x86) with/without AES-NI
2.0.8 Addition of FreeBSD 8.4 32-bit on Xeon E5440 (x86)
2.0.9 Addition of VMware Horizon Workspace 2.1 x86 under vSphere ESXi 5.5 on Intel Xeon E3-1220 (x86) with/without AES-NI
2.0.9 Addition of QNX 6.5 on ARMv4 Freescale i.MX25 (ARMv4)
2.0.9 Addition of Apple iOS 7.1 64-bit on ARMv8 Apple A7 (ARMv8) with/without NEON
2.0.9 Addition of TS-Linux 2.4 on ARMv4

Revisions 2.0.6 and 2.0.7 constitute an unfortunate perversity. The 2.0.6 revision removed the Dual EC DRBG implementation which at the time of submission of the official paperwork (Maintenance Letter) on January 20, 2014 had already been officially repudiated by NIST. However, approval of the 2.0.6 revision languished for more than six months. In the meantime eleven[13] new platforms were tested using the most recent officially approved revision, 2.0.5, plus platform specific modifications, resulting in revision 2.0.7 which still included the Dual EC DRBG revision[14]. The official paperwork for the 2.0.7 revision was submitted months after 2.0.6 but both revisions were approved with the span of a single week, with the preverse result that the 2.0.7 revision of the OpenSSL FIPS Object Module still contained the deprecated and disgraced Dual EC DRBG. It was again (and permanently) removed with revision 2.0.8.

2.8 Prior FIPS Object Modules

The 2.0 version of the FIPS Object Module is the latest in a series of open source based validated modules derived from the OpenSSL product. As with those prior modules this version is delivered in source code form and results in a statically linked object module.

There are some differences with respect to the previous version 1.2.x series of modules which have been widely used, both directly as validated for certificate #1051, and indirectly as models for separate "private label" validation. Some of the key differences are:

1. The source code distribution for the 1.2.x FIPS modules was a modified OpenSSL distribution that contained a considerable amount of code superfluous to the generation of the FIPS module. The 2.0 FIPS module is provided in a separate dedicated source distribution containing far less extraneous code.

2. The 1.2.x FIPS modules were compatible only with the "FIPS capable" 0.9.8 baseline. The 2.0 FIPS module is compatible with the "FIPS capable" 1.0.1 baseline, and will probably remain usable with future OpenSSL versions (1.1.0 and later).

3. The 2.0 FIPS module has a significantly faster POST performance. The slow POST for the 1.2.x modules was a significant impediment to use on some low-powered processors.

4. The 2.0 FIPS module contains several additional cryptographic algorithms, including all of Suite B.

[13]Only ten new platforms acually appeared with the 2.0.7 revision due to an unexplained "paperwork error" at the CAVP which required repeating some of the algorithm tests for the eleventh platform which was thus omitted from the 2.0.7 revision. The eleventh platform will be included in a future revision.

[14]Approval of the removal of Dual EC DRBG implementation was far from certain; several interested parties including one accredited test lab were absolutely certain it would not be permitted. While that issue was pending we did not want to put the eleven new platforms at risk by testing on a revision that omitted Dual EC DRBG. As it was the unfortunate sponsors of those new platforms had to wait up to six months for final official approval.

5. The 2.0 FIPS module more directly accommodates cross-compilation, as both native and cross-compilation now use the same technique for determining the module integrity digest at build time.

2.9 Future FIPS Object Modules

The open source based OpenSSL FIPS Object Module validations are difficult and expensive, and as a result have been done infrequently. The long intervals between validations compound the difficulty of obtaining each new validation:

1. The companion OpenSSL product changes significantly, requiring significant rework to both that product and the new FIPS module for the "FIPS capable" functionality;

2. A number of new and relatively untried algorithm tests are introduced by the CAVP;

3. New validation requirements are introduced by the CMVP.

The result is a vicious cycle: the new validation takes much more effort and time, during which these factors continue to mount (the CMVP can and does introduce new requirements in the course of an ongoing validation). That cost and difficulty becomes an intimidating factor for planning, and soliciting funding and/or collaboration for, the next validation.

In order to try and bypass this cycle the OSF would like to perform open source based validations more frequently, ideally as often as the interval required to obtain a validation which is about a year. That would mean that at any point in time there will be a relatively current completed validation and a new validation in process. New features or modifications that would adversely impact the ongoing validation can then be deferred to the next upcoming one. New requirements and algorithm tests can be addressed a few at a time instead of all at once in a huge onslaught.

Potential sponsors of such an effort are welcome, and are invited to contact OSF to express their interest.

3. Compatible Platforms

The FIPS Object Module is designed to run on a wide range of hardware and software platforms. Any computing platform that meets the conditions in the *Security Policy* can be used to host a FIPS 140-2 validated FIPS Object Module provided that module is generated in accordance with the *Security Policy*.

At the time the *OpenSSL FIPS Object Module v2.0* was developed, all Unix®[15]-like environments supported by the full OpenSSL distribution were also supported by the FIPS validated source files included in the FIPS Object Module. However, successful compilation of the FIPS Object Module for all such platforms was not verified. If any platform specific compilation errors occur that can only be corrected by modification of the FIPS distribution files (see Appendix B of the *Security Policy*), then the FIPS Object Module will not be validated for that platform.

It is also noted that a platform which is currently supported (but untested) may not be supported in the future as revisions are made to the FIPS validated sources. For example, a change made for one platform may adversely affect another, untested platform.

By default, the FIPS Object Module software utilizes assembly language optimizations for some supported platforms. Currently assembler language code residing within the cryptographic module boundary is used for the x86/Intel®[16] ELF and ARM®[17] machine architectures. The FIPS Object Module build process will automatically select and include these assembly routines by default when building on a x86 platform. The assembly language code was included in the validation testing, so a FIPS Object Module built using the x86/Intel® assembly language routines will result in a FIPS 140-2 validated Object Module. Assembly Language and Optimizations are discussed in detail in Section 3.2.3 Assembler Optimizations.

3.1 Build Environment Requirements

The platform portability of the FIPS Object Module source code is contingent on several basic assumptions about the build environment:

1. The environment is either a) "Unix®-like" with a **`make`** command and a **`ld`** command with a "**`-r`**" (or "**`-i`**") option, or Microsoft Windows.

   Creation of the monolithic FIPS Object Module **`fipscanister.o`** requires a linker capable of merging several object modules into one. This requirement is known to be a problem with VMS and some older versions of **`LD.EXE`** under Windows®.

[15]UNIX is a registered trademark of The Open Group
[16]Intel is a registered trademark of the Intel Corporation
[17]ARM is a trademark of ARM Limited.

2. The compiler is required to place variables declared with the **`const`** qualifier in a read-only segment. This behavior is true of almost all modern compilers. If the compiler fails to do so the condition will be detected at run-time and the in-core hashing integrity check will fail.

3. The platform supports execution of compiled code on the build system (i.e. build host and target are binary compatible); or an appropriate "incore" utility is available to calculate the digest from the on-disk resident object code. See further discussion of cross-compilation in §3.4.

4. Cross-compilation uses a technique for determining the integrity check digest that may not work for all cross-compilation environments, so each such new environment must be analyzed for suitability. See further discussion of cross-compilation in §3.4.

3.2 Known Supported Platforms

The generation of a monolithic object module and the in-core hashing integrity test have been verified to work with both static and shared builds on the following platforms (note the **`./config "shared"`** option is forbidden by the terms of the validation when building a FIPS validated module, but the **`fipscanister.o`** object module can be used in a shared library[18]). Note a successful build of the FIPS module may be possible on other platforms; only the following were explicitly tested as of the date this document was last updated:

- Android®[19] on ARMv7[20] 32 bit
- Android® on ARMv7 with NEON 32 bit
- HP-UX®[21], on IA64 with 32 and 64 bit
- Linux®[22] on ARMv6, ARMv7 32 bit
- Linux on x86-64 32 and 64 bit
- Linux on x86-64 32 with SSE2 and 64 bit
- Linux on x86-64 with AES-NI 32 and 64 bit
- Linux on PowerPC®[23]
- Solaris®[24] on x86-64 with 32 and 64 bit
- Solaris® on SPARCv9[25] with 32 and 64 bit
- Solaris® on x86-64 with SSE2 32 and 64 bit
- Windows® on x86-64 with SSE2 32 and 64 bit

[18]A convenient way of generating a shared library containing **`fipscanister.o`** is discussed in Appendix B
[19]Android is a trademark of Google Inc.
[20]ARM, is a trademark or registered trademark of ARM Ltd or its subsidiaries.
[21]HP-UX is a registered trademark of Hewlett-Packard Company.
[22]Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
[23]PowerPC is a trademark of International Business Machines Corporation in the United States, other countries, or both.
[24]Solaris is a registered trademark of Oracle and/or its affiliates.
[25]SPARC® is a registered trademark of SPARC International, Inc.

- uClinux®[26] on ARMv4
- VxWorks®[27] on MIPS®[28]
- DSP Media Framework 1.4 on TI®[29] C64x+
- Apple®[30] iOS® on ARMv7
- Windows CE on ARMv7
- NetBSD[31] on PowerPC
- NetBSD on x86-64

Among the platforms known to not be supported are Windows on x86-64 with AES-NI, VMS®[32], Mac OS X®[33].

Platform Cross Reference

Operating System → / Processor ↓	Android 2.2, 4.0	HP-UX 11i	Linux 2.6	Solaris 10	Solaris 11	Windows 7	uCLinux 0.9	VxWorks 6.8	Windows CE	NetBSD							
Apple A6 (ARMv7 and ARMv7s)																	
Apple A5 (ARMv6 and ARMv7)																	
ARMv4										✔							
ARMv6														✔			

[26]uClinux is a registered trademark of Arcturus Networks Inc.
[27]VxWorks is a registered trademarks of Wind River Systems, Inc.
[28]MIPS is a trademark or registered trademark of MIPS Technologies, Inc. in the United States and other countries.
[29]TI is a registered trademark of Texas Instruments Incorporated
[30]Apple and iOS are registered trademarks of Apple Inc.
[31]NetBSD® is a registered trademark of The NetBSD Foundation, Inc.
[32]VMS is a registered trademark of Digital Equipment Corporation.
[33]Mac OS X is a registered trademark of Apple, Inc.

Platform Cross Reference															
ARMv7													✔		✔
ARMv7 NEON															✔
IA64 32 bit														✔	
IA64 64 bit														✔	
MIPS								✔							
PowerPC													✔		
SPARCv9 32 bit											✔	✔			
SPARCv9 64 bit											✔	✔			
x86-64 32 bit													✔		
x86-64 64 bit													✔		
x86-64 SSE2 32 bit										✔	✔		✔		
x86-64 SSE2 64 bit										✔	✔		✔		
x86-64 AES-NI 32 bit											✔		✔		
x86-64 AES-NI 64 bit											✔		✔		

Table 3.2

A commonly asked question is "does this validation extend to my specific platform X"? For instance: “is use of the Module validated on CentOS x86-64 when CentOS was not formally tested but Fedora was?” Or “is use with Linux kernel 2.6.35 validated when only 2.6.33 was formally tested?” Unfortunately there is no hard and fast answer to such questions.

Based on extensive discussions over the years we have developed some informal rules of thumb to determine when a given target platform corresponds with a formally tested platform (Operational Environment)

> ***Important Disclaimer***
>
> *Only the CMVP can provide authoritative answers to questions about FIPS 140-2. The following discussion represents the un-enlightened and non-authoritative opinions of persons and institutions lacking any official standing to interpret the meaning or intent of FIPS 140-2 or the validation process. CMVP guidance always takes precedence over any statements in this document.*

Rules of thumb:

1. Does the target system "code path" (see following section) correspond with that of a formally tested platform?

2. Do any run-time selectable optimizations (see section §3.2.3) correspond with those of a formally tested platform?

3. Will a binary module that builds and runs on one of the formally tested platforms (or was built on the build-time system for a formally tested cross-compiled platform) run as-is on the target system?

4. Does the processor "core" (ARMv6 versus ARMv7, for instance) correspond to that of a formally tested platform? Here the consideration is ABI compatibility -- two processors which can interchangeably execute the same set of machine instructions are effectively equivalent.

5. Does the "major" OS version (e.g. Solaris 10 versus Solaris 11) correspond to that of a formally tested platform? The "major" version is generally taken to be the full revision label for OS's using only one or two "dot" levels (e.g., Android 2.2 or Solaris 10, 11), and the first two "dot" levels for OS's using more than two "dot" levels (e.g., Linux 2.6.37, uCLinux 0.9.29)[34].

If the answer to all of these questions is "yes" then -- in general -- the prospective target platform can in general be reasonably considered as equivalent to a formally tested platform.

Arguments based on apparent "common sense" considerations should be used cautiously where FIPS 140-2 is concerned, but where general purpose validated software modules are concerned a little thought shows that strict insistence on an exact match between target platforms and formally tested Operational Environments would make it effectively impossible to widely deploy validated software through most enterprises. For instance, one of the formally tested platforms was "Android 2.2.20.A995" on an "ARMv7 rev 2 v71" processor. If a formally tested platform had to correspond at that level of detail then provision of validated modules would be very difficult, as the extensive amount of time required to obtain a FIPS 140-2 validation means that the specific platform used for testing will be updated or obsolete by the time the validation is completed.

The role of the compiler used for building the validated Module has never been fully delineated. The general – and unofficial – consensus of the FIPS 140-2 user and test lab communities appears to be that the precise version of the compiler need *not* correspond exactly with that used for the generation of the formally tested Module (for instance, gcc 4.4.1 versus 4.4.7).

If a review determines that no formally tested platform corresponds to the target platform of interest, there are several options:

[34]Note this rule of thumb has implications for the recent and more or less arbitrary jump of the Linux kernel version number from 2.6.x to 3.0.x.

1. Vendor or user "affirmation" per section G.5 of the Implementation Guidance document (Reference 3). This topic is discussed in more detail in §5.5.

2. A "change letter" modification to extend an existing validation to include the platform of interest. The change letter process can often be performed in a few weeks with a price tag in the low five figures, as opposed to the many months and high five figure to low six figure price tag of a conventional full validation.

3. A full validation leveraging the source code and documentation from the OpenSSL FIPS Object Module validation. Such a "private label" validation will still take many months but is typically much less expensive than an unrelated validation. An advantage of the "private label" validation is that upon formally engaging an accredited test lab the vendor becomes eligible[35] to have the prospective module listed on the "Modules In Process" list[36] (http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140InProcess.pdf). The presence of a vendor module on that list is a sufficient condition for completion of many procurement actions in the U.S. Department of Defense and federal government.

3.2.1 Code Paths and Command Sets

For the purposes of the validation testing a “platform” is a unique combination of source code and the specific build-time options used to turn that source code into binary code. The build-time inclusion of assembler optimizations effectively changes the source code, and source code selections vary based on the target architecture word size of 32 or 64 bits.

Due to budget and schedule constraints only some assembler optimizations for ARM and x86-64 were tested, so only those optimizations are available for building the FIPS Object Module. Two separate sets of source code were identified to cover plain C (no assembler) for x86-64 Linux 32 and 64 bits.

Even though the same source code is used for both Linux/Unix and Windows operating systems, the build instructions are sufficiently unique to each of the two OS families that the decision was made to test each code path for both OS families.

The resulting test cases can be represented in the following tables:

Code Path	**Command Set**		**Representative Platform**	
	Linux/Unix	Windows	Linux/Unix	Windows
pure C 32 bit	U1	W1	u1	w1

[35]Strictly speaking the test lab must also be in possession of drafts of all required documentation. In the case of private label validations closely modeled on an OpenSSL FIPS Object Module validation that is readily accomplished, usually before the formal contract with the test lab is executed.
[36]The "Module in Process" list is often referred to as the "pre-val" list.

Code Path	Command Set		Representative Platform	
	Linux/Unix	Windows	Linux/Unix	Windows
pure C 64 bit	U2	W2	u1	w2
x86 assembler	U3	W3	u2	w3
x86-64 assembler	U4	W4	u2	w4

Table 3.2.1a - Code Paths and Command Sets

where the command sets are

Command Set Name		Build Commands

U1	Linux/Unix, pure C	`./config no-asm`
		`make`
		`make install`

U2	Linux/Unix with x86/x86-64 optimizations	`./config`
		`make`
		`make install`

W1	Windows, pure C	`ms\do_fips no-asm`
W2	Windows with x86/x86-64 optimizations	`ms\do_fips`

3.2.1b - Command Sets

The actual representative systems tested for the validation were:

Generic System		***Actual System*** ***OS - Processor - Optimization***		
1	Android 2.2 on ARMv7 with NEON	Android 2.2 (HTC Desire)	Qualcomm QSD 8250 (ARMv7)	NEON
1	Android 2.2 on ARMv7 with NEON	Android 2.2 (HTC Desire)	Qualcomm QSD 8250 (ARMv7)	NEON
2	Android 2.2 on ARMv7	Android 2.2 (Dell Streak)	Qualcomm QSD 8250 (ARMv7)	None
3	Windows x86 32 bit	Microsoft Windows 7 32 bit	Intel Celeron (x86)	None
4	uCLinux on ARMv4	uClinux 0.9.29	ARM 922T (ARMv4)	None

Generic System		***Actual System*** ***OS - Processor - Optimization***		
5	Linux 2.6 on x86 with AES-NI 64 bit	Fedora 14	Intel Core i5 (x86)	AES-NI
6	HP-UX 11 on IA64 32 bit	HP-UX 11i (hpux-ia64-cc, 32 bit mode)	Intel Itanium 2 (IA64)	None
7	HP-UX 11 on IA64 64 bit	HP-UX 11i (hpux64-ia64-cc, 64 bit mode)	Intel Itanium 2 (IA64)	None
8	Linux on x86 32bit	Ubuntu 10.04	Intel Pentium T4200 (x86)	None
9	Android 2.2 on ARMv7 (duplicate of platform 2)	Android 2.2 (Motorola Xoom)	NVIDIA Tegra 250 T20 (ARMv7)	None
10	Linux 2.6 on PPC	Linux 2.6.27	PowerPC e300c3 (PPC)	None
11	Windows on x86 64 bit	Microsoft Windows 7 64 bit	Intel Pentium 4 (x86)	None
12	Linux 2.6 on x86 with AES-NI 32 bit	Ubuntu 10.04 32 bit	Intel Core i5 (x86)	AES-NI
13	Linux 2.6 on PPC (duplicate of platform 10)	Linux 2.6.33	PowerPC32 e300 (PPC)	None
16	Android 2.2 on ARMv7 with NEON (duplicate of platform 1)	Android 2.2	OMAP 3530 (ARMv7)	NEON
17	C64x+ DSP	DSP Media Framework 1.4	TI C64x+	None
19	VxWorks 6.8 on MIPS	VxWorks 6.8	TI TNETV1050 (MIPS)	None
20	Linux 2.6 on ARMv6	Linux 2.6	Broadcom BCM11107 (ARMv6)	None
21	Linux 2.6 on ARMv7	Linux 2.6	TI TMS320DM6446 (ARMv4)	None
22	Linux 2.6 on ARMv7	Linux 2.6.32	TI AM3703CBP (ARMv7)	None
23	Solaris 10 on SPARCv9 32 bit	Solaris 10 32bit	SPARC-T3 (SPARCv9)	None
24	Solaris 10 on SPARCv9 32 bit	Solaris 10 64bit	SPARC-T3 (SPARCv9)	None
25	Solaris 11 on x86-64 32 bit	Solaris 11 32bit	Intel Xeon 5260 (x86)	None
26	Solaris 11 on x86-64 64 bit	Solaris 11 64bit	Intel Xeon 5260 (x86)	None
27	Solaris 11 on x86-64 with AES-NI 32 bit	Solaris 11 32bit	Intel Xeon 5260 (x86)	AES-NI

Generic System		***Actual System*** ***OS - Processor - Optimization***		
28	Solaris 11 on x86-64 with AES-NI 64 bit	Solaris 11 64bit	Intel Xeon 5260 (x86)	AES-NI
29	Oracle Linux 5 on x86-64 64 bit	Oracle Linux 5 64bit	Intel Xeon 5260 (x86)	None
30	CascadeOS 6.1 3 on x86 32 bit	CascadeOS 6.1 32bit	Intel Pentium T4200 (x86)	None
31	CascadeOS 6.1 3 on x86 64 bit	CascadeOS 6.1 64bit	Intel Pentium T4200 (x86)	None
32	Linux 2.6 on x86-64 32 bit	Ubuntu 10.04 32bit	Intel Pentium T4200 (x86)	None
33	Linux 2.6 on x86-64 64 bit	Ubuntu 10.04 64bit	Intel Pentium T4200 (x86)	None
34	Oracle Linux 5 on x86-64 with AES-NI	Oracle Linux 5	Intel Xeon 5675 (x86)	AES-NI
35	Oracle Linux 6 on x86-64	Oracle Linux 6	Intel Xeon 5675 (x86)	None
36	Oracle Linux 6 on x86-64 with AES-NI	Oracle Linux 6	Intel Xeon 5675 (x86)	AES-NI
37	Solaris 11 32bit on SPARCv9	Solaris 11 32bit	SPARC-T3 (SPARCv9)	None
38	Solaris 11 64bit on SPARCv9	Solaris 11 64bit	SPARC-T3 (SPARCv9)	None
39	Android 4.0 on ARMv7	Android 4.0 (Motorola Xoom)	NVIDIA Tegra 250 T20	None
40	Linux 2.6 on PPC	Linux 2.6	Freescale PowerPC-e500	None
41	Apple iOS 5.1 on ARMv7	Apple iOS 5.1	ARMv7	None
42	WinCE 6.0 on ARMv5TEJ	WinCE 6.0	ARMv5TEJ	None
43	WinCE 5.0 on ARMv7	WinCE 5.0	ARMv7	None
44	Android 4.0 on ARMv7	Android 4.0	OMAP 3	NEON
45	NetBSD 5.1 on PPC	NetBSD 5.1	PowerPC-e500	None
46	NetBSD 5.1 on x86-64	NetBSD 5.1	Intel Xeon 5500 (x86)	None
47	Windows 2008 32-bit under vSphere on x86-64	Windows 2008	Xeon E3-1220v2 (x86)	None
48	Windows 2008 64-bit under vSphere on x86-64	Windows 2008	Xeon E3-1220v2 (x86)	None
49	RHEL 6 32-bit on x86-64	RHEL 6	Xeon E3-1220v2 (x86)	None
50	RHEL 6 64-bit on x86-64	RHEL 6	Xeon E3-1220v2 (x86)	None

Generic System		***Actual System*** ***OS - Processor - Optimization***		
51	Windows 7 64-bit on x86-64 with AES-NI	Windows 7	Intel Core i5-2430M (x86)	AES-NI
52	Android 4.1 on ARMv7	Android 4.1	TI DM3730 (ARMv7)	None
53	Android 4.1 on ARMv7 with NEON	Android 4.1	TI DM3730 (ARMv7)	NEON
54	Android 4.2 on ARMv7	Android 4.2	Nvidia Tegra 3 (ARMv7)	None
55	Android 4.2 on ARMv7 with NEON	Android 4.2	Nvidia Tegra 3 (ARMv7)	NEON
56	Windows Embedded Compact 7 on ARMv7 with NEON	Windows Embedded Compact 7	Freescale i.MX53xA (ARMv7)	NEON
57	Windows Embedded Compact 7 on ARMv7 with NEON	Windows Embedded Compact 7	Freescale i.MX53xA (ARMv7)	NEON
58	Android 4.0 on ARMv7 with NEON	Android 4.0	Qualcomm Snapdragon APQ8060 (ARMv7)	NEON
59	VMware Horizon Mobile 1.3 under VMware under Android 4.0 on ARMv7 with NEON	VMware Horizon Mobile 1.3 under VMware under Android 4.0	Qualcomm MSM8X60 (ARMv7)	NEON
60	Apple OS X 10.7 on x86-64	Apple OS X 10.7	Intel Core i7-3615QM (x86)	None
61	Apple iOS 5.0 on ARMv7 with NEON	Apple iOS 5.0	ARM Cortex A8 (ARMv7)	NEON
62	OpenWRT 2.6 on MIPS	OpenWRT 2.6	MIPS 24Kc	None
63	QNX 6.4 on ARMv4	QNX 6.4	Freescale i.MX25 (ARMv4)	None
64	Apple iOS 6.1 on ARMv7s	Apple iOS 6.1	Apple A6X SoC (ARMv7s)	None
65	eCos 3 on ARMv5TEJ	eCos 3	Freescale i.MX27 926ejs (ARMv5TEJ)	None
66	VMware Horizon Workspace 1.5 under vSphere on x86-64	VMware Horizon Workspace 1.5 under vSphere	Intel Xeon E3-1220 (x86)	None
67	VMware Horizon Workspace 1.5 under vSphere on x86-64 with AES-NI	VMware Horizon Workspace 1.5 under vSphere	Intel Xeon E3-1220 (x86)	AES-NI
68	Ubuntu 13.04 on ARMv7	Ubuntu 13.04	AM335x Cortex-A8 (ARMv7)	None

Generic System		***Actual System*** ***OS - Processor - Optimization***		
69	Ubuntu 13.04 on ARMv7 with NEON	Ubuntu 13.04	AM335x Cortex-A8 (ARMv7)	NEON
70	Linux 3.8 on ARMv5TEJ	Linux 3.8	ARM926 (ARMv5TEJ)	None
71	Linux 3.4 under Citrix XenServer on x86-64	Linux 3.4 under Citrix XenServer	Intel Xeon E5-2430L (x86)	None
72	Linux 3.4 under Citrix XenServer on x86-64 with AES-NI	Linux 3.4 under Citrix XenServer	Intel Xeon E5-2430L (x86)	AES-NI
73	Linux 3.4 under VMware ESX on x86-64	Linux 3.4 under VMware ESX	Intel Xeon E5-2430L (x86)	None
74	Linux 3.4 under VMware ESX on x86-64 with AES-NI	Linux 3.4 under VMware ESX	Intel Xeon E5-2430L (x86)	AES-NI
75	Linux 3.4 under Microsoft Hyper-V on x86-64	Linux 3.4 under Microsoft Hyper-V	Intel Xeon E5-2430L (x86)	None
76	Linux 3.4 under Microsoft Hyper-V on x86-64 with AES-NI	Linux 3.4 under Microsoft Hyper-V	Intel Xeon E5-2430L (x86)	AES-NI
77	Apple iOS 6.0 on ARMv7	Apple iOS 6.0	Apple A5 / ARM Cortex-A9 (ARMv7)	None
78	Apple iOS 6.0 on ARMv7 with NEON	Apple iOS 6.0	Apple A5 / ARM Cortex-A9 (ARMv7)	NEON
79	PexOS 1.0 under vSphere on x86-64	PexOS 1.0 under vSphere	Intel Xeon E5-2430L (x86)	None
80	PexOS 1.0 under vSphere on x86-64 with AES-NI	PexOS 1.0 under vSphere	Intel Xeon E5-2430L (x86)	AES-NI
81	Linux 2.6 on PPC	Linux 2.6	Freescale e500v2 (PPC)	None
82	AcanOS 1.0 on x86-64	AcanOS 1.0	ntel Core i7-3612QE (x86)	None
83	AcanOS 1.0 on x86-64 with AES-NI	AcanOS 1.0	Intel Core i7-3612QE (x86)	AES-NI
84	AcanOS 1.0 on ARMv5	AcanOS 1.0	Intel Core i7-3612QE (x86)	None
85	FreeBSD 8.4 on x86-64	FreeBSD 8.4	Intel Xeon E5440 (x86)	None
86	FreeBSD 9.1 on x86-64	FreeBSD 9.1	Xeon E5-2430L (x86)	None

Generic System		*Actual System* *OS - Processor - Optimization*		
87	FreeBSD 9.1 on x86-64 with AES-NI	FreeBSD 9.1	Xeon E5-2430L (x86)	AES-NI
88	ArbOS 5.3 on x86-64	ArbOS 5.3	Xeon E5645 (x86)	None
89	ArbOS 5.3 on x86-64 with AES-NI	ArbOS 5.3	Xeon E5645 (x86)	AES-NI
90	Linux ORACLESP 2.6 on ARMv5	Linux ORACLESP 2.6	ASPEED AST-Series (ARMv5)	None
91	Linux ORACLESP 2.6 on ARMv5	Linux ORACLESP 2.6	Emulex PILOT 3 (ARMv5)	None
92	FreeBSD 9.2 on x86-64	FreeBSD 9.2	Xeon E5-2430L (x86)	None
93	FreeBSD 9.2 on x86-64 with AES-NI	FreeBSD 9.2	Xeon E5-2430L (x86)	AES-NI
94	FreeBSD 10.0 on x86-64	FreeBSD 10.0	Xeon E5-2430L (x86)	None
95	FreeBSD 10.0 on x86-64 with AES-NI	FreeBSD 10.0	Xeon E5-2430L (x86)	AEs-NI
96				
97				
98				
99				
100				

Table 3.2.1c - Representative Systems

3.2.2 32 versus 64 Bit Architectures

Many 64 bit platforms provide backward compatible support for 32 bit code via hardware or software emulation. Software built on a 32 bit version of a specific operating system will generally run as-is on the equivalent 64 bit version of that operating system. Software built on a 64 bit operating system can be either 32 bit or 64 bit code depending on vendor build environment defaults and explicit build time options. Any such 64 bit code will not run on a 32 bit equivalent operating system, so care must be taken when compiling code for distribution to both 32 and 64 bit systems.

By default the FIPS Object Module build process will generate 64 bit code on 64 bit systems.

Since the command sets included in the validation testing do not permit the explicit specification of the compile time options that would otherwise be used to specify the generation of 32 or 64 bit code, it may be necessary for some platforms to build a 32 bit FIPS Object Module on a 32 bit system, and conversely for 64 bit.

It is also possible on most 64-bit platforms to install a 32-bit build environment which would be supported. Details as to how to configure such an environment are beyond the scope of this document.

3.2.3 Assembler Optimizations

The only option for processor architectures other than x86/x86-64 and ARM is to use the pure C language implementation and not any of the hand-coded performance optimized assembler as each assembler implementation requires separate FIPS testing. For example, an Itanium or PowerPC system can only build and use the pure C language module.

For the x86/x86-64 and ARM processors several levels of optimization are supported by the code. Note that most such optimizations, if compiled into executable code, are selectively enabled at runtime depending on the capabilities of the target processor. If the Module is built and executed on the same platform (the build-time and run-time systems are the same) then the appropriate optimization will automatically be utilized (assuming that the build+target system corresponds to a formally tested platform).

For x86-64 there are three possible optimization levels:

1. No optimization (plain C)
2. SSE2 optimization
3. AES-NI+PCLMULQDQ+SSSE3 optimization

Note that other theoretically possible combinations (e.g. AES-NI only, or SSE3 only) are not addressed individually, so that a processor which does not support all three of AES-NI, PCLMULQDQ, and SSSE3 will fall back to only SSE2 optimization.

The runtime environment variable OPENSSL_ia32cap=~0x200000200000000 disables use of AES-NI, PCLMULQDQ, and SSSE3 optimizations for x86-64.

For ARM there are two possible optimization levels:

1. Without NEON
2. With NEON (ARM7 only)

The runtime variable OPENSSL_armcap=0 disables use of NEON optimizations for ARM.

If all optimization levels have not been formally tested for a given platform, care must be taken to verify that the optimizations enabled at run-time on any target systems correspond to a formally tested platform. For instance, if "Windows on x86 32-bit" was formally tested but "Windows on x86 with AES-NI 32-bit" was not[37] then the Module would be validated when executed on a non-AES-NI capable target processor, but would *not* be validated when executed on an AES-NI capable system. Note the processor optimization capabilities will often not be obvious to administrators or end users installing software.

When the target platforms are not known to have capabilities corresponding to tested platforms then the risk of inadvertently utilizing the unvalidated optimizations at run-time can can be avoided by setting the appropriate environment variables at run-time[38]:

Disabling run-time selectable optimizations		
Platform	**Environment Variable**	**Value**
x86/x86-64	OPENSSL_ia32cap	~0x200000200000000
ARM	OPENSSL_armcap	0

3.3 Creation of Shared Libraries

The FIPS Object Module is not directly usable as a shared library, but it can be linked into an application that is a shared library. A “FIPS compatible” OpenSSL distribution will automatically incorporate an available FIPS Object Module into the **`libcrypto`** shared library when built using the **`fips`** option (see §4.2.3).

3.4 Cross-compilation

Compilers and linkers are separate programs which work together to generate object code for a target system. They are also programs composed of object code that is executed on the build system. When the build and target systems are the same we say the process is referred to as a "native" build; when they are different it is referred to as a "cross-compilation" build.

Many compilers and linkers (or build environments containing compilers and linkers) are capable of creating object code for multiple target platforms. For the case of the native build the **`./config`** command[39] automatically determines the target system from the characteristics of the build system. This determination is made by setting a series of variables that are used to select an

[37]This was the case as of the initial OpenSSL FIPS Object Module 2.0 validation, though such platforms may be added by subsequent modifications.
[38]An alternative is to sponsor the addition of the unsupported platform optimization to the validated Module
[39]Microsoft Windows platforms are handled somewhat differently and are discussed elsewhere.

arbitrary architecture label defined in the **./Configure** command that is invoked by **./config**. This architecture label can be displayed with the "**-t**" command line option:

```
$ ./config -t
Operating system: i686-whatever-linux2
Configuring for linux-elf
/usr/bin/perl ./Configure linux-elf -march=pentium -Wa,--
noexecstack
$
```

In this example the architecture target is "linux-elf" and the **./Configure** command will be invoked with the additional arguments "**-march=pentium -Wa,--noexecstack** ".

This implicit determination of the target architecture can be overridden by manually specifying the appropriate environment variables. This explicit determination is optional and unnecessary for native builds, but required for cross-compilation. A typical example is shown here for cross-compilation for the Android ARM target platform:

```
#!/bin/sh
# Edit this to wherever you unpacked the NDK
export ANDROID_NDK="$PWD"
# Edit to wherever you put incore script
export FIPS_SIG="$PWD/incore"

# Shouldn't need to edit anything past here.
PATH=$ANDROID_NDK/android-ndk-r4b/build/prebuilt/linux-
x86/arm-eabi-4.4.0/bin:$PATH ; export PATH
export MACHINE=armv7l
export RELEASE=2.6.32.GMU
export SYSTEM=android
export ARCH=arm
export CROSS_COMPILE="arm-eabi-"
export ANDROID_DEV="$ANDROID_NDK/android-ndk-
r4b/build/platforms/android-8/arch-arm/usr"
export HOSTCC=gcc
```

With those environment variables specified on a Linux x86 system the **./config** now selects a different target architecture:

```
$ ./config -t
Operating system: armv7l-whatever-android
Configuring for android-armv7
```

```
/usr/bin/perl ./Configure android-armv7 -Wa,--noexecstack
$
```

When building using cross-compilation a different technique must be used to determine the embedded integrity check digest value. For native builds an interim executable is created and executed to calculate this digest from live memory, in the same way that the digest is calculated at runtime during the POST integrity test. When cross-compiling that technique cannot be used because the cross-compiled executables cannot (in general) be run on the build host.

Instead of building and executing an interim executable, a special purpose utility is used to calculate the digest by examining the cross-compiled object code as it resides on disk. One such utility, *incore*, is provided to handle ELF formats. Even though this utility is effectively platform neutral on most Linux-like operating systems , the process as a whole is not designed to work with arbitrary ELF code and can be relied on only for explicitly verified cross-compile cases as reflected in *fips/fips_canister.c*. Accommodation of new cross-compilation targets is likely to be trivial but will still require separate validation.

Thus, although the *incore* utility is theoretically capable of handling arbitrary ELF binary code (native or not), it is not used in non-cross-compile/native cases. Cross-compiled non-ELF platforms would require different utilities and separate validation.

In general the C compiler is required to segregate constant data in a contiguous area (e.g. by placing it in a dedicated segment) to compile the FIPS module. Some compilers were found to fail to meet the const data segment requirement. In the cases where the errant behavior was observed, the compiler was instructed to generate position-independent code[40].

In such cases it might be possible to rectify the problem by defining the *__fips_constseg* macro in *fips/fipssyms.h* and harmonizing that definition with declaration of *FIPS_rodata_start* and *FIPS_rodata_end* in *fips/fips_canister.c*. Unfortunately, such an approach will require a separate FIPS 140-2 validation, however.

[40]The primary reason for compiling the FIPS 2.0 module with *-fPIC* is for versatility, so that the *fipscanister* object module will be usable in either the context of a statically-linked application or dynamic library. Use of non-PIC code is inappropriate in a dynamic library, but linking PIC statically was proven to work on all tested platforms. Thus, where such versatility is not of interest then *-fPIC* could be dropped to target statically-linked applications only. A separate validation will be required, of course.

4. Generating the FIPS Object Module

This section describes the creation of a FIPS Object Module for subsequent use by an application. The *Security Policy* provides procedures for acquiring, verifying, building, installing, protecting, and initializing the FIPS Object Module. In case of discrepancies between the *User Guide* and the *Security Policy*, the *Security Policy* should be used.

Finally, recall from Section 2.4.2, *Object Module (Link Time) Integrity*, that applications link against **`libcrypto.so`** or **`libcrypto.a`**, and not directly to **`fipscanister.o`**.

4.1 Delivery of Source Code

The OpenSSL FIPS Object Module software is only available in source format. The specific source code distributions can be found at http://www.openssl.org/source/[41]. as files with names of the form *openssl-fip-2.0.N.tar.gz* where the revision number *N* reflects successive extensions of the FIPS Object Module to support additional platforms:

http://www.openssl.org/source/openssl-fips-2.0.tar.gz
http://www.openssl.org/source/openssl-fips-2.0.1.tar.gz
http://www.openssl.org/source/openssl-fips-2.0.2.tar.gz

The latest revision will be suitable for all tested platforms, whereas earlier revisions will work only for the platforms tested as of that revision.

The CMVP introduced significant new requirements for verification of the 2.0 source code distribution. This requirement is discussed in more detail in §4.1.3; but in summary, it can no longer be downloaded and used as before. A "trusted path" must be used for transfer of the source code distribution.

At present the one method known to satisfy the “trusted path” requirement is obtain the source code distribution from the vendor of record (OSF) on physical media (CD). For instructions on requesting this CD see http://openssl.com/fips/verify.html.

The OpenSSL FIPS Object Module software was delivered to the FIPS 140-2 testing laboratory in source form as this complete OpenSSL distribution, and was built by the testing laboratory using the standard build procedure as described in the Security Policy document and reproduced below and in Appendix B.

[41]Closely related distributions lacking binary curve ECC, opensl-fips-ecp-2.0.*N*.tar.gz, are also available; see §6.5.

For each of the `openssl-fips-2.0.N.tar.gz` distributions there is also a distribution file with the name of the form `openssl-fips-ecp-2.0.N.tar.gz`. These "ecp" distributions are the same as the corresponding 2.0.*N* distributions with binary curve ECC omitted (see Section 6.5).

Note: OSF recommends that the downloaded tarballs be considered untrusted for any purpose until verified as described in §4.1.2.

4.1.1 Creation of a FIPS Object Module from Other Source Code

Many OpenSSL distributions other than the specific distributions used for the validation can be used to build a **`fipscanister.o`** object using undocumented build-time options. The reader is reminded that any such object code *cannot* be used or represented as FIPS 140-2 validated. The Security Policy document is very clear on that point.

4.1.2 Verifying Integrity of Distribution (Best Practice)

This step is optional and not mandated by the FIPS140-2 validation. It is also not recognized as having any value by the CMVP, but is considered a best practice by the OpenSSL team for all software downloads from OpenSSL.

The integrity and authenticity of the complete OpenSSL distribution should be validated manually with the PGP signatures[42] published by the OpenSSL team with the distributions (ftp://ftp.openssl.org/source/) to guard against a corrupted source distribution. Note this check is *separate and distinct* from the CMVP mandated FIPS 140-2 source file integrity check (§4.1.3).

The PGP signatures are contained in the file

`openssl-fips-2.0.tar.gz.asc`

This digital signature of the distribution file can be verified against the OpenSSL PGP public key by using the PGP or GPG applications (GPG can be obtained free of charge from http://www.gnupg.org/)[43]. This validation consists of confirming that the distribution was signed by a known trusted key as identified in Appendix A, "*OpenSSL Distribution Signing Keys*".

First, find out which key was used to sign the distribution. Any of several different valid keys may have been used for this purpose. The "hexadecimal key id", an identifier used for locating keys on the keystore servers, is displayed when attempting to verify the distribution. If the signing key is not already in your keyring the hexadecimal key id of the unknown key will still be displayed:

```
$ gpg  openssl-1.0.1z.tar.gz.asc
gpg: Signature made Tue Sep 30 09:00:37 2009 using RSA key ID 49A563D9
gpg: Can't check signature: public key not found
$
```

[42] No
[43] No
valid
used. The examples given here are applicable to both GPG and PGP.

Example 4.1.2a - Find Id of Signing Key

In this example the key id is **0x49A563D9.** Next see if this key id belongs to one of the OpenSSL core team members authorized to sign distributions. The authorized keys are listed in Appendix A.

Note that some older versions of gpg will not display the key id of an unknown public key; either upgrade to a newer version or load all of the authorized keys.

If the hexadecimal key id matches one of the known valid OpenSSL core team keys then download and import the key.

PGP keys can be downloaded interactively from a keyserver web interface or directly by the pgp or gpg commands.

The hexadecimal key id of the team member key (for example, the search string "**0x49A563D9**" can be used to download the OpenSSL PGP key from a public keyserver (http://www.keyserver.net/, http://pgp.mit.edu, or others). Keys can be downloaded interactively to an intermediate file or directly by the pgp or gpg program.

Once downloaded to an intermediate file, *markcox.key* in this example, the key can be imported with the command:

```
$ gpg --import markcox.key
gpg: key 49A563D9: public key "Mark Cox <mjc@redhat.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
$
```

Example 4.1.2b - Importing a Key from a Downloaded file

These examples assume the *pgp* or *gpg* software is installed. The key may also be imported directly into your keyring:

```
$ gpg --keyserver pgp.mit.edu --recv-key 49a563d9
gpg: key 49A563D9: public key "Mark Cox <mjc@redhat.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

Example 4.1.2c - PGP Key Import

Note that at this point we have not yet established that the key is authentic or that the distribution was signed with that key; a key that *might* be authentic has been obtained in a form where it can be utilized for further validation.

To verify that the distribution file was signed by the imported key use the `pgp` or `gpg` command with the signature file as the argument, with the distribution file also present in the same directory:

```
$ gpg /work/build/openssl/openssl-1.0.1.tar.gz.asc
gpg: Signature made Tue Sep 30 09:00:37 2009  using RSA key ID 49A563D9
gpg: Good signature from "Mark Cox <mjc@redhat.com>"
gpg:                  aka "Mark Cox <mark@awe.com>"
gpg:                  aka "Mark Cox <mark@c2.net>"
gpg:                  aka "Mark Cox <mcox@c2.net>"
gpg:                  aka "Mark Cox <mark@ukweb.com>"
gpg:                  aka "Mark Cox <mjc@apache.org>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 7B 79 19 FA 71 6B 87 25  0E 77 21 E5 52 D9 83 BF
$
```

Example 4.1.2d - PGP File Signature Verification

In this example the validity of the file signature with respect to the key was verified. That is, the target file **`openssl-fips-2.0.tar.gz`** was signed by the key with id `49A563D9`. The warning message in this example is alerting the key is not part of the "web of trust", a relational ranking system based on manually assigned confidence levels. Instead of relying on the web of trust which will differ from one user to another, the key should be matched directly to a list of known valid keys.

The final step of verification is to establish that the signing key is authentic. To do so, confirm the key fingerprint of the key which signed the distribution is one of the valid OpenSSL core team keys listed in Appendix A, "*OpenSSL Distribution Signing Keys*". In this example, **`7B 79 19 FA 71 6B 87 25  0E 77 21 E5 52 D9 83 BF`** is in fact authentic according to Appendix A.

4.1.3 Verifying Integrity of the Full Distribution for the FIPS Object Module

IMPORTANT NOTE: This step has changed from prior validations, and is required per the OpenSSL Security Policy!

The validation now includes a requirement for "secure installation." In practice that means the distribution file should be obtained directly from the vendor (OSF) on physical media. A more complete discussion of this requirement including the elaborate steps needed when the distribution is *not* obtained on physical media can be found in §6.6.

Physical media can be requested from OSF at:

OpenSSL Software Foundation, Inc.
1829 Mount Ephraim Road
Adamstown, MD 21710

USA
+1 877-OPENSSL
(+1 877 673 6775)
verifycd@verifycd@openssl.com

An E-mail containing the full postal address is the preferred point of contact. It is our intention to provide these CDs at no cost as long as we are able. We ask that you only request this CD if you plan to use it for generation of FIPS 140-2 validated cryptography in a context that requires such compliance. For any other purposes the downloaded files are bit-for-bit identical and will generate exactly the same results.

The simpler verification requirement for prior OpenSSL FIPS Object Module validations, namely:

> The HMAC-SHA-1 digest of the distribution file is published in Appendix B of the *Security Policy*. The *Security Policy* can be found at NIST, http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1051.pdf.
>
> This digest should be calculated and compared against the published value, as in:
>
> $ env OPENSSL_FIPS=1 openssl sha1 -hmac etaonrishdlcupfm openssl-fips-2.0.tar.gz
>
> where the **`openssl`** command is from a recent version of OpenSSL that supports the **`-hmac`** option[44]. If you don't have the **`openssl`** command yet it will be generated by the build process.

...is now specifically disallowed. With the new requirement use of the **`openssl`** command, even from another version of the OpenSSL FIPS Object Module, is no longer permitted as in general it will not have been obtained via a "secure installation".

4.2 Building and Installing the FIPS Object Module with OpenSSL (Unix/Linux)

Due to significant differences in the two basic operating system families, Unix®/Linux® and Microsoft® Windows® platforms are discussed separately. Instructions for Windows® are given in §4.3. In addition, a Mac OS X example is offered at E.1 Apple OS X Support; and an iOS example is given in Error: Reference source not found.

4.2.1 Building the FIPS Object Module from Source

Next build the FIPS Object Module from source. The FIPS 140-2 validation specific code is incorporated into the resulting FIPS Object Module when the **`fips`** configuration option is

[44]The **`OPENSSL_FIPS=1`** environment variable will enable **`FIPS`** mode for an **`openssl`** command built from a FIPS capable OpenSSL distribution.

specified. Per the conditions of the FIPS 140-2 validation only two configuration commands may be used:

```
./config
```

or

```
./config no-asm
```

where the specific option used depends on the platform (see §3.2.1). Note that "fips canister" is implied, so there is no need for either **./config fipscanisterbuild** or **./config fips**.

The environment variable **FIPSDIR**, if present, points to the pathname of the location where the validated module will be installed. This location defaults to **/usr/local/ssl/fips-2.0**.

The specification of any other options on the command line, such as

```
./config shared
```

is *not* permitted. Note that in the case of the "**shared**" option position independent code is generated by default so the generated FIPS Object Module can be included in a shared library[45].

Note that as a condition of the FIPS 140-2 validation no other user specified configuration options may be specified. This restriction means that an optional install prefix cannot be specified – however, there is no restriction on subsequent manual relocation of the generated files to the desired final location.

Then:

```
make
```

to generate the FIPS Object Module file **fipscanister.o**, the digest for the FIPS Object Module file, **fipscanister.o.sha1**, and the source file used to generate the embedded digest, **fips_premain.c**. The **fipscanister.o**, **fipscanister.o.sha1**, and **fips_premain.c** files are intermediate files (i.e., used in the generation of an application but not referenced by that application at runtime). The object code in the **fipscanister.o** file is incorporated into the runtime executable application at the time the binary executable is generated.

This should also be obvious, but modifications to any of the intermediate files generated by the "**./config**" or "**make**" commands are not permitted. If the original distribution is modified, or if anything other than those three specified commands are used, or if any intermediate files are modified, the result is *not* FIPS validated.

[45] If not for the FIPS validation prohibition, on most but not all platforms the "**shared**" option could safely be chosen regardless of the intended use. See Appendix E for one known exception.

4.2.2 Installing and Protecting the FIPS Object Module

The system administrator should install the generated **fipscanister.o**, **fipscanister.o.sha1**, and **fips_premain.c** files in a location protected by the host operating system security features. These protections should allow write access only to authorized system administrators (FIPS 140-2 Crypto Officers) and read access only to authorized users.

For Unix® based or Linux® systems this protection usually takes the form of *root* ownership and permissions of *0755* or less for those files and all parent directories. When all system users are not also authorized users the world (public) read and execute permissions should be removed from these files.

The usual

```
make install
```

will install the **fipscanister.o**, **fipscanister.o.sha1**, **fips_premain.c**, and **fips_premain.c.sha1** files in the target location (typically ***/usr/local/ssl/fips-2.0/lib/*** for Unix® based or Linux® systems, or as specified by the **FIPSDIR** environment variable) with the appropriate permissions to satisfy the security requirement. These four files constitute the validated FIPS Object Module; the other files also installed by this command are *not* validated. Note that it is also permissible to install these files in other locations by other means, provided that they are protected with appropriate permissions as noted above:

```
cp fipscanister.o fipscanister.o.sha1 <target-directory>
cp fips_premain.c fips_premain.c.sha1 <target-directory>
```

Note that **fipscanister.o** can either be statically linked into an application binary executable, or statically linked into a shared library.

4.2.3 Building a FIPS Capable OpenSSL

Once the validated FIPS Object Module has been generated it is usually combined with an OpenSSL distribution in order to provide the standard OpenSSL API. Any 1.0.1 release can be used for this purpose. The commands

```
./config fips <...other options...>
make <...options...>
make install
```

will build and install the new OpenSSL without overwriting the validated FIPS Object Module files. The **FIPSDIR** environment variable or the **--with-fipsdir** command line option can be used to explicitly reference the location of the FIPS Object Module (**fipscanister.o**).

The combination of the validated FIPS Object Module plus an OpenSSL distribution built in this way is referred to as a *FIPS capable OpenSSL*, as it can be used either as a drop-in replacement for a non-FIPS OpenSSL or for use in generating FIPS mode applications.

Note that a standard OpenSSL distribution built for use with the FIPS Object Module must have the **./config fips** option specified. Other configuration options may be specified in addition to **fips**, but omission of the **fips** option will cause errors when using the OpenSSL libraries with the FIPS Object Module.

4.3 Building and Installing the FIPS Object Module with OpenSSL (Windows)

The build procedure for Windows is similar to that for the regular OpenSSL product, using MSVC and NASM for compilation. Note MASM is not supported.

The second stage uses VC++ to link OpenSSL 1.0.1 or later against the installed FIPS module, to obtain the complete FIPS capable OpenSSL. Both static and shared libraries are supported.

4.3.1 Building the FIPS Object Module from Source

Build the FIPS Object Module from source:

```
ms\do_fips [no-asm]
```

where the **no-asm** option may or may not be present depending on the platform (see §3.2.1).

Note that as a condition of the FIPS 140-2 validation no other user specified configuration options may be specified.

4.3.2 Installing and Protecting the FIPS Object Module

The system administrator should install the generated **fipscanister.lib**, **fipscanister.lib.sha1**, and **fips_premain.c** files in a location protected by the host operating system security features. These protections should allow write access only to authorized system administrators (FIPS 140-2 Crypto Officers) and read access only to authorized users.

For Microsoft® Windows® based systems this protection can be provided by ACLs limiting write access to the *administrator* group. When all system users are not authorized users the Everyone (public) read and execute permissions should be removed from these files.

4.3.3 Building a FIPS Capable OpenSSL

The final stage is VC++ compilation of a standard OpenSSL distribution to be referenced in conjunction with the previously built and installed FIPS Object Module.

Download an OpenSSL 1.0.1 distribution. Follow the standard Windows® build procedure except that instead of the command:

```
perl Configure VC-WIN32
```

do:

```
perl Configure VC-WIN32 fips --with-fipsdir=c:\fips\path
```

where "**c:*fips**path***" is wherever the FIPS module from the first stage was installed. Static and shared library builds are supported.

This command is followed by the usual

```
ms\do_nasm
```

and

```
nmake -f ms\ntdll.mak
```

to build the shared libraries only, or

```
nmake -f ms\nt.mak
```

to build the OpenSSL static libraries. The standard OpenSSL build with the **fips** option will use a base address for **libeay32.dll** of **0xFB00000** by default. This value was chosen because it is unlikely to conflict with other dynamically loaded libraries. In the event of a clash with another dynamically loaded library which will trigger runtime relocation of **libeay32.dll**, the integrity check will fail with the error

```
FIPS_R_FINGERPRINT_DOES_NOT_MATCH_NONPIC_RELOCATED
```

A base address conflict can be resolved by shuffling the other DLLs or re-compiling OpenSSL with an alternative base address specified with the **--with-baseaddr=** option.

Note that the developer can identify which DLLs are relocated with the Process Explorer utility from http://www.microsoft.com/technet/sysinternals/ProcessesAndThreads/ProcessExplorer.mspx.

The resulting FIPS capable OpenSSL can be used for shared or static linking. The shared library built (when **`ms\ntdll.mak`** is used as the Makefile) links **`fipscanister.lib`** into **`libeay32.dll`** using **`fipslink.pl`** in accordance with the requirements of the *Security Policy*.

5. Creating Applications Which Reference the FIPS Object Module

Only minor modifications are needed to adapt most applications that currently use OpenSSL for cryptography to use the FIPS capable OpenSSL with the FIPS Object Module. The checklist in Figure 4 summarizes the modifications which are covered in more detail in the following discussion:

- Use the FIPS Object Module for all cryptography
- Initialize FIPS mode with **`FIPS_mode_set()`**
- Generate application executable object with embedded FIPS Object Module digest
- Protect critical security parameters

Figure 4 - Application Checklist

Appendix C contains a simple but complete sample application utilizing the FIPS Object Module with OpenSSL as described in this section.

5.1 Exclusive Use of the FIPS Object Module for Cryptography

In order for the referencing application to claim FIPS 140-2 validation, all cryptographic functions utilized by the application must be provided exclusively by the FIPS Object Module. The OpenSSL API used in conjunction with the FIPS Object Module in FIPS mode is designed to automatically disable all non-FIPS cryptographic algorithms.

5.2 FIPS Mode Initialization

Somewhere very early in the execution of the application FIPS mode must be enabled. This should be done by invocation of the **`FIPS_mode_set()`** function call, either directly or indirectly as in these following examples.

Note that it is permitted to *not* enable FIPS mode, in which case OpenSSL should function as it always has. The application will not, of course, be operating in validated mode.

The FIPS_mode_set() function call when invoked with any positive argument will enable the FIPS mode of operation. Depending on the argument it may also enable additional restrictions. For example, an argument of 1 will enable the basic FIPS mode where all FIPS approved algorithms are available. An argument of **`FIPS_SUITEB`** (2) will restrict the available algorithms to those allowed by the Suite B specification.

Option 1: Direct call to FIPS_mode_set()

```
#ifdef OPENSSL_FIPS
if(options.no_fips <= 0)
{
if(!FIPS_mode_set(1))
{
ERR_load_crypto_strings();
ERR_print_errors_fp(stderr);
exit(1);
            }
else
fprintf(stderr,"*** IN FIPS MODE ***\n");
}
#endif
```

Example 5.2a – Direct Invocation of FIPS_mode_set()

Option 2: Indirect call via OPENSSL_config()

The **OPENSSL_config()** call can be used to enable FIPS mode via the standard **openssl.conf** configuration file:

```
OPENSSL_config("XXXX_conf")

#ifdef OPENSSL_FIPS
if (FIPS_mode())
{
      fprintf(stderr,"*** IN FIPS MODE ***\n");
}
#endif
```

Example 5.2b – Indirect Invocation of FIPS_mode_set()

```
 # Default section
XXXX_conf = XXXX_options

...

[ XXXX_options ]
alg_section = algs

...

[ algs ]
fips_mode = yes

...
```

Example 5.2c – Sample openssl.conf File

The call to **OPENSSL_config("XXXX_conf")** will check the system default OpenSSL configuration file for a section **XXXX_conf**. If section **XXXX_conf** is not found then the section defaults to **openssl_conf**. The resulting section is checked for an **alg_section** specification naming a section that can contain an optional "**fips_mode = yes**" statement.

Note that **OPENSSL_config()** has no return code. If a configuration error occurs it will write to STDERR and forcibly exit the application. Applications that want finer control can call the underlying functions such as **CONF_modules_load_file()** directly.

5.3 Generate Application Executable Object

Note that applications interfacing with the FIPS Object Module are outside of the cryptographic boundary.

When statically 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 code and read-only data must be generated and embedded in the application executable object for use by the **FIPS_mode_set()** function at runtime initialization.

Note the application that statically links the Module can be a shared library (DLL for Microsoft Windows).

When the FIPS Object Module has been incorporated in a shared library then subsequent dynamic linking of an application to that shared library is done the usual way and these steps are irrelevant.

For static linking the embedding of the runtime digest can be accomplished in one of two ways:

1. Two Step Linking with Interim Runtime Executable

   Earlier versions of the FIPS Object Module supported only this technique, where an initial link is performed to create an interim executable which is then executed in the target environment to calculate and display the digest value. A second link is performed to create the final executable with the embedded digest value. This two step process is typically performed by the fipslink.pl utility.

   This two step technique works well enough for native builds, where the build system and runtime target system are the same, but is awkward at best for cross-compilation due to the need to move the interim executable to the target system, execute it, and retrieve the calculated digest.

   This technique does have the advantage of working (at least in principle) for all platforms.

2. In-place Editing of the Object Code

   In order to ease the task of cross-compiling the FIPS Object Module, a new technique was developed. Instead of determining the runtime digest value by actual execution on the target system, a utility is used to analyze the compiled object code on the build system and calculate the digest. This utility is platform (or object code format) sensitive. For ELF binaries it is called *incore*, for Microsoft Windows *msincore*, for OS X and iOS *incore_macho*.

5.3.1 Linking under Unix/Linux

The OpenSSL distribution contains a utility, **`fipsld`**, which both performs the check of the FIPS Object Module and generates the new HMAC-SHA-1 digest for the application executable. The **`fipsld`** utility has been designed to act as a front end for the actual compilation and linking operations in order to ease the task of modifying an existing software project to incorporate the FIPS Object Module. It can be used to create either binary executables or shared libraries.

The **`fipsld`** command requires that the **`CC`** and/or **`FIPSLD_CC`** environment variables be set, with the latter taking precedence. These variables allow a typical Makefile to be used without modification by specifying a command of the form

```
make CC=fipsld FIPSLD_CC=gcc
```

where **`fipsld`** is invoked by **`make`** in lieu of the original compiler and linker (**`gcc`** in this example), and in turn invokes that compiler where appropriate. Note that **`CC=fipsld`** can be passed to autoconf configure scripts as well.

This type of command line macro overloading will work for many smaller software projects. The makefile can also be modified to achieve the same macro substitutions. Depending on the form of the Makefile this substitution may be as simple as defining **FIPSLD_CC** to reference the actual C compiler and redefining the **CC** macro to reference **fipsld**:

```
FIPSLD_CC = $(CC)
CC = fipsld
     .
     .
     .
<application>: $(OBJS)
     $(CC) $($CFLAGS) -o $@ $(OBJS) $(LIBCRYPTO) ...
```

Setting **CC=fipsld** is appropriate when the link rules rely on **$(CC)** instead of **ld** to produce the executable images, but in some cases it may be desirable or necessary to not redefine the $(CC) macro variable. A typical makefile rule referencing **fipsld** directly for the link step would look something like[46]:

```
OPENSSLDIR = /usr/local/ssl/fips-2.0
FIPSMODULE = $(OPENSSLDIR)/lib/fipscanister.o
     .
     .
     .
<application>: $(OBJS) $(FIPSMODULE)
    env FIPSLD_CC=$(CC) fipsld $(CFLAGS) -o $@ $(OBJS) \
          $(LIBS) $(LIBCRYPTO)
```

Even though the **fipsld** command name implies use as a replacement for the **ld** command, it also invokes the C compiler between the two link stages, hence **fipsld** can also replace **$(CC)** in rules producing .o object files, replacing both compilation and linking steps for the entire Makefile, i.e.:

```
<application>.o: <application>.c
     $(CC) $(CFLAGS) -c <application>.c ...

<application>: $(OBJS)
     ld -o $@ $(OBJS) $(LIBCRYPTO) ...
```

becomes

[46]The use of **env** is actually redundant in a Makefile context, but is specified here to give a command line also valid for non-Bourne shells.

```
<application>: <application>.c
     env FIPSLD_CC=$(CC) fipsld $(CFLAGS) -o $@ $@.c \
          $(LIBCRYPTO) ...
```

Larger software projects are likely to prefer to modify only the Makefile rule(s) linking the application itself, leaving other Makefile rules intact. For these more complicated Makefiles the individual rules can be modified to substitute **`fipsld`** for just the relevant compilation linking steps.

The **`fipsld`** command is designed to locate **`fipscanister.o`** automatically. It will verify that the HMAC-SHA-1 digest in file **`fipscanister.o.sha1`** matches the digest generated from **`fipscanister.o`**, and will then create the file ***`<application>`*** containing the object code from **`fipscanister.o`**, and embedded within that the digest calculated from the object code and data in **`fipscanister.o`** .

At runtime the **`FIPS_mode_set()`** function compares the embedded HMAC-SHA-1 digest with a digest generated from the text and data areas. This digest is the final link in the chain of validation from the original source to the application executable object file.

5.3.2 Linking under Windows

For a shared library application just linking with the DLL is sufficient. Linking an application with the static libraries involves a bit more work, and can be complicated by the fact that GUI based tools are often used for such linking.

For the Windows® environment a perl script **`fipslink.pl`** is provided which performs a function similar to **`fipsld`** for Unix®/Linux®. Several environment variables need to be set:

`FIPS_LINK` is the linker name, normally "**`link`**"

`FIPS_CC` is the C compiler name, normally "**`cl`**"

`FIPS_CC_ARGS` is a string of C compiler arguments for compiling **`fips_premain.c`**

`PREMAIN_DSO_EXE` should be set to the path to **`fips_premain_dso.exe`** if a DLL is being linked (can be omitted otherwise)

`PREMAIN_SHA1_EXE` is the full path to **`fips_standalone_sha1.exe`**

`FIPS_TARGET` is the path of the target executable or DLL file

FIPSLIB_D is the path to the directory containing the installed FIPS module

When these variables are specified **fipslink.pl** can be called in the same way as the standard linker. It will automatically check the hashes, link the target, generate the target in-core hash, and link a second time to embed the hash in the target file.

The static library Makefile **ms\nt.mak** in the OpenSSL distribution gives an example of the usage of **fipslink.pl**.

5.4 Application Implementation Recommendations

This section describes additional steps not strictly required for FIPS 140-2 validation but recommended as good practice.

Provide an Indication of FIPS Mode

Security and risk assessment auditors will want to verify that an application utilizing cryptography is using FIPS 140-2 validated software in a FIPS compliant mode. Many such applications will superficially appear to function the same whether built with a non-FIPS OpenSSL, when built with the FIPS Object Module and running in non-FIPS mode, and when built with the FIPS Object Module and running in FIPS mode.

As an aid to such reviews the application designer should provide a readily visible indication that the application has initialized the FIPS Object Module to FIPS mode, after a successful return from the **FIPS_mode_set()** API call. The indication can take the form of a **tty** or **stdout** message, a **syslog** entry, or an addition to a protocol greeting banner. For example a SSH server could print a protocol banner of the form:

```
SSH-2.0-OpenSSH_3.7.1p2 FIPS
```

to provide an easily referenced indication that the server was properly initialized to FIPS mode.

Graceful Avoidance of Non-FIPS Algorithms

Many applications allow end user and/or system administrator configurable specification of cryptographic algorithms. The OpenSSL API used with the FIPS Object Module in FIPS mode is designed to return error conditions when an attempt is made to use a non-FIPS algorithm via the OpenSSL API. These errors may result in unexpected failure of the application, including fatal assert errors for algorithm function calls lacking a testable return code. However, there is no guarantee that the OpenSSL API will always return an error condition in every possible permutation or sequence of API calls that might invoke code relating to non-FIPS algorithms. In any case, it is the responsibility of the application programmer to avoid the use of non-FIPS algorithms. Unexpected run-time errors can be avoided if the cipher suites or other algorithm selection options

are defaulted to FIPS approved algorithms, and if warning or error messages are generated for any end user selection of non-FIPS algorithms.

5.5 Documentation and Record-keeping Recommendations

The supplier or developer of a product based on the FIPS Object Module cannot claim that the product itself is FIPS 140-2 validated under certificate #1747. Instead a statement similar to the following is recommended:

> *Product XXXX uses an embedded FIPS 140-2-validated cryptographic module (Certificate #1747) running on a YYYY platform per FIPS 140-2 Implementation Guidance section G.5 guidelines.*

where XXXX is the product name ("Cryptomagical Enfabulator v3.1®") and YYYY is the host operating system ("Solaris 10").

This statement asserts "user affirmation" of the validation per Section G.5 of the *Implementation Guidance* document.

While not strictly required by the Security Policy or FIPS 140-2, a written record documenting compliance with the Security Policy would be a prudent precaution for any party generating and using or distributing an application that will be subject to FIPS 140-2 compliance requirements. This record should document the following:

For the FIPS Object Module generation:

1. Where the **`openssl-fips-2.0.tar.gz`** distribution file was obtained from, and how the HMAC SHA-1 digest of that file was verified per Appendix B of the Security Policy.

2. The host platform on which the **`fipscanister.o`**, **`fipscanister.o.sha1`**, **`fips_premain.c`**, and **`fips_premain.c.sha1`** files were generated. This platform identification at a minimum should note the processor architecture ("x86", "PA-RISC",...), the operating system ("Solaris 10", "Windows XP",...), and the compiler ("gcc 3.4.3",...).

3. An assertion that the **`fipscanister.o`** module was generated with the three commands
   ```
   ./config [no-asm]
   make
   make install
   ```
   and specifically that no other build-time options were specified.

4. A record of the HMAC SHA-1 digest of the **`fipscanister.o`** (the contents of the **`fipscanister.o.sha1`** file). That digest identifies this specific FIPS Object Module;

if you immediately build another module it will have a different digest and is a different FIPS Object Module.

5. An assertion that the contents of the distribution file were not manually modified in any way at any time during the build process.

For the application in which the FIPS Object Module is embedded:

1. A record of the HMAC SHA-1 digest of the **`fipscanister.o`** that was embedded in the application.
2. An assertion that the application does not utilize any cryptographic implementations other that those provided by the FIPS Object Module or contained in the FIPS capable OpenSSL 1.0.1 libraries (where non-FIPS algorithms are disabled in FIPS mode).
3. A description of how the application clearly indicates when FIPS mode is enabled (assuming that FIPS mode is a runtime selectable option). Note that the application must call **`FIPS_mode_set()`**, whether that call is triggered by runtime options or not.

5.6 When is a Separate FIPS 140-2 Validation Required?

When a decision is made on whether a particular IT solution is FIPS 140-2 compliant, multiple factors need to be taken into account, including the FIPS Pub 140-2 standard, FIPS 140-2 Derived Test Requirements, CMVP FAQ and Implementation Guidance. The ultimate authority in this process belongs to the CMVP. The CMVP provides its current interpretations and guidelines as to the interpretation of the FIPS 140-2 standard and the conformance testing/validation process on its public web site http://csrc.nist.gov/cryptval/.

In particular, the only official document known to us which discusses use of embedded cryptographic modules is the CMVP FAQ available at http://csrc.nist.gov/groups/STM/cmvp/documents/CMVPFAQ.pdf. This FAQ (Frequently Asked Questions document) discusses incorporation of another vendor's cryptographic modules in a subsection of Section 2.2.1 entitled *"Can I incorporate another vendor's validated cryptographic module"*. In particular, the following is specified:

> *"Yes. A cryptographic module that has already been issued a FIPS 140-1 or FIPS 140-2 validation certificate may be incorporated or embedded into another product. The new product may reference the FIPS 140-1 or FIPS 140-2 validated cryptographic module so long as the new product does not alter the original validated cryptographic module. A product which uses an embedded validated cryptographic module cannot claim itself to be validated; only that it utilizes an embedded validated cryptographic module. There is no assurance that a product is correctly utilizing an embedded validated cryptographic module - this is outside the scope of the FIPS 140-1 or FIPS 140-2 validation."*

Note that the CMVP FAQ does specify that a FIPS 140-1/2 validated module may be incorporated into another product. It then specifies that making a decision on whether a product is correctly utilizing an embedded module is outside of the scope of the FIPS 140-1 or FIPS 140-2 validation.

A subsection of Section 2.1 of the CMVP FAQ entitled *"A vendor is selling me a crypto solution - what should I ask?"* states:

> *"Verify with the vendor that the application or product that is being offered is either a validated cryptographic module itself (e.g. VPN, SmartCard, etc) or the application or product uses an embedded validated cryptographic module (toolkit, etc). Ask the vendor to supply a signed letter stating their application, product or module is a validated module or incorporates a validated module, the module provides all the cryptographic services in the solution, and reference the modules validation certificate number."*

It is specified that the module provides "all the cryptographic services in the solution". It is not specified that the module provides "all the security-relevant services in the solution". A typical IT product may provide a variety of services, both cryptographic and non-cryptographic. A network protocol such as SSH or TLS provides both cryptographic services such as encryption and network services such as transmission of data packets, packet fragmentation, etc.

The FIPS 140-2 standard is focused on the cryptography. There are many generic security relevant functionalities such as anti-virus protection, firewalling, IPS/IDS and others which are not currently covered by the FIPS 140-2 standard. An anti-virus solution which uses a cryptographic module for its operations can satisfy requirements of the FIPS 140-2 by delegating its cryptographic functions to an embedded FIPS 140-2 validated module. Including the entire anti-virus solution in the FIPS 140-2 validation would hardly improve the overall security since FIPS 140-2 does not currently have requirements in the field of anti-virus protection. In a similar fashion, the FIPS 140-2 standard does not currently have requirements related to network vulnerabilities or denial of service attacks.

Validated modules typically provide algorithm implementations only, no network functionality such as IPSec, SSH, TLS etc. This does not, for example, prevent Microsoft Windows from providing IPSec/IKE and TLS/SSL functionality. Therefore, for example, an OpenSSH based product properly using the OpenSSL FIPS Object Module would not differ from Microsoft using its Microsoft Kernel Mode Crypto Provider in Microsoft IPSec/IKE client which is shipped with every copy of Windows. If an application product delegates all cryptographic services to a validated module the entire product will be FIPS compliant.

Since the CMVP does not have a formal program for validation of IT solutions with embedded FIPS 140-2 modules, the question is one of how the actual compliance/non-compliance is determined. In practice the compliance is determined by the federal agency/buyer selecting the solution. During the process the customer may contact the CMVP, testing labs or security experts for an opinion. In many cases, though, the buyers make such decisions independently. Here it

should be noted that FIPS 140-2 is only a baseline and each federal agency may establish its own requirements exceeding the requirements of FIPS 140-2. In the particular example of network protocols federal agencies generally do accept networking products (IPSec/TLS/SSH etc.) with embedded FIPS 140-2 validated cryptographic software modules or hardware cards as FIPS 140-2 compliant.

For those vendors desiring a “sanity check” of the compliance status of their OpenSSL FIPS Object Module based product, the OpenSSL Software Foundation (OSF) can perform a review and provide an opinion letter stating whether, based on information provided by the vendor, that product appears to OSF to satisfy the requirements of the OpenSSL FIPS Object Module Security Policy. This opinion letter can include a review by one or more CMVP test labs and/or a OpenSSL team member as appropriate. This opinion letter clearly states that only the CMVP can provide an authoritative ruling on FIPS 140-2 compliance.

5.7 Common Issues and Misconceptions

In the years since the first versions of the OpenSSL FIPS Object Module were validated we've seen new users of the FIPS module struggle with some of the same issues over and over again. Here we attempt to offer some possibly useful advice:

5.7.1 Don't Fight It

Rightly or wrongly, the Security Policy very clearly mandates specific fixed build commands. Normal and natural practice in other contexts is to use build-time configuration options to control aspects of the build process, but that is not an option here. Instead think about the end result you want to accomplish and whether that can be done by any other means. For instance, the default install location can't be specified by the usual `--prefix=` build-time configuration option. But, once created via the canonical commands you can copy the `fipscanister.o` and associated files somewhere else. So, one option is to create a new build system, build the FIPS module with whatever permissions necessary to write to the default `--prefix` location, copy from there to the desired destination, and then discard the build system. Yes, that's a silly waste of time from a technical software developer objective, but you wouldn't be using the FIPS module in the first place on purely technical considerations.

5.7.2 Don't Overthink It

We have seen quite a few software vendors make the mistake of trying to force the FIPS module build process into an in-house configuration management scheme. Our recommendation: don't do that. There is no point in trying to manage the individual source files of the FIPS module source tarball because the canonical build process mandates that you start with the original tarball, `openssl-fips-2.0.tar.gz`, which has a fixed digest and cannot be modified.

Likewise there is no point in constantly rebuilding the FIPS module from source. While legal, as long as the Security Policy build process is followed, there is no benefit to be gained from the generation of multiple binary modules. The source code can never change (the usual reason for a

structured build-from-source process), and per the recommendations in §5.5 each distinct binary FIPS module should be separately tracked.

In lieu of trying to jam the mandated FIPS module build process into an existing elaborate in-house configuration management process, we recommend that the binary FIPS module be generated by hand one time only (per distinct platform) in a solemn documented ceremony, and that the resulting binary files be managed through the formal source/version/configuration control process.

6. Technical Notes

This section has technical details of primary interest to the FIPS module developers and more advanced users. The typical application developer will not need to reference this material.

6.1 DRBGs

With very rare exceptions the internal functioning of the DRBGs is irrelevant to the end user and application software. In FIPS mode DRBGs are transparently used by the OpenSSL RAND API and applications will automatically use them.

Random numbers are critical for the proper operation of cryptographic software and hardware. The DRBG or Deterministic Random Bit Generator is intended as a higher quality replacement for the earlier PRNGs or Pseudo-Random Number Generators and is defined by SP 800-90A.

6.1.1 Overview

The way entropy is gathered and used for the DRBG is part of the FIPS capable OpenSSL so it can be modified outside the context of the FIPS 140-2 validation. The current version is in **`crypto/rand/rand_lib.c`**.

There is a "default DRBG" whose context is accessed using **`FIPS_get_default_drbg()`**. This default DRBG is mapped to the **`RAND_*()`** calls. By default, the FIPS Object Module will use the AES/CTR generator from SP800-90A, Section 10.2, *DRBG Mechanisms Based on Block Ciphers*. The default generator can be overridden by the calling application at runtime via the function **`RAND_set_fips_drbg_type()`**. The default is equivalent to CTR_DRBG using AES with a 256 bit key and a derivation function.

The actual default DRBG type can also be specified via a preprocessor macro when the "FIPS capable" OpenSSL is built:

```
#ifndef OPENSSL_DRBG_DEFAULT_TYPE
#define OPENSSL_DRBG_DEFAULT_TYPE       NID_aes_256_ctr
#endif
#ifndef OPENSSL_DRBG_DEFAULT_FLAGS
```

```
#define OPENSSL_DRBG_DEFAULT_FLAGS      DRBG_FLAG_CTR_USE_DF
#endif
```

This might be useful in environments where some DRBG type is mandated by local policy. For example, to use the HMAC DRBG with sha256 by default:

```
./config -DOPENSSL_DRBG_DEFAULT_TYPE=NID_hmacWithSHA256 \
           -DOPENSSL_DRBG_DEFAULT_FLAGS=0 (other options)
```

The **RAND_add()** function just seeds the OpenSSL non-standard PRNG and does not feed into the DRBG directly. However that function would be used if the DRBG was reseeded. The reason it does this is that the DRBG design does not permit the addition of "out of band" entropy; the addition of entropy needs to be combined with a generate operation (additional input) or a full reseed/reinstantiate (which would require the minimum entropy). Environments with a better source of entropy (e.g. fast hardware RNG) could do far better.

The entropy callbacks are completely under application control so the calling application can override the ones provided by default. They can be set by supplying a callback function to **FIPS_drbg_set_callbacks()** after calling **OPENSSL_init()**. This callback function is invoked whenever the DRBG requires additional entropy:

```
size_t (*get_entropy)(DRBG_CTX *ctx, unsigned char **pout,
          int entropy, size_t min_len, size_t max_len)
```

A call to this function requests **entropy** bits of entropy in a buffer of between **min_len** and **max_len** size bytes inclusive. The values of these are mechanism specific and taken from SP800-90 tables. This callback should then return the amount of data in the buffer ***pout** and the length in the return value, or zero in case of being unable to retrieve sufficient entropy.

Once this call completes successfully the DRBG is instantiated at the appropriate (maximum) security strength again taking values from SP800-90 and SP800-57.

We request random data from the caller of sufficient entropy for the security level of the DRBG.

When asymmetric algorithms are used (key generation, parameter generation and indeed signing for DSA/ECDSA) we check that the RNG has sufficient security strength (as dictated by the relevant standards) to perform the operation. Insufficient security strength is an error and the operation cannot be performed.

There is a mechanism, "entropy draining", which causes the DRBG to automatically reseed after a certain number of uses. See SP800-90 for details of how this operates. The function **FIPS_drbg_set_reseed_interval()** can be used to modify the number of calls before auto reseeding.

The function **FIPS_rand_strength()** returns the security strength of the default RNG (the one used for key generation et. al.).

Individual operations (for example key generation) then check the security strength of the RNG and return a fatal error if there is insufficient security strength to complete the operation. The values used are from SP800-57.

This check is performed by the following functions:

```
fips_check_dsa_prng()
fips_check_rsa_prng()
fips_check_ec_prng()
```

Currently there is no equivalent for DH. One could be added if required but it isn't clear how the strengths should be compared when PKCS#3 DH is used.

There is no version for ECDH either but the only operation performed by that code (shared secret computation) does not make use of the RNG.

By default the health checks are automatically performed every 2^{24} generate operations; this count can be modified (up or down) by the calling application via the **FIPS_drbg_set_check_interval()** function. If a DRBG health check fails then the DRBG is placed in an error state that can be cleared by uninstantiating and reinstantiating the DRBG.

For the CTR DRBG a flag allows the optional use of a derivation function. Note the DRBG is always instantiated at maximum security.

6.1.2 The DRBG API

All DRBG operations are performed through an opaque **DRBG_CTX** structure which corresponds to an SP800-90 "instance". The function

```
DRBG_CTX *FIPS_drbg_new(int type, unsigned int flags);
```

allocates and initializes a new **DRBG_CTX** structure for DRBG. The "**type**" and "**flags**" parameters determine the mechanism and primitives used and the security strength. Only the maximum security strength is supported for each type: i.e. it is not possible to instantiate the DRBG at lower than the maximum strength.

In addition to type specific values the "**flags**" field can be set to **DRBG_FLAG_TEST** to enable "test mode". This mode disables periodic health checks and the continuous PRNG test. It is used for internal purposes and to support algorithm validation testing. This flag MUST NOT be set for a live instance.

Before a valid **DRBG_CTX** is returned to the application an extensive health check is performed on a DRBG using the same mechanism and primitives. If the check fails an error is returned.

If the type parameter is set to 0 an uninitialized DRBG structure is returned. This structure may be initialized by calling **FIPS_drbg_init()**. This function returns a valid **DRBG_CTX** structure if it succeeds or NULL if it fails (for example a invalid type parameter).

DRBG Characteristics

All four DRBGs defined by SP800-90 are implemented. The mechanisms, parameters and strength are summarized below:

Hash DRBG

The type parameters **NID_sha1, NID_sha224, NID_sha256, NID_sha384** and **NID_sha512** select the hash DRBG and the corresponding hash primitive. The SHA1 Hash DRBG has a security strength of 128 bits, the SHA224 DRBG has a security strength of 192 bits and all others 256 bits.

HMAC DRBG

The type parameters **NID_hmacWithSHA1, NID_hmacWithSHA224, NID_hmacWithSHA256, NID_hmacWithSHA384** and **NID_hmacWithSHA512** select the HMAC DRBG mechanism and associated hash primitive. Security strengths are the same as for the Hash DRBG.

CTR DRBG

The type parameters **NID_aes_128_ctr, NID_aes_192_ctr** and **NID_aes_256_ctr** select the CTR DRBG type using AES and the appropriate key length. TDES is not supported. The security strength matches the number of bits in the key. For this DRBG type the flag **DRBG_FLAG_CTR_USE_DF** is supported which enables the use of a derivation function. If this flag is not set a derivation function is not used.

Dual EC DRBG

The type parameter is of the form ***(curve << 16) | hash***. The curve value **NID_X9_62_prime256v1** corresponds to the curve P-256, **NID_secp384r1** to P-384, and **NID_secp521r1** to P-521. The hash value should be set to the same value as for the Hash DRBG. Thus **(NID_secp384r1 << 16) | NID_sha224** corresponds to P-384 with hash SHA-224. As indicated in SP800-90 SHA1 can only be used with P-256 and SHA-224 cannot be used with P-521. P-256 has a security strength of 128 bits, P-384 192 bits and P-521 256 bits.

Note: Due to widely reported serious vulnerabilities Dual EC DRBG has been removed from recent versions of the OpenSSL FIPS module.

General Functions

FIPS_drbg_init

The function

```
int FIPS_drbg_init(DRBG_CTX *dctx, int type,
                   unsigned int flags);
```

initializes a pre-existing DRBG_CTX. This is an efficiency measure to avoid the need to reallocate a new **DRBG_CTX**. This function returns 1 for success and zero or a negative value for failure. The return value -2 is used to indicate an invalid or unsupported type value. The type value cannot be 0. This function is otherwise identical to **FIPS_drbg_new()**.

FIPS_drbg_free

The function

```
void FIPS_drbg_free(DRBG_CTX *dctx);
```

frees up a **DRBG_CTX**. After this call the **DRBG_CTX** pointer is no longer valid. The underlying DRBG is first uninstantiated.

FIPS_drbg_set_callbacks

The function

```
int FIPS_drbg_set_callbacks(DRBG_CTX *dctx,
     size_t (*get_entropy)(DRBG_CTX *ctx, unsigned
          char**pout, int entropy, size_t min_len, size_t
          max_len),
     void (*cleanup_entropy)(DRBG_CTX *ctx, unsigned char
          *out, size_t olen), size_t entropy_blocklen,
          size_t (*get_nonce)(DRBG_CTX *ctx, unsigned char
          **pout, int entropy, size_t min_len, size_t
          max_len),
     void (*cleanup_nonce)(DRBG_CTX *ctx, unsigned char *out,
     size_t olen)
     );
```

sets entropy and nonce callbacks for a **DRBG_CTX**. The **DRBG_CTX** must be in an uninstantiated state to set the callbacks: i.e. the callbacks cannot be set on an instantiated DRBG. This function is typically called immediately following **FIPS_drbg_new()**. This function returns 1 for success and 0 if an error occurred: the only way an error can occur is if an attempt is made to set the callbacks of an instantiated DRBG.

Whenever the DRBG requires entropy or a nonce the corresponding callbacks will be called.

The callbacks **get_entropy** and **get_nonce request** "**entropy**" bits of entropy in a buffer of between **min_len** and **max_len** bytes. The function should set ***pout** to the buffer containing the entropy and return the length in bytes of the buffer.

If the source of entropy or nonce is unable to satisfy the request it MUST return zero. This will place the DRBG in an error condition due to the source failure.

The callbacks **cleanup_entropy** and **cleanup_nonce** are called after the entropy or nonce buffers have been used and can be utilized to zeroize the buffers. The "**out**" and "**olen**" parameters contains the same value returned by the get function.

The "**entropy_blocklen**" is used to specify the block length of the underlying entropy source. This is used for the continuous RNG test on the entropy source.

FIPS_drbg_instantiate

The function

```
int FIPS_drbg_instantiate(DRBG_CTX *dctx,
     const unsigned char *pers, size_t perslen);
```

instantiates a DRBG with the supplied personalization string **pers**. This function returns 1 for success and 0 for failure.

If the personalization string is of an invalid length for the DRBG mechanism a non-fatal error is returned.

Internally this function instantiates the DRBG. This will request entropy and a nonce using the supplied get_entropy and get_nonce callbacks.

There are no security strength and prediction resistance arguments to this function. The DRBG is always instantiated at the maximum strength for its type and prediction resistance requests are always supported.

This function returns 1 for success and 0 for failure.

FIPS_drbg_reseed

The function

```
int FIPS_drbg_reseed(DRBG_CTX *dctx, const unsigned char
     *adin, size_t adinlen);
```

reseeds the DRBG using optional additional input "**adin**" of length "**adinlen**".

If the additional input is of an invalid length for the DRBG mechanism a non-fatal error is returned.

The **get_entropy** callback of the DRBG is called internally to request entropy.

An extensive health check is performed on a DRBG of the same type before reseeding the DRBG. If this fails the DRBG is placed in an error condition and the caller must un-instantiate and re-instantiate the DRBG before attempting further calls.

This function returns 1 for success and 0 for failure.

FIPS_drbg_generate

The function

```
int FIPS_drbg_generate(DRBG_CTX *dctx, unsigned char *out,
     size_t outlen,int prediction_resistance,
     const unsigned char *adin, size_t adinlen);
```

attempts to generate "**outlen**" bytes of random data from the DRBG. Using optional additional input "**adin**" of length "**adinlen**". If the "**predication_resistance**" parameter is non-zero a prediction resistance request will be made and internal reseeding of the DRBG and requesting entropy as required by SP800-90 is performed.

If an attempt is made to request too much data for a single request or to supply additional input of an invalid length a non-fatal error is returned.

If an internal request for entropy fails a fatal error occurs and the DRBG is placed in an error state. The caller must un-instantiate and re-instantiate the DRBG before attempting further calls.

This function returns 1 for success and 0 for failure.

FIPS_drbg_uninstantiate

The function

```
int FIPS_drbg_uninstantiate(DRBG_CTX *dctx);
```

uninstantiates a DRBG. This zeroizes any CSPs and returns the DRBG to an uninitialized state.

FIPS_drbg_get_app_data, FIPS_drbg_set_app_data

The two functions

```
void *FIPS_drbg_get_app_data(DRBG_CTX *ctx);
void FIPS_drbg_set_app_data(DRBG_CTX *ctx, void *app_data);
```

get and retrieve an application defined pointer value. The meaning of this pointer is application defined and might for example contain a pointer to a handle representing the entropy source and the **get_entropy** function could retrieve and make use of that pointer.

FIPS_drbg_get_blocklength

The function

```
size_t FIPS_drbg_get_blocklength(DRBG_CTX *dctx);
```

returns the block length of the DRBG.

FIPS_drbg_get_strength

The function

```
int FIPS_drbg_get_strength(DRBG_CTX *dctx);
```

returns the security strength of the DRBG in bits.

FIPS_drbg_set_reseed_interval

The function

```
void FIPS_drbg_set_reseed_interval(DRBG_CTX *dctx, int
     interval);
```

modifies the reseed interval value. The default is 2^{24} blocks for the Dual EC DRBG and 2^{24} generate operations for all other types. These values are lower than the maximums specified in SP800-90.

RAND interface

Cryptographic operations make use of the OpenSSL RAND PRNG API to request random data. A brief description of this is given below:

```
int  RAND_bytes(unsigned char *buf,int num);
```

Generate **num** random bytes and write to **buf**.

```
int  RAND_pseudo_bytes(unsigned char *buf,int num);
```

Generate **num** random bytes and write to **buf**. The random data does not have to be cryptographically strong.

```
void RAND_seed(const void *buf,int num);
```

This function is used at various points to add data which may be useful for adding entropy to the PRNG. The buffer **buf** contains **num** bytes which can be used.

```
void RAND_add(const void *buf,int num,double entropy);
```

This is similar to **RAND_seed()** except that the data supplied has entropy "**entropy**".

Default DRBG

A special DRBG instance called the "default DRBG" is used to map the DRBG to the RAND interface.

The function

```
int FIPS_drbg_set_rand_callbacks(DRBG_CTX *dctx,
     size_t (*get_adin)(DRBG_CTX *ctx, unsigned char
          **pout),
     void (*cleanup_adin)(DRBG_CTX *ctx, unsigned char
          *out, size_t olen),
     int (*rand_seed_cb)(DRBG_CTX *ctx, const void *buf,
          int num),
     int (*rand_add_cb)(DRBG_CTX *ctx,
          const void *buf, int num, double entropy)
```

```
);
```

defines various callbacks which control how RAND calls are mapped to DRBG calls.

The **get_adin** callback is used to retrieve optional additional data used whenever a request for random data is made using **RAND_bytes()** or **RAND_pseudo_bytes()**. When this operation is complete **cleanup_adin** is called to release the buffer.

Note that **RAND_bytes()** and **RAND_pseudo_bytes()** are equivalent operations for the RAND mapping to the DRBG; they both call **FIPS_drbg_generate()**. The **FIPS_drbg_generate()** function can be called multiple times to satisfy a single request if the **num** value exceeds the amount of data that can be handled in a single DRBG request.

The callbacks **rand_seed_cb** and **rand_add_cb** are called directly whenever **RAND_seed()** or **RAND_add()** are called. These are entirely application defined and could be used for example to add seed information to the entropy source.

The function

```
DRBG_CTX *FIPS_get_default_drbg(void);
```

retrieves the default DRBG context. This can then be manipulated using the standard DRBG functions such as **FIPS_drbg_init()**.

The function

```
int FIPS_rand_strength(void);
```

returns the security strength in bits of the default PRNG.

DRBG Health Checks

The function

```
int FIPS_drbg_health_check(DRBG_CTX *dctx);
```

initiates a health check on the DRBG. In addition health checks are also performed when a DRBG is first initiated (using **FIPS_drbg_new()** or **FIPS_drbg_set()**) when a DRBG is reseeded explicitly using **FIPS_drbg_reseed()** and every **health_check_interval** calls to the generate function. This interval is by default 2^{24} but can be modified by:

```
void FIPS_drbg_set_check_interval(DRBG_CTX *dctx, int
     interval);
```

If any health check fails the DRBG is placed in an error state and no further operations can be performed on the DRBG instance until it has been reinitialized (uninstanstiated and initialized).

Extended KAT of all DRBG Functions

The function **`fips_drbg_single_kat()`** performs an extended Known Answer Test (KAT) of all functions:

1. Instantiate DRBG with known data (entropy, nonce, personalization string).
2. Perform generate operation without prediction resistance and check output matches expected value.
3. Reseed with known data (entropy, additional input).
4. Perform second generate operation without prediction resistance and check output matches expected value.
5. Uninstantiate DRBG.
6. Instantiate DRBG in test mode with known data (entropy, nonce, personalization string).
7. Perform generate operation with prediction resistance and check output matches expected value set known entropy and additional input during this step.
8. Perform second generate operation with prediction resistance and check output matches expected value.
9. Uninstantiate DRBG.

It is asserted that checking the output of the generate function in steps 2, 4, 7 and 8 ensures the previous operations completed successfully: i.e. set the DRBG internal state to the expected values.

Extended Error DRBG Checking

Extended error checking is performed by function **`fips_drbg_error_check()`**:

Invalid parameters are fed into all DRBG functions in sequence as follows. Note that some tests (e.g. entropy source failure) leave the test DRBG in an error state and it has to be uninstantiated and instantiated again to clear the error condition.

1. Instantiate with invalid personalization string length.
2. Instantiate DRBG with entropy source failure (returning zero entropy).
3. Attempt to generate DRBG output from DRBG from step 2.
4. Instantiate DRBG with too short an entropy string.
5. Instantiate DRBG with too long an entropy string.
6. Instantiate DRBG with too small a nonce (if nonce used in mechanism).
7. Instantiate DRBG with too large nonce (if nonce used in mechanism).
8. Instantiate DRBG with good data and generate output, check calls succeed.
9. Attempt to generate too much output for a single request.

10. Attempt to generate with too large additional input.
11. Attempt to generate with prediction resistance and entropy source failure.
12. Set reseed counter to reseed interval, generate output and check reseed has been performed.
13. Test explicit reseed operation with too large additional input.
14. Test explicit reseed with entropy failure.
15. Test explicit reseed with too large entropy string.
16. Test explicit reseed with too small entropy string.
17. Uninstantiate DRBG: check internal state is zeroed.

Health Checking Performance

The health checks are performed:

1. When a DRBG is first initialized (i.e. before instantiation) a health check is performed on a test instantiation using the same mechanism and parameters.

2. When a reseed operation is performed (other than for a prediction resistance request) a health check is performed on a test instantiation. This effectively performs a superset of the requirements for testing the reseed function i.e. it tests *all* functions including the reseed function.

3. An internal counter determines the number of generate operations since the last health check. When this reaches a preset limit a health check is performed. This limit is set by default to 2^{24} operations but it can be set to an alternative value by an application.

4. When an application explicitly requests a health check with the call **`FIPS_drbg_health_check()`**.

Abbreviated POST

During a Power On Self Test (POST) an abbreviated KAT only is performed for one instance of each supported DRBG mechanism. This simply performs an instantiate and generate operation and checks that the output matches an expected value.

6.2 Role Based Module Authentication

Summary

A role based authentication mechanism is implemented for the purpose of satisfying a U.S. Army procurement policy. The implementation is transparent to end users if the relevant default build options are left undisturbed, as should generally be the case.

IMPORTANT NOTE:

After the role based authentication mechanism described in this section was implemented, we learned that the original Army policy that motivated this implementation is no longer in effect, as confirmed in correspondence dated 2011-10-28 with CIO/G6 that yielded this statement:

> As a result of the Army's transition to the DoD Unified Capabilities Approved Products List (UC APL), both the Army's IA Approved Products List process and the May 21, 2009 "Letter to Industry were rescinded. (CIO/G-6 memorandum dated May 2011). The above referenced document in its entirety, including paragraph 5a, no longer applies.
> For more details please send inquires to: ArmyIATools@conus.army.mil, 703-545-1677 or 703-545-1672.

Background

The FIPS module is a software library so the concept of authentication to the module doesn't make any sense. For a Level 1 validation the CMVP does not require any module authentication, and there is no circumstance that we can envision for which such authentication would have any practical value for vendors or users. A little thought shows that authentication of a general purpose cryptographic library itself must necessarily be a pointless nuisance; consider for instance the vendor of a Linux distribution (Ubuntu, Red Hat, etc.) that elected to utilize authentication with the OpenSSL libraries. Such an OS distribution will typically default to dozens of individual applications utilizing those libraries, with dozens to hundreds more available as optional packages. Each and every one of those applications would have to contain the correct authentication credentials at all times. Application vendors would either have to be informed of those credentials, widely and publicly, or would be forced to ship their product with unauthenticated OpenSSL libraries (or libraries authenticated with different known credentials) to avoid the failures that would be caused by mismatched credentials. The result would be a mess that would provide more opportunities than obstacles to Evil Hackers.

However, in 2009 the U.S. Army specified a set of acquisition requirements, in the form of a memo with a subject line of "*Letter to Industry Concerning the Approval and Acquisition of Information Assurance (IA) Tools and Products in the United States Army*" (see https://chess.army.mil/ascp/commerce/scp/downloads/standardspolicy_files/letter_to_industry.pdf). This mandate imposes additional requirements for FIPS 140-2 validated products, beyond those mandated by the CMVP. In particular, for Level 1 validations such as ours, it requires:

> *5. Federal Information Processing Standards (FIPS):*
>
> *a. FIPS 140-2, Level 1: This applies to cryptographic modules that are software only solutions (the software cannot be bundled or sold as a hardware-software solution) that are unable to achieve FIPS 140-2 Security Level 2. Overall FIPS 140-2 Level 1 solutions must incorporate the following Cryptographic Modules Specifications to a higher security level: Roles, Services, and Authentication (Security Level 2) and Design Assurance (Security Level 3).*

The OpenSSL FIPS Object Module 2.0 validation cannot be at overall Level 2 because such a validation would necessarily tie the module to specific hardware. This Army policy was evidently directed at turnkey appliances (firewalls, mobile devices, etc.) and failed to consider the case of general purpose cryptographic libraries.

The earlier v1.2.3 FIPS module (certificate #1051) predated the release of the Letter to Industry, and since then we've heard from quite a few software vendors who have experienced difficulty in selling to the Army because the v1.2.3 module didn't meet the 5a requirement. It turns out that satisfying this requirement is easily handled at modest cost as a pure documentation effort in some contexts, such as when the test platforms have Common Criteria (CC) certified operating systems or the module itself actually implements authentication.

However, the CMVP takes the not unreasonable position that validation at Roles, Services, and Authentication at Level 2 is not appropriate unless authentication actually takes place (note that in this context a non-CC certified operating system is considered to provide no authentication services). CC certified platforms are few and far between, and it makes no sense to implement authentication to a general purpose cryptographic library.

So, that left us with a bit of a dilemma. The CMVP and Army policies are in direct conflict, and if we knew of any easy way to get two government bureaucracies to reconcile conflicting policies we'd tackle some easier challenges like brokering a permanent peace in the Middle East.

After some deliberation and consultation with the test lab we concluded that the best resolution to this dilemma was to implement role-based authentication in a way that would satisfy both the CMVP and Army requirements without significantly impacting the end users. This goal was accomplished by requiring role based authentication for use of the module in FIPS mode, and then automatically and transparently performing that authentication in the "FIPS capable" OpenSSL. The end result is that the FIPS module plus "FIPS capable" OpenSSL combination -- by far the most common use of the FIPS module -- will behave for the calling application as if the role based authentication were not required.

Note we already have a well established precedent for publishing secret credentials in the context of an open source based validation. The integrity test mandated by FIPS 140-2, which is accorded great significance, requires a HMAC-SHA1 digest of the module contents (object code, roughly speaking). The HMAC digest is calculated from a secret HMAC key plus the data of interest, the purpose being to allow both authentication and confirmation of data integrity (only the entity knowing the secret key can generate the correct digest). For the very first validation we were faced with the challenge of where to store the secret HMAC key, as in open source code there is no suitable hiding place. After some deliberation the CMVP instructed us to just code the HMAC-SHA1 digest as mandated and leave the secret key exposed in the source code. That same "secret" key has been in every validation since and is published in the corresponding Security

Policy documents (Appendix B, it is **65 74 61 6f 6e 72 69 73 68 64 6c 63 75 70 66 6d**, equivalent to the ASCII string "**etaonrishdlcupfm**").

Implementation

The **FIPS_mode_set()** function familiar to users of past versions of the OpenSSL FIPS Object Module is now defined in the "FIPS capable" OpenSSL, i.e. externally to the FIPS module. The corresponding function in the FIPS module that enables the FIPS mode of operation requires role based authentication in the form of a password argument. Note that FIPS 140-2 requires at least two roles; we defined two roles but both perform identically in all respects.

The build process for the FIPS module references three environment variables, with defaults if not explicitly set:

```
FIPS_AUTH_KEY
FIPS_AUTH_CRYPTO_OFFICER
FIPS_AUTH_CRYPTO_USER
```

These environment variables define the HMAC key and the HMACs of the passwords respectively. This are utilized during the standard:

```
./config
make
```

The **FIPS_AUTH_KEY** defines the HMAC key which defaults to **"etaonrishdlcupfm".** The two passwords default to "**Default FIPS Crypto Officer Password**" and "**Default FIPS Crypto User Password**" respectively and appear in **fips/fips_utl.h**.

There are several ways to get the right format for the password HMACs, such as:

```
echo –n <password> | openssl sha1 –hmac <hmac_key>
```

At runtime the calling application invokes **FIPS_module_mode_set(1, password)**. Internally this function generates the digest **HMAC(FIPS_AUTH_KEY, password)** and checks to see if that value matches either of **FIPS_AUTH_CRYPTO_OFFICER** or **FIPS_AUTH_CRYPTO_USER**. If the password does not match the error is treated the same as a fatal POST error.

Validation Testing

For use by the test lab in testing the role based authentication the following command line options are defined for the **fips_test_suite** utility, to specify the password value to be passed to **FIPS_module_mode_set()**:

none	Null password
bad	Invalid password of sufficient length
user	The **FIPS_AUTH_CRYPTO_USER** password
officer	The **FIPS_AUTH_CRYPTO_OFFICER** password

If none of those command line options are given the **FIPS_AUTH_CRYPTO_USER** password is used.

Support in the "FIPS capable" OpenSSL

A means is provided in the "FIPS capable" OpenSSL (which is just another application from the perspective of the FIPS module) to specify non-default passwords:

./**config** ***<...options...>*** **-DFIPS_AUTH_USER_PASS="\"*...password...*\""**

Please note this is ***not*** something likely to be of value in any real-world context, and a FIPS module built with non-default passwords is a likely source of problems.

6.3 Self Tests

As required by FSIP 140-2 the FIPS module implements numerous self tests. Typically at least one self test is required for each cryptographic algorithm.

Each test as it is performed can be examined through an optional callback:

```
int (*fips_post_cb)(int op, int id, int subid, void *ex);
```

Unless otherwise stated below the callback should always return 1. The "**op**" parameter indicates the operation being performed and can be one of:

FIPS_POST_BEGIN: indicates that testing has begun but no tests have been performed yet.

FIPS_POST_END: indicates all tests have been completed. The "**id**" parameter indicates the overall status of tests. It is 1 if all tests completed successfully and 0 if at least one test failed.

For the remaining "**op**" values the "**id**", "**subid**" and "**exstr**" parameters indicate details of the specific test being performed. See complete descriptions of each test type for the meaning of these parameters.

FIPS_POST_STARTED: indicates an individual test has started.
FIPS_POST_SUCCESS: individual self test was successful.
FIPS_POST_FAIL: individual self test failed.
FIPS_POST_CORRUPT: a query as to whether self test failure mode should be set.

If the callback returns 0 a failure is simulated for the referenced self test. The method used to simulate failure is documented against each test.

6.3.1 POST Tests

The tests performed during POST are described below, along with the corresponding **fips_test_suite** option(s) to trigger the test (see Appendix B.5).

6.3.1.1 Integrity Test

The **id** field is set to **FIPS_TEST_INTEGRITY**. The remaining parameters are not used. This is indicated while incore integrity testing of the module itself is being performed. This operation performs an HMAC over sections of incore data and checks the value against an expected value set when the application is compiled [see §2.2 for a more comprehensive description of this operation].

If failure is being simulated an additional byte is HMACed in addition to the incore data to produce an HMAC value which will differ from the stored value.

Triggered by the **integrity** option to **fips_test_suite**.

6.3.1.2 DRBG Self Test

The **id** field is set to **FIPS_TEST_DRBG**. The **subid** field is set to the NID of the DRBG being tested and the "**exstr**" field is of type **(int *)** which points to the DRBG flags being tested.

An abbreviated KAT only test (not a full health check) is performed on each supported DRBG mechanism. Specifically, it is initialized in test mode, instantiated using known parameters, output is generated and the result compared with known good values.

If failure is being simulated the "additional input" parameter to the generate operation is perturbed by setting it to a shorter length than the KAT value. This will result in data being generated which does not match the expected value.

Currently the following DRBG mechanisms and primitives are tested as part of the POST:

a) CTR DRBG using 256 bit AES and a derivation function.
b) CTR DRBG using 256 bit AES without a derivation function.
c) Hash DRBG using SHA256.
d) HMAC DRBG using SHA256.
e) Dual EC DRBG using P-256 and SHA-256.

Triggered by the **drbg** option to **fips_test_suite**.

6.3.1.3 X9.31 PRNG Self Test

The **id** field is set to **FIPS_TEST_X931**. The **subid** field is set to the key length of the PRNG in bytes.

For the test the PRNG is set up in test mode. A known key, V (seed) and DT (date time vector) is supplied and the generated output (R) compared to an expected value.

If failure is being simulated the known V value is corrupted by incrementing the first byte. This will result in generated data which does not match the expected value.

Currently the POST tests the X9.31 PRNG using 128, 192 and 256 bit key lengths.

Triggered by the **rng** option to **fips_test_suite**.

6.3.1.4 Digest Test

The **id** field is set to **FIPS_TEST_DIGEST**. The **subid** field is set to the digest NID being tested. The "**ex**" argument is not used. Currently only SHA1 is tested in this way. Known data is digested and the resulting hash compared to a known good value.

If failure is being simulated an extra byte is digested in addition to the known data which will result in a digest which does not match the expected value.

Triggered by the **sha1** option to **fips_test_suite**.

6.3.1.5 HMAC Test

The **id** field is set to **FIPS_TEST_HMAC**. The **subid** field is set to the associate digest NID being tested. The "**ex**" argument is not used.

Known data is HMACed and the resulting hash compared to a known good value.

If failure is being simulated an extra byte is HMACed in addition to the known data which will result in an HMAC which does not match the expected value.

The digests SHA1, SHA224, SHA256, SHA384 and SHA512 are tested in this way.

Triggered by the **`hmac`** option to **`fips_test_suite`**.

6.3.1.6 CMAC Test

The **`id`** field is set to **`FIPS_TEST_CMAC`**. The **`subid`** field is set to the associated cipher NID being tested. The "**`ex`**" argument is not used.

Known data is CMACed and the resulting CMAC compared to a known good value.

If failure is being simulated an extra byte is CMACed in addition to the known data which will result in an HMAC which does not match the expected value.

The triple DES cipher and AES using 128, 192 and 256 bytes is tested for CMAC.

Triggered by the **`cmac`** option to **`fips_test_suite`**.

6.3.1.7 Cipher Self Tests

The **`id`** is field is set to **`FIPS_TEST_CIPHER`**. The **`subid`** field is set to the NID of the cipher being tested, "**`ex`**" is not used.

A known key, IV and plaintext is encrypted and the output ciphertext compared to a known good value.

The ciphertext is then decrypted using the same key and IV and the result compared to the original plaintext.

If a failure is being simulated the ciphertext is corrupted (first byte XORed with 0x1) before the decryption test.

AES in ECB mode with a 128 bit key and triple DES in ECB mode are tested.

Triggered by the **`aes, des`** options to **`fips_test_suite`**.

6.3.1.8 GCM Self Test

The **`id`** is field is set to **`FIPS_TEST_GCM`**. The **`subid`** field is set to the NID of the cipher being tested, "**`ex`**" is not used.

A known key, IV, AAD and plaintext is encrypted and the output ciphertext and tag compared to known good values.

The ciphertext and take is then decrypted using the same key, IV, AAD and expected tag and the result compared to the original plaintext.

If a failure is being simulated the tag is corrupted (first byte XORed with 0x1) before the decryption test.

AES in GCM mode with a 256 key is tested.

Triggered by the **`aes-gcm`** option to **`fips_test_suite`**.

6.3.1.9 CCM Self Test

The **`id`** is field is set to **`FIPS_TEST_CCM`**. The **`subid`** field is set to the NID of the cipher being tested, "**`ex`**" is not used. The test is otherwise identical to the CCM test.

AES in CCM mode with a 192 bit key is tested.

Triggered by the **`aes-ccm`** option to **`fips_test_suite`**.

6.3.1.10 XTS Self Test

The **`id`** field is set to **`FIPS_TEST_XTS`**. The test is otherwise identical to the cipher tests.

AES in XTS mode with a 128 and a 256 bit key is tested.

Triggered by the **`aes-xts`** option to **`fips_test_suite`**.

6.3.1.11 Signature Algorithm Tests

The **`id`** field is set to **`FIPS_TEST_SIGNATURE`**. The **`subid`** field is set to the NID of the associated digest. The "**`ex`**" field is set to the EVP_PKEY structure of the key being used in the KAT. By examining **`exstr`** the type of key being tested can be determined.

A signature is calculated using a known private key and data to be signed.

For deterministic signature algorithms (i.e. RSA in some padding modes) the signature is compared to a known good value.

The signature is then verified using the same data used to create the signature.

If failure is being simulated an extra byte is digested in addition to the known data for signature creation only. This will result in a signature which does not match the expected value (if this test is being performed) or the verification will fail.

The following algorithms are tested:

a) RSA using PSS padding and SHA256 with a 2048 bit key.
b) ECDSA using P-224 and SHA512.
c) ECDSA using K-233 and SHA512 if binary fields are supported.
d) DSA using SHA384 and a 2048 bit key.

Triggered by the **`dsa, ecdsa, rsa`** option to **`fips_test_suite`**.

6.3.12 ECDH Self Tests

The **`id`** field is set to **`FIPS_TEST_ECDH`**. The subid field is set to the NID of the curve used. The "**`ex`**" field is not used.

Known private and public ECDH keys are used to compute a shared secret (Z) value. This is compared to a known good value.

If failure is being simulated the computed shared secret is corrupted after generation. This will result in a mismatch with the expected value.

Triggered by the **`ecdh`** option to **`fips_test_suite`**.

6.3.2 Conditional self tests.

6.3.2.1 Pairwise consistency Test

When an asymmetric signature key is generated a signature test identical to the POST signature tests is performed on the generated key. The only difference is the id field is set to **`FIPS_TEST_PAIRWISE`**.

In the case of RSA keys a consistency test is also performed using an RSA PKCS#1 padding encryption and decryption operation: this operation is not registered with the callback. Specifically: known data is encrypted, the ciphertext checked it does not match the plaintext and then decrypted. The decrypted value is checked against the original plaintext.

For RSA keys the SHA256 digest is used and three tests performed PKCS#1, X931 and PSS padding.

For DSA and ECDSA keys one test using SHA256 is performed.

Triggered by the **dsakeygen** and **rsakeygen** options to **fips_test_suite**.

6.3.2.2 Continuous PRNG Test

When not in test mode (i.e. an operational "live" PRNG) the output of the PRNG is put through the continuous PRNG test for FIPS 140-2.

The callback is not used for this operation.

If the function **FIPS_x931_stick()** is called then the X9.31 PRNG output is copied to the stored last block to ensure the test will fail on the next generate operation.

If the function **FIPS_drbg_stick()** is called then the X9.31 PRNG output is copied to the stored last block to ensure the test will fail on the next generate operation.

The continuous PRNG test for the PRNG itself is triggered by the **drbgstick** and **rngstick** options to **fips_test_suite**. The continuous PRNG test for the entropy source is triggered by the **drbgentstick** option to **fips_test_suite**.

6.4 ECDH

The CAVP defines a test for ECDH in the form of "ECC CDH Primitive" tests:

http://csrc.nist.gov/groups/STM/cavp/#09

When this ECDH testing was introduced for FIPS 140-2 we initially assumed that with the growing use of ECDH in TLS the intent was to ensure that usage was covered by an approved algorithm.

That turns out not to be the case. The algorithm now available for testing is "cofactor ECDH" (formally known as ECC CDH) which is NOT the same as regular ECDH (formally known as as the ECKAS-DH1 scheme) used with TLS -- it is a variant of ECDH that is not the same as that commonly used in actual applications.

The differences between the two algorithms are small but enough to make the two incompatible in subtle ways.

For regular ECDH the shared secret **Z** is the **x** component of the value **dQ** where **d** is one sides private key (an integer) and **Q** the other sides public key (an elliptic curve point).

For cofactor ECDH the shared secret **Z** is the **x** component of the value **hdQ** where the new value **h** is something called the cofactor (another integer) which is a property of the curve. For most primes[47] curves **h = 1** whereas for many binary curves **h ≠ 1**. So for many prime curves (but not all) the two algorithms yield the same result. For binary curves they do not.

Note that the addition of a few lines to the ECDH algorithm implementation changes it to cofactor ECDH at which point it passes the CAVP ECC CDH Primitive test. However, if we change our ECDH implementation to unconditionally use cofactor ECDH then it will not be interoperable with TLS using binary curves.

Even though the use of cofactor ECDH is rare at present, there could conceivably be a need at some point. In order to accommodate that possibility while preserving compatibility with existing applications we added a flag to the **EC_KEY** structure to enable cofactor ECDH for use with the FIPS 140-2 algorithm tests. This flag is set with the **EC_KEY_set_flags()** function:

```
EC_KEY_set_flags(key, EC_FLAG_COFACTOR_ECDH);
```

If this flag it is not explicitly set then the ECKAS-DH1 (TLS compatible) scheme is used.

6.5 ECC and the NSA Sublicense

Why are there two versions of the OpenSSL FIPS Object Module 2.0?

At least some implementations of Elliptic Curve Cryptography (ECC) are perceived to be encumbered in the United States by a complex set of patents. Concern about the possible risks of patent infringement have been a significant disincentive to more widespread use of ECC.

In order to counter such concerns for the ECC necessary to implement the Suite B algorithms, the NSA established a process for sub-licensing the patents for that subset of ECC (see http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml). The OSF has obtained such a sublicense (http://openssl.com/testing/docs/NSA-PLA.pdf).

However, that sublicense only covers the specific patents presumed relevant to the prime curve ECC used for Suite B. It does not cover other possible types of ECC such as binary curves which are implemented in OpenSSL.

Judging the risks of a patent infringement lawsuit is difficult, and not only because the patents themselves are usually incomprehensible to the software developer. The mere threat of a patent

[47]The standard tested prime curves all use h = 1 excepting one non standard prime curve with h != 1; that is a 128 bit curve and so forbidden in approved mode. Effectively this means that for an implementation only checking prime curves (as many do) then the discrepancy would never be apparent. FIPS 140-2 does allow non-standard curves so two "tested" algorithms could yield the different results.

lawsuit can be crippling to even a medium sided enterprise, regardless of the legitimacy of the accusation of infringement.

It is the belief of the OpenSSL team that the implementation of ECC in OpenSSL, both primary and binary curve, does not infringe any patents[48]. However, we aren't lawyers and patent law is notoriously perverse. Some potential users are still concerned about the risk of patent litigation, understandably so given the extent to which such litigation has been used as an offensive commercial tactic in recent years. For the OpenSSL software such users can use built-time options to omit specific algorithms of concern from the resulting binary code.

However, the restrictions of FIPS 140-2 prevent the use of such build-time options or modification of the source code. One of the validation sponsors was concerned about patent risks and so a separate "patent troll" source distribution of the OpenSSL FIPS Object Module 2.0 was created which entirely omits the binary curve ECC. That distribution, `openssl-fips-ecp-2.0.tar.gz`, is functionally identical to the full distribution except for the omission of those algorithms, and all discussion of the full distribution elsewhere in this document applies.

Note that when using the "`ecp`" distributions the corresponding "FIPS capable" OpenSSL must be built with the `no-ec2m` option.

6.6 The "Secure Installation" Issue

This latest of the OpenSSL FIPS Object Module ("FIPS module") FIPS 140-2 validations saw the introduction of a new requirement by the CMVP:

> *The distribution tar file, shall be verified using an independently acquired FIPS 140-2 validated cryptographic module...*

We're told that this distribution tar file verification requirement comes directly from the assertions AS10.03 and AS14.02 of the Derived Test Requirements document:

> *AS10.03: (Levels 1, 2, 3, and 4) Documentation shall specify the procedures for secure installation, initialization, and startup of the cryptographic module.*
>
> *AS14.02: (Levels 1, 2, 3, and 4) The cryptographic module security policy shall consist of: a specification of the security rules, under which the cryptographic module shall operate, including the security rules derived from the requirements of the standard and the additional security rules imposed by the vendor.*

Subsequent discussions mediated by the test lab elaborated this "secure installation" requirement to mean that one of the following conditions must be true:

[48]Also note that the bulk of the binary curve ECC implementation to the OpenSSL project was contributed by a corporation, the former Sun Microsystems, with the legal resources to analyze such risks.

1) The distribution file is obtained via a "trusted path", which is one of:
   a) Transfer via physical media (e.g. CD-ROM disk) sent by postal or delivery service (USPS, UPS, FedEx);
   b) Electronic transfer using cryptography (e.g. SSH, HTTPS, IPsec) implemented by FIPS 140-2 validated products. That requirement was further elaborated to state that those products must themselves be a result of "secure installation".
2) The distribution file is verified (HMAC-SHA-1 digest checked) using a pre-existing FIPS 140-2 validated product that is itself the result of a "secure installation".

Note the recursive nature of the "secure installation" requirement represents a non-trivial challenge; in order to transfer or verify a new validated product an existing securely installed validated product must already be present. We're still struggling to understand the scope and implications of this requirement. The FIPS 140-2 scripture (The FIPS 140-2 standard [Reference 1], the DTR [Reference 4], and the IG [Reference 3] documents) doesn't shed a lot of light -- the term "trusted path" for instance is only referenced in the context of Level 3 validations. Note those "secure installation" and "trusted path" requirements as explained to us say that validated software cannot be distributed by traditional methods, which leads to some interesting questions about the use of other validated modules (puzzlement over why all other modules aren't similarly impacted is a large part of our confusion). Those questions aside, prospective users of this FIPS module need to determine at least one known valid way to satisfy the requirement for this specific validation -- a way not at risk of being ruled invalid by the CMVP after software has been shipped or deployed.

So far the CMVP has declined to answer specific questions about options for satisfying this requirement; they quote the formal documentation (as noted above) and refer us to the test labs. We have actively discussed this issue with several accredited test labs and selected members of the FIPS validation community. Unfortunately the test labs are not in close agreement. So far we have collected a lot of opinions but not much certainty. If you have experience or insights directly relevant to this issue we'd love to hear from you[49].

Very Important Note:

The conclusions presented here are still tentative as they have neither been confirmed nor refuted by the CMVP; they simply represent our best understanding of the situation at this point in time. These conclusions could change dramatically based on relevant feedback from the CMVP, or more slowly in response to an accumulated consensus of opinion from the test labs and FIPS 140-2 community of interest.

6.6.1 What Won't Work

This new requirement doesn't sound so bad until you try to pin down exactly what steps need to be taken to satisfy it. We're still working on figuring this out, but we can eliminate some options that have been considered but which apparently are not allowed:

[49] http://openssl.com/contact.html

- No delegation: one entity (OSF for instance) can't perform the verification of the source tarball and then post that verified tarball on a website for download by everyone else unless the download qualifies as a "trusted path", which in practice will mean the user performing the download will need to obtain and install FIPS 140-2 validated client software (also through a trusted path ... which is a circular problem for many users).
- The new module itself (what is built from the source distribution) cannot be used to perform the verification of the source distribution it was built from.
- Earlier FIPS modules (such as the 1.2.3 FIPS module, validation certificate number #1051) apparently cannot be used to perform the verification. Apparently the new tarball verification requirement will be retroactively applied to the older OpenSSL FIPS Object Module validations. We do not know if that will mean that all deployed instances of these older modules will be declared invalid (that would have a *huge* impact), but the consensus of our discussions is that the older modules can't be leveraged to verify the new module.
- Use of an earlier binary module validation (certificate #1111) was suggested by the CMVP. There are two problems with that suggestion; first, that particular validation took so long (with a 13 month wait for CMVP action) that it had no economic value by the time it was finally completed, and as a result it was abandoned and we no longer have the corresponding binary module; and second, per our understanding that binary module would need to be executed on some very obsolete platforms (OpenSuSE 10.2, no longer downloadable from the maintainer, or Microsoft Windows XP SP2, no longer sold by the vendor). Also in many environments (such as DoD) use of such unsupported operating systems is forbidden by security policy.
- One of our first thoughts was to create (by some means) an executable binary utility program to perform the verification, that could be run on one or more common platforms (e.g. Linux, Windows), and that we could provide publicly for everyone. However, it seems we can't just post that utility for download on a typical web site as the downloaded file would not have been obtained through a "trusted path". Our understanding is that a trusted path over a network would require formally FIPS 140-2 validated software at both the client and server which fails to address the issue of how to get validated cryptography in the first place.
- Another clever idea that was suggested was for us to provide a utility based on a known common commercial validated cryptographic implementation, such as CryptoAPI in Microsoft Windows. The utility could be freely downloaded because it would not contain the actual cryptography. However, many prospective users will have obtained that validated cryptography (the Microsoft Windows OS itself) by non-trusted means (the MSDN download of ISO images does not use FIPS validated cryptography, nor does the usual Internet based update process). Likewise an NSS based utility for Red Hat Enterprise Linux would have the same problem (non-trusted installation and update). Even if the initial OS installation was done with a trusted path, the subsequent routine updates are not[50]; so one

[50]We were able to connect to both Microsoft and Red Hat distribution servers with non-allowed cryptographic algorithms (e.g. RC4); hence we can deduce that those servers are not utilizing FIPS 140-2 validated cryptography.

would have to install the OS using a vendor supplied CD/DVD and then not subsequently update it over the Internet.

Note this last point is downright mind-boggling: it amounts to an assertion that essentially all installations of validated software modules are illegitimate.

Many other options have been considered as well, without a clear consensus from those in the test labs and the community of interest who we have consulted.

6.6.2 What Might Work

The options that we are fairly confident will satisfy the new requirement are:

- Use of a commercial proprietary product using FIPS 140-2 validated cryptography, obtained via a trusted path (e.g. snail-mailed CD or DVD), to display the HMAC-SHA-1 digest of the source tarball. That product should be capable of performing the equivalent of:

  openssl sha1 -hmac etaonrishdlcupfm openssl-fips-2.0.tar.gz

  As noted above, for reasons we don't understand the earlier OpenSSL FIPS Object Module validations (e.g. #1051) are apparently not eligible for this role. At this point we are not aware of any specific commercial products that perform this operation on a file, nor how much they cost or how to purchase them. However, such products must exist. If you know of or find a suitable product please let us know[51] the details.

- Use of a source code distribution that can be obtained from OSF on physical media (a CD-ROM disk) via snail-mail (USPS).

  Note this option is specifically documented[52] as acceptable in the Security Policy itself -- a huge comfort factor for those concerned about the lack of clear guidance in this area. Also note that some experienced and respected commentators in the FIPS 140-2 community of interest that we consulted felt strongly that physical media should *not* constitute a trusted path. However, a direct statement as placed in the Security Policy and approved by the CMVP trumps any such concerns.

  Until and if the postage costs get out of hand we will send those CDs on request at no cost. Please send your request including a full postal address to verifycd@openssl.com. Note that the files you will receive on these CDs will be identical in every respect (except for FIPS 140-2 compliance) with the files you can download from the *openssl.org* web site, so we ask that you only request this CD if you plan to use it for generation of FIPS 140-2 validated cryptography in a context that requires such compliance. The downloaded files are bit-for-bit identical and for any other purposes will generate exactly the same results.

[51] http://openssl.com/contact.html

[52] The discussions leading to this statement in the Security Policy were responsible for several weeks of delay in obtaining the validation. We felt the issue of having one specific affirmatively approved process for satisfying this new requirement was so critical as to warrant any necessary delay; placement of that statement in the Securitiy Policy itself was essentially our only opportunity to obtain a definitive response on the topic from the CMVP.

6.6.3 Still Confused?

Welcome to the club. As we learn more about specific options that will and won't satisfy the requirement we will post that information on the OSF web site and in updates to this document. In the meantime the only definitive answers will have to come from the CMVP itself, either directly or indirectly. The best point of contact is the Director of NIST CMVP[53]. If you choose to contact the CMVP then please:

- Keep all inquiries polite and respectful.
- Remember that the CMVP have a *very* different perspective on computers and software than the average information technology practitioner. They do not have a software development background.
- Note that they are not the enemy; if it was their intent to consciously block or sabotage the OpenSSL FIPS Object Module validations they could have done so easily long ago using a wide range of bureaucratic tactics.
- Note that if you disagree with what you are told by the Director of NIST CMVP you have no recourse to appeal to any higher authority; his word is definitive and final (technically the CMVP is a joint U.S.-Canadian program with the CSE[54] as the Canadian equivalent of NIST, but for U.S. users at least the NIST CMVP opinion is what matters. Canadian users may want to consult the CSE).
- If you learn anything of interest please share it with us[55] and/or one of the OpenSSL mailing lists[56].

6.7 GMAC

The FIPS module was originally tested with, and awarded an algorithm validation for, AES GCM including GMAC. The CAVP subsequently revised the algorithm and retroactively designated a number of validations, including our, as "GMAC not supported"

6.7.1 CAVP Action

We first heard of this in an E-mail forwarded by our test lab, at which time the CAVP and CMVP web site listings had already been updated to show "GMAC not supported" for multiple validations.

The CAVP noted that our GCM implementation gave an incorrect answer when a zero length plaintext is given with an AAD input length that is not a multiple of 128 bits. The original GMAC test only checked input lengths that were a multiple of 128 bits.

[53] http://csrc.nist.gov/groups/STM/cmvp/contacts.html
[54] http://www.cse-cst.gc.ca/index-eng.html
[55] http://openssl.com:/contact.html
[56] http://openssl.org/support/community.html

Note this preemptive action appears to be a little unusual, typically the CAVP/CMPV will contact a vendor to discuss problems before taking unilateral action.

6.7.2 Options for Addressing

The fix is a trivial one line code change, http://cvs.openssl.org/chngview?cn=22745, which has been applied to the regular OpenSSL releases. However, changes to FIPS 140-2 validated software, no matter how trivial, are not easily effected. In this case the CAVP insisted on retesting of all of the 50 some previously tested platforms.

Retesting was not economically feasible due to multiple factors:

- Many test devices had already been returned to the platform sponsors. Some of those were one-off prototype or evaluation units and arranging with the sponsors to re-ship that equipment to the OSF test lab would have taken a substantial amount of time and effort. Even shipping costs themselves were non-trivial, as OSF pays return shipping for customer supplied equipment. Those costs alone were several thousand dollars for the initial 2.0 FIPS module testing.

- Many man-weeks of effort would have been required to repeat the process of installing and configuring each test device and then running the software build and execution process.

- We would have to pay the test lab for the testing, a very substantial cost. Even with negotiations to take into account the fact that the testing process was already fully documented and tested for each device, that cost would probably have been at least US$50,000.

All told we estimated the cost of retesting every platform would exceed US$70,000 even with OSF personnel working for minimum wage.

Fortunately the practical impact of removing GMAC from the 2.0 module validation appears to be minimal, as discussed in the following section.

This incident does illustrate the risk of unpredictable and unilateral CAVP/CMVP action. Passing all the formal testing and receiving a validation award is no guarantee that the validation will not disappear overnight[57]. That perceived risk is a large part of the appeal of the "private label" validations for risk-adverse clients.

6.7.3 Practical Impact

[57]That has happened before, for instance the earlier OpenSSL FIPS Object Module validation #733 which was effectively revoked by the CMVP. See http://veridicalsystems.com/blog/index.html?p=55.html for a discussion of that incident.

The AES-GCM algorithm is an authenticated encryption algorithm. It is in some ways equivalent to the separate HMAC and encryption algorithms used in some ciphersuites. It is an attractive choice because it does everything all in one go and thus is is considerably faster than the separate encryption+MAC operation. The first widespread use of GCM is in TLS 1.2 in new ciphersuites.

AES-GCM as its input can take (among other things) some additional authenticated data (AAD) and plaintext (in encrypt mode). Its output is ciphertext and a MAC.

The AAD is used as some additional data to throw into the MAC calculation but it does not appear in the output. The ciphertext is the encrypted plaintext.

If there is any plaintext/ciphertext at all then the operation is called GCM, with or without AAD.

If there is no ciphertext/plaintext and only AAD then the operation is called GMAC. So GMAC is a special case of GCM.

The bug in the FIPS module GCM implementation is triggered when GMAC is used, i.e. there is no ciphertext/plaintext and *only* AAD. Also the bug is not manifested unless the AAD is not a multiple of 16 bytes.

So if the AAD is a multiple of 16 bytes *and/or* there is any ciphertext/plaintext then the FIPS module implementation works just fine.

During normal operation of the TLS protocol GMAC is not used because there is always some data to encrypt or decrypt. The degenerate case of a zero length fragment we *think* could trigger this but OpenSSL never produces such a thing and there is no reason for a non-OpenSSL TLS stack to do so either. Further review many be needed to determine if a TLS 1.2 zero length fragment case is even theoretically possible.

So to summarize: under any normal use cases the OpenSSL TLS implementation works in FIPS mode just fine without GMAC.

6.8 DH

The version of DH used by TLS is a variant on PKCS#3 and not the X9.42 specification, and hence is not compliant with SP800-56A. For example, the requirement:

> *Each private key shall be unpredictable and shall be generated in the range [1, q-1] using an Approved random bit generator.*

For TLS clients that requirement cannot be satisfied as stated because the parameter "q" is not sent from server to client, only the parameter "p". Clients generate a private key in the range [1, p-1] instead.

6.9 DSA

The DSA private key value is calculated as follows:

The function `fips_check_dsa_prng()` checks parameters and that the PRNG strength is consistent with them when a private key is generated. The function `fips_ffc_strength()` which takes the values directly from SP800-131A is used as well.

7. REFERENCES

1. *OpenSSL FIPS 140-2 Security Policy*, Version 2.0, Open Source Software Institute. This document is available at http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140spNNNN.pdf and http://www.openssl.org/docs/fips/.

2. FIPS PUB 140-2, *Security Requirements for Cryptographic Modules*, May 2001, National Institute of Standards and Technology, available at http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

3. *Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*, January 26, 2007, National Institute of Standards and Technology, available at http://csrc.nist.gov/cryptval/140-1/FIPS1402IG.pdf.

4. *Derived Test Requirements [DTR] for FIPS PUB 140-2, Security Requirements for Cryptographic Modules*, January 4, 2011, National Institute of Standards and Technology, available at http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402DTR.pdf.

5. *Network Security with OpenSSL*, John Viega et. al., 15 June 2002, O'Reilly & Associates

6. *NSA Suite B Cryptography*
http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

7. *The Transitioning of Cryptographic Algorithms and Key Sizes*
http://csrc.nist.gov/groups/ST/key_mgmt/documents/Transitioning_CryptoAlgos_070209.pdf

8. *DRAFT Recommendation for the Transitioning of Cryptographic Algorithms and Key Sizes*
http://csrc.nist.gov/publications/drafts/800-131/draft-sp800-131_spd-june2010.pdf

9. FIPS 186-3, *Digital Signature Standard (DSS)*
http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

10. SP 800-90, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)*,
http://csrc.nist.gov/publications/nistpubs/800-90/SP800-90revised_March2007.pdf

11. SP 800-56A, *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*,
http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf

12. *Suite B Implementer's Guide to NIST SP 800-56A*, http://www.nsa.gov/ia/_files/SuiteB_Implementer_G-113808.pdf

13. SP 800-56B, *Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography*, http://csrc.nist.gov/publications/nistpubs/800-56B/sp800-56B.pdf

14. SP 800-108, Recommendation for Key Derivation Using Pseudorandom Functions, http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf

15. *AES Key Wrap Specification* http://csrc.nist.gov/groups/ST/toolkit/documents/kms/AES_key_wrap.pdf

16. *Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*, http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf

17. May 21, 2009 Army "*Letter to Industry*", https://chess.army.mil/ascp/commerce/scp/downloads/standardspolicy_files/letter_to_industry.pdf

18. *OpenSSL FIPS Object Module User's Guide*, http://openssl.org/docs/fips/UserGuide.pdf

19. The OpenSSL license, http://openssl.org/source/license.html

20. *Alice in Wonderland*, Lewis Carroll, 1865, ISBN 978-0486275437, https://www.gutenberg.org/files/11/11-pdf.pdf

Appendix A OpenSSL Distribution Signing Keys

In order to be considered FIPS 140-2 validated the FIPS Object Module must be derived from an OpenSSL distribution signed by one of these authorized keys, as shown by the value in the **Fingerprint** row. These keys are subject to change and the list at https://openssl.org/about/ will generally be more current.

The procedure for verifying that a source distribution was signed by one of these keys is described in detail in §4.1.2.

Note the fingerprint formats are slightly different for the two different types of keys (RSA and DSA).

OpenSSL Core Team PGP Keys	
Key Id	**Team member**
0E604491	Matt Caswell matt@openssl.org
fingerprint: 8657 ABB2 60F0 56B1 E519 0839 D9C4 D26D 0E60 4491	
49A563D9	Mark J. Cox mark@openssl.org
fingerprint: 7B 79 19 FA 71 6B 87 25 0E 77 21 E5 52 D9 83 BF	
	Viktor Dukhovni viktor@openssl.org
FA40E9E2	Dr. Stephen Henson steve@openssl.org
fingerprint: 6260 5AA4 334A F9F0 DDE5 D349 D357 7507 FA40 E9E2	
41FBF7DD	Tim Hudson tjh@openssl.org
fingerprint: 60A6 0B21 E22D CEDD C50C 0773 06CC 497B 0EEA BFE4	
BDD52F1C	Lutz Jänicke jaenicke@openssl.org
fingerprint: 0A77 335A ADE7 4E6B B36C AD8A DFAB 592A BDD5 2F1C	
	Emilia Käsper emilia@openssl.org C
2118CF83	Ben Laurie ben@openssl.org
fingerprint: 7656 55DE 62E3 96FF 2587 EB6C 4F6D E156 2118 CF83	

6D1892F5	Steve Marquess marquess@openssl.org
fingerprint: FEAB 1FB2 6537 1742 9B0B 894F 4317 11F7 6D18 92F5	

7DF9EE8C	Richard Levitte levitte@openssl.org S
fingerprint: 7953 AC1F BC3D C8B3 B292 393E D5E9 E43F 7DF9 EE8C	

4A397EA2	Bodo Möller bodo@openssl.org
fingerprint: 3FD2 C7DB D3EA 28B7 B0C6 1B5D E9A7 C808 4A39 7EA2	

1FE8E023	Andy Polyakov appro@openssl.org
fingerprint: B652 F27F 2B8D 1B8D A78D 7061 BA6C DA46 1FE8 E023	

41C25E5D	Kurt Roeckx kurt@openssl.org
fingerprint: E5E5 2560 DD91 C556 DDBD A5D0 2064 C536 41C2 5E5D	

5C51B27C	Rich Salz rsalz@openssl.org
fingerprint: D099 684D C7C2 1E02 E14A 8AFE F234 7945 5C51 B27C	

E18C1C32	Geoff Thorpe geoff@openssl.org
fingerprint: 1B3D F808 C221 D2A5 ED74 172F 0833 F510 E18C 1C32	

Appendix B CMVP Test Procedure

Instructions for building OpenSSL and performing the FIPS 140-2 and related algorithm tests on Linux®/Unix® Microsoft Windows® based platforms are given here. These instructions are primarily of interest to the CMVP testing laboratory performing the validation testing, or anyone wishing to verify that the executable library generates generates the same output for the algorithm tests performed by the testing laboratory.

Note there is no requirement for end users or application developers to run these tests; this discussion is included for reference purposes to illustrate the algorithm testing performed by the CMVP test lab. Note this step requires a large directory tree of input test data files produced by the testing lab using a NIST provided tool (CAVS); several sets of input and response values can be found http://openssl.com/testing/validation-2.0/testvectors/. The file

http://openssl.com/testing/validation-2.0/testvectors/tv.tar.gz

contains a complete set of 259 test vector files with correct responses that can be used for a single comprehensive test. Note the number and format of these test vector files changes over time, so this set may not correspond exactly to what the CAVS tool currently produces.

B.1 Building the Software - Linux/Unix

1. Copy the OpenSSL distribution (**`openssl-fips-2.0.tar.gz`**) to a directory on the test system. Approximately 80Mb free space is needed for this file and the resulting work area.

2. Perform the standard build. Use of a **`script`** file or comparable means of capturing the output is highly recommended.

```
gunzip -c openssl-fips-2.0.tar.gz | tar xf -
cd openssl
./config [no-asm]
make
```

   ...where the **`no-asm`** option may or not be present depending on the platform.

3. Run

```
make build_tests
```

to generate the standalone additional programs to support the testing process. To generate a single program that contains the functionality of fips_test_suite and the individual standalone algorithm test programs, run

```
make build_algvs
```

to build the **fips_algvs** program. This program is necessary for some platforms that do not provide a suitable command shell and for which the execution of many separate programs is awkward or difficult, and may be convenient in other circumstances.

The **fips_algvs** program can be used to execute specific tests, for instance

```
fips_algv fips_test_suite post
fips_algv fips_dssvs pqg "tv/req/PQGGen.req"
      "tv/resp/PQGGen.rsp"
```

or if given no command line options it will process the subcommands in a minimal shell script as generated by

```
perl fipsalgtest.pl --dir=<testvectors> --minimal-script
     --generate-script=fipstests.sh perl --tprefix=
```

which will produce a file **fipstests.sh** with the subcommands corresponding to each request file, e.g.:

```
fips_dssvs pqg "tv/req/PQGGen.req" "tv/resp/PQGGen.rsp"
```

The **fips_algvs** program supports the following command line options:

-quiet	suppress any progress output.
-verbose	echo full command lines of executed commands (default is to omit filenames)
-script *<filename>*	script to use, default is fipstests.sh

In absence of any options it assumes a script file **fipstests.sh** should be read from the current directory. If the first argument doesn't begin with a '-' it is taken as the name of a sub program to run:

```
fips_aesavs
fips_algvs
fips_cmactest
fips_desmovs
fips_dhvs
```

```
fips_drbgvs
fips_dsatest
fips_dssvs
fips_ecdhvs
fips_ecdsavs
fips_gcmtest
fips_hmactest
fips_randtest
fips_rngvs
fips_rsagtest
fips_rsastest
fips_rsavtest
fips_shatest
fips_test_suite
```

Note that for future validations the **fips_algvs** program will probably entirely replace the separate **fips_test_suite** and algorithm test driver programs.

B.2 Algorithm Tests - Linux/Unix

4. Add the subtree of test data to the distribution work area:

```
cd fips
unzip <zipfile of test vectors>.zip –d testvectors
```

5. Run the FIPS 140-2 algorithm tests:

```
perl fipsalgtest.pl --dir=testvectors
```

This step runs the algorithm tests specific to the FIPS mode. Again a large amount of output will be generated. If an error occurs processing will be aborted. The output from the cryptographic tests will be compared against the response files already present in the test data and not permanently stored. This comparison automatically suppresses the whitespace and comment line differences and ignores the seven test vector files that are always different[58].

[58]Due to the nature of the cryptographic operations involved the following responses files will always be different:

KeyPair.rsp	DSA
PQGGen.rsp	DSA
SigGen.rsp	DSA
SigGen15.rsp	RSA
SigGenPSS.rsp	RSA
SigGenRSA.rsp	RSA
SigGenPSS.rsp	RSA

The special case cryptographic operations are listed in the associative array **%verify_specials** tin the *fipsalgvs.pl* perl script.

6. To generate and preserve new response files use the **`--generate`** option:

   ```
   perl fipsalgtest.pl --dir=testvectors --generate
   ```

   Many (approximately 259) generated **`*.rsp`** files will be found in the **`./testvectors/`** directory tree under **`./fips`**/:

   ```
   find testvectors/ -name '*.rsp'
   ```

7. The tree of **`*.rsp`** files can also be extracted for comparison with another tree:

   ```
   find testvectors -name '*.rsp' | cpio -oc > rsp1.cpio
        .
        .
        .
   cd /tmp
   mkdir rsp1 rsp2
   cd rsp1; cpio -ic < rsp1.cpio
   cd ../rsp2; cpio -ic < rsp2.cpio
   diff -r . ../rsp1
   ```

If the only other differences are the commented date-time labels then the trees match:

```
diff -r ./testvectors/aes/resp/CBCGFSbox128.rsp \
     ../rsp1/testvectors/aes/resp/CBCGFSbox128.rsp
6c6
< # Thu Mar  4 11:05:36 2004
---
> # Fri Feb 20 12:21:24 2004
diff -r ./testvectors/aes/resp/CBCGFSbox192.rsp \
     ../rsp1/testvectors/aes/resp/CBCGFSbox192.rsp
6c6
< # Thu Mar  4 11:05:36 2004
---
> # Fri Feb 20 12:21:24 2004
     .
     .
     .
```

B.3 Building the Software - Windows

1. Copy the OpenSSL distribution (**openssl-fips-2.0.tar.gz**) to a directory on the test system. Approximately 80Mb free space is needed.

2. Perform the standard build.

```
cd openssl
ms\do_fips [no-asm]
out32dll\fips_test_suite
```

...where the **no-asm** option may or not be present depending on the platform.

B.4 Algorithm Tests - Windows

3. This procedure is similar to that for Linux/Unix:

```
cd fips
unzip <zipfile of test vectors>.zip -d testvectors
perl fipsalgtest.pl --win32 --dir=testvectors
.\fipstests.bat
```

There is no bundled zip/unzip command for most versions of Microsoft Windows, but many third party implementations are available, such as *http://gnuwin32.sourceforge.net/packages/unzip.htm*.

B.5 FIPS 140-2 Test - All Platforms

A test driver program has been provided to demonstrate both successful and failed power-up self-tests and the invocation of some basic cryptographic operations. This program was developed during the course of the FIPS 140-2 validation as a aid to the test lab evaluators. This test program, **fips_test_suite**, can be found in the **./test/** subdirectory. This program behaves the same for Linux/Unix and Windows; for Windows invoke as **.\fips_test_suite** instead of **./fips_test_suite** as shown in this example.

1. When executed with no argument output similar to the full suite of algorithm tests is performed, producing the following output:

```
$ FIPS-mode test application
      FIPS 2.0-dev unvalidated test module xx XXX xxxx

            DRBG AES-256-CTR DF test started
            DRBG AES-256-CTR DF test OK
1. Non-Approved cryptographic operation test...
      a. Included algorithm (D-H)......successful
      POST started
            Integrity  test started
```

```
Integrity  test OK
DRBG AES-256-CTR DF test started
DRBG AES-256-CTR DF test OK
DRBG AES-256-CTR test started
DRBG AES-256-CTR test OK
DRBG SHA256 test started
DRBG SHA256 test OK
DRBG HMAC-SHA256 test started
DRBG HMAC-SHA256 test OK
DRBG P-256 SHA256 test started
DRBG P-256 SHA256 test OK
X9.31 PRNG keylen=16 test started
X9.31 PRNG keylen=16 test OK
X9.31 PRNG keylen=24 test started
X9.31 PRNG keylen=24 test OK
X9.31 PRNG keylen=32 test started
X9.31 PRNG keylen=32 test OK
Digest SHA1 test started
Digest SHA1 test OK
Digest SHA1 test started
Digest SHA1 test OK
Digest SHA1 test started
Digest SHA1 test OK
HMAC SHA1 test started
HMAC SHA1 test OK
HMAC SHA224 test started
HMAC SHA224 test OK
HMAC SHA256 test started
HMAC SHA256 test OK
HMAC SHA384 test started
HMAC SHA384 test OK
HMAC SHA512 test started
HMAC SHA512 test OK
CMAC AES-128-CBC test started
CMAC AES-128-CBC test OK
CMAC AES-192-CBC test started
CMAC AES-192-CBC test OK
CMAC AES-256-CBC test started
CMAC AES-256-CBC test OK
CMAC DES-EDE3-CBC test started
CMAC DES-EDE3-CBC test OK
Cipher AES-128-ECB test started
Cipher AES-128-ECB test OK
CCM  test started
CCM  test OK
GCM  test started
GCM  test OK
XTS AES-128-XTS test started
XTS AES-128-XTS test OK
XTS AES-256-XTS test started
```

```
            XTS AES-256-XTS test OK
            Cipher DES-EDE3-ECB test started
            Cipher DES-EDE3-ECB test OK
            Cipher DES-EDE3-ECB test started
            Cipher DES-EDE3-ECB test OK
            Signature RSA test started
            Signature RSA test OK
            Signature ECDSA P-224 test started
            Signature ECDSA P-224 test OK
            Signature ECDSA K-233 test started
            Signature ECDSA K-233 test OK
            Signature DSA test started
            Signature DSA test OK
            ECDH P-224 test started
            ECDH P-224 test OK
      POST Success
2. Automatic power-up self test...successful
3a. AES encryption/decryption...successful
3b. AES-GCM encryption/decryption...successful
            Pairwise Consistency RSA test started
            Pairwise Consistency RSA test OK
            Pairwise Consistency RSA test started
            Pairwise Consistency RSA test OK
            Pairwise Consistency RSA test started
            Pairwise Consistency RSA test OK
4. RSA key generation and encryption/decryption...successful
5. DES-ECB encryption/decryption...successful
            Pairwise Consistency DSA test started
            Pairwise Consistency DSA test OK
6. DSA key generation and signature validation...successful
7a. SHA-1 hash...successful
7b. SHA-256 hash...successful
7c. SHA-512 hash...successful
7d. HMAC-SHA-1 hash...successful
7e. HMAC-SHA-224 hash...successful
7f. HMAC-SHA-256 hash...successful
7g. HMAC-SHA-384 hash...successful
7h. HMAC-SHA-512 hash...successful
8a. CMAC-AES-128 hash...successful
8b. CMAC-AES-192 hash...successful
8c. CMAC-AES-256 hash...successful
8e. CMAC-TDEA-3 hash...successful
9. Non-Approved cryptographic operation test...
      a. Included algorithm (D-H)...successful as expected
            Pairwise Consistency RSA test started
            Pairwise Consistency RSA test OK
            Pairwise Consistency RSA test started
            Pairwise Consistency RSA test OK
            Pairwise Consistency RSA test started
            Pairwise Consistency RSA test OK
```

```
 Generated 128 byte RSA private key
      BN key before overwriting:
400e460169e1e37d8f415fe50c40fab493185c17e99b76e123bc0f3d7d0c8b1f42881ff7396b
3ee388c3b973cece2d7d231109a7202016daf1e26caca9e704b9bffd9bd6151d61ab3050a82e
78510abf2e450a6c57e9fb7db8a837f81fc93db0c6c95d090ac6752b8ac4ee51623ffcbd270b
0ed281ebbe2e6a3a9d0a4012a991  BN key after overwriting:
668d6314da4f25ca496a6f98e2f6986437be60f2d34880e8d08060263dd10a3bde7345ef99ed
00e2edeedf43a1bda7053c58b6474051bbaf9c9e5bf70a488a7b94d88c67fc9e16fc9e4bb231
8836dc47282c8e41d3c35bc400949cd2d2b5e0ee0bd84ce8dffdb02dfc6c9528d0be43b0d95f
ce6e979c561070e6da5a05b9e53e  char buffer key before overwriting:
            4850f0a33aedd3af6e477f8302b10968
      char buffer key after overwriting:
            788fadb58c8163405e883a63550fd732
10. Zero-ization...
      successful as expected
11. Complete DRBG health check...
            DRBG AES-128-CTR DF test started
            DRBG AES-128-CTR DF test OK
            DRBG AES-192-CTR DF test started
            DRBG AES-192-CTR DF test OK
                  .
                  .
                  .
            (very long list of DRBG tests)
                  .
                  .
                  .
            DRBG P-521 SHA384 test started
            DRBG P-521 SHA384 test OK
            DRBG P-521 SHA512 test started
            DRBG P-521 SHA512 test OK
      successful as expected
12. DRBG generation check...
            DRBG SHA1 test started
            DRBG SHA1 test OK
            DRBG SHA1 test started
            DRBG SHA1 test OK
                  .
                  .
                  .
            (very long list of DRBG tests)
                  .
                  .
            DRBG P-521 SHA512 test OK
            DRBG P-521 SHA512 test started
            DRBG P-521 SHA512 test OK
      successful as expected

All tests completed with 0 errors
```

The **nodh** option skips the glacial and largely pointless DH test.

The **nodrbg** option skips the slow full DRBG test

The **fullpost** option gives a complete POST listing instead of induced failure and unexpected errors. The output is then much more verbose as it contains every successful test too.

The **fullerr** option is useful for code tracing. Normally during the induced failure test library errors are not printed out. With this option the error codes corresponding to each operation are displayed showing the exact line and error code output.

2. When executed with the **post** command line option only module initialization will be performed:

```
$ test/fips_test_suite post
      FIPS-mode test application
      FIPS 2.0-dev unvalidated test module xx XXX xxxx

            DRBG AES-256-CTR DF test started
            DRBG AES-256-CTR DF test OK
      POST started
            Integrity  test started
            Integrity  test OK
            DRBG AES-256-CTR DF test started
            DRBG AES-256-CTR DF test OK
            DRBG AES-256-CTR test started
            DRBG AES-256-CTR test OK
            DRBG SHA256 test started
            DRBG SHA256 test OK
            DRBG HMAC-SHA256 test started
            DRBG HMAC-SHA256 test OK
            DRBG P-256 SHA256 test started
            DRBG P-256 SHA256 test OK
            X9.31 PRNG keylen=16 test started
            X9.31 PRNG keylen=16 test OK
            X9.31 PRNG keylen=24 test started
            X9.31 PRNG keylen=24 test OK
            X9.31 PRNG keylen=32 test started
            X9.31 PRNG keylen=32 test OK
            Digest SHA1 test started
            Digest SHA1 test OK
            Digest SHA1 test started
            Digest SHA1 test OK
            Digest SHA1 test started
            Digest SHA1 test OK
            HMAC SHA1 test started
            HMAC SHA1 test OK
```

```
            HMAC SHA224 test started
            HMAC SHA224 test OK
            HMAC SHA256 test started
            HMAC SHA256 test OK
            HMAC SHA384 test started
            HMAC SHA384 test OK
            HMAC SHA512 test started
            HMAC SHA512 test OK
            CMAC AES-128-CBC test started
            CMAC AES-128-CBC test OK
            CMAC AES-192-CBC test started
            CMAC AES-192-CBC test OK
            CMAC AES-256-CBC test started
            CMAC AES-256-CBC test OK
            CMAC DES-EDE3-CBC test started
            CMAC DES-EDE3-CBC test OK
            Cipher AES-128-ECB test started
            Cipher AES-128-ECB test OK
            CCM  test started
            CCM  test OK
            GCM  test started
            GCM  test OK
            XTS AES-128-XTS test started
            XTS AES-128-XTS test OK
            XTS AES-256-XTS test started
            XTS AES-256-XTS test OK
            Cipher DES-EDE3-ECB test started
            Cipher DES-EDE3-ECB test OK
            Cipher DES-EDE3-ECB test started
            Cipher DES-EDE3-ECB test OK
            Signature RSA test started
            Signature RSA test OK
            Signature ECDSA P-224 test started
            Signature ECDSA P-224 test OK
            Signature ECDSA K-233 test started
            Signature ECDSA K-233 test OK
            Signature DSA test started
            Signature DSA test OK
            ECDH P-224 test started
            ECDH P-224 test OK
      POST Success
Power-up self test successful
$
```

Note this invocation is useful for a quick estimation of the performance impact of module initialization.

3. To demonstrate the correct functioning of the integrity and KAT test failures a set of corruption tests are run automatically when the unqualified **`fips_test_suite`** option is specified. In

the implementation of the **fips_algvs** utility these tests are specified in the **fail_list_flist** structure and a series of in-line tests which are traversed by the static function **do_fail_all()** at the point where the line

```
13. Induced test failure check...
```

is printed. Each specific test is preceded by one of the lines

```
Testing induced failure of XXXX
Testing operation failure with XXXX
```

and the conclusion of all the corruption tests should end with the lines

```
Induced failure test completed with 0 errors
      successful as expected
```

Note the use of three static variables by the function **do_fail_all()** to specify the specific corruption tests to be performed.

The individual tests in the order performed are:

```
Integrity
AES
DES3
AES-GCM
AES-CCM
AES-XTS
Digest
HMAC
CMAC
DRBG
X9.31 PRNG
RSA
DSA
ECDSA
ECDH
RSA keygen
DSA keygen
ECDSA keygen
DRBG CPRNG
DRBG entropy CPRNG
X9.31 CPRNG
```

DRBG entropy failure

This full set of corruption tests should appear as follows:

```
13. Induced test failure check...
    Testing induced failure of Integrity test
    POST started
         Integrity  test failure induced
         Integrity  test failed as expected
    POST Failed
    Testing induced failure of AES test
    POST started
         Cipher AES-128-ECB test failure induced
         Cipher AES-128-ECB test failed as expected
    POST Failed
    Testing induced failure of DES3 test
    POST started
         Cipher DES-EDE3-ECB test failure induced
         Cipher DES-EDE3-ECB test failed as expected
    POST Failed
    Testing induced failure of AES-GCM test
    POST started
         GCM  test failure induced
         GCM  test failed as expected
    POST Failed
    Testing induced failure of AES-CCM test
    POST started
         CCM  test failure induced
         CCM  test failed as expected
    POST Failed
    Testing induced failure of AES-XTS test
    POST started
         XTS AES-128-XTS test failure induced
         XTS AES-128-XTS test failed as expected
         XTS AES-256-XTS test failure induced
         XTS AES-256-XTS test failed as expected
    POST Failed
    Testing induced failure of Digest test
    POST started
         Digest SHA1 test failure induced
         Digest SHA1 test failed as expected
         Digest SHA1 test failure induced
```

```
     Digest SHA1 test failed as expected
     Digest SHA1 test failure induced
     Digest SHA1 test failed as expected
POST Failed
Testing induced failure of HMAC test
POST started
     HMAC SHA1 test failure induced
     HMAC SHA1 test failed as expected
     HMAC SHA224 test failure induced
     HMAC SHA224 test failed as expected
     HMAC SHA256 test failure induced
     HMAC SHA256 test failed as expected
     HMAC SHA384 test failure induced
     HMAC SHA384 test failed as expected
     HMAC SHA512 test failure induced
     HMAC SHA512 test failed as expected
POST Failed
Testing induced failure of CMAC test
POST started
     CMAC AES-128-CBC test failure induced
     CMAC AES-128-CBC test failed as expected
     CMAC AES-192-CBC test failure induced
     CMAC AES-192-CBC test failed as expected
     CMAC AES-256-CBC test failure induced
     CMAC AES-256-CBC test failed as expected
     CMAC DES-EDE3-CBC test failure induced
     CMAC DES-EDE3-CBC test failed as expected
POST Failed
Testing induced failure of DRBG test
POST started
     DRBG AES-256-CTR test failure induced
     DRBG AES-256-CTR DF test failed as expected
     DRBG AES-256-CTR test failure induced
     DRBG AES-256-CTR test failed as expected
     DRBG SHA256 test failure induced
     DRBG SHA256 test failed as expected
     DRBG HMAC-SHA256 test failure induced
     DRBG HMAC-SHA256 test failed as expected
     DRBG P-256 SHA256 test failure induced
     DRBG P-256 SHA256 test failed as expected
POST Failed
Testing induced failure of X9.31 PRNG test
```

```
POST started
     X9.31 PRNG keylen=16 test failure induced
     X9.31 PRNG keylen=16 test failed as expected
     X9.31 PRNG keylen=24 test failure induced
     X9.31 PRNG keylen=24 test failed as expected
     X9.31 PRNG keylen=32 test failure induced
     X9.31 PRNG keylen=32 test failed as expected
POST Failed
Testing induced failure of RSA test
POST started
     Signature RSA test failure induced
     Signature RSA test failed as expected
POST Failed
Testing induced failure of DSA test
POST started
     Signature DSA test failure induced
     Signature DSA test failed as expected
POST Failed
Testing induced failure of ECDSA test
POST started
     Signature ECDSA P-224 test failure induced
     Signature ECDSA P-224 test failed as expected
POST Failed
Testing induced failure of ECDH test
POST started
     ECDH P-224 test failure induced
     ECDH P-224 test failed as expected
POST Failed
Testing induced failure of RSA keygen test
POST started
POST Success
     Pairwise Consistency RSA test failure induced
     Pairwise Consistency RSA test failed as expected
RSA key generation failed as expected.
Testing induced failure of DSA keygen test
POST started
POST Success
     Pairwise Consistency DSA test failure induced
     Pairwise Consistency DSA test failed as expected
DSA key generation failed as expected.
POST started
POST Success
```

```
Testing induced failure of ECDSA keygen test
     Pairwise Consistency ECDSA test failure induced
     Pairwise Consistency ECDSA test failed as expected
ECDSA key generation failed as expected.
POST started
POST Success
Testing induced failure of DRBG CPRNG test
DRBG continuous PRNG failed as expected
POST started
POST Success
Testing induced failure of DRBG entropy CPRNG test
DRBG continuous PRNG entropy failed as expected
POST started
POST Success
POST started
POST Success
Testing induced failure of X9.31 CPRNG test
X9.31 continuous PRNG failed as expected
POST started
POST Success
Testing operation failure with DRBG entropy failure
DSA key generated OK as expected.
DRBG entropy fail failed as expected
DSA signing failed as expected
ECDSA key generation failed as expected.
Induced failure test completed with 0 errors
successful as expected
```

So, the presence of the line

```
Induced failure test completed with 0 errors
```

for the block of tests beginning with the line

```
13. Induced test failure check...
```

is a readily observed indication that all corruption tests performed as expected.

4. To demonstrate the module authentication one of four command line options may be given to specify the password value to be passed to **`FIPS_module_mode_set()`**:

   `nopass` Null password

badpass	Invalid password of sufficient length
user	The **FIPS_AUTH_CRYPTO_USER** password
officer	The **FIPS_AUTH_CRYPTO_OFFICER** password

If none of those command line options are given the **FIPS_AUTH_CRYPTO_USER** password is used. Invocation with **none** or **badpass** will fail:

```
$ test/fips_test_suite badpass
  FIPS-mode test application
  FIPS 2.0-dev unvalidated test module xx XXX xxxx

       DRBG AES-256-CTR DF test started
       DRBG AES-256-CTR DF test OK
ERROR:2D078097:lib=45,func=120,reason=151:file=fips.c:line=
300
Power-up self test failed
$
```

and invocation with **user** or **officer** will successfully perform the POST test.

B.6 Testvector Data Files and the fipsalgtest.pl Utility

The FIPS 140-2 test labs use CAVP provided Windows based software known as the “CAVS tool” to generate the test vector data files used for the algorithm tests. The algorithms desired are typically specified using forms proprietary to the specific test lab performing the testing. Non-proprietary facsimiles of those forms specifying the algorithms tests foe the 2.0 module validation can be found at http://openssl.com/testing/validation-2.0/forms/.

The test lab uses the CAVS tools to generate a set of "request" files for which corresponding "response" files must be generated by the module (the IUT or Implementation Under test). The set of request files is typically delivered in a single zip or tar file containing a directory tree with arbitrary pathnames. The only constant is the names of the actual ***.rsp** response files of input data. Since matching filenames up by hand quickly becomes tedious we have developed a utility, **fipsalgtest.pl**, that will search through a directory hierarchy and identify the relevant test vector files.

For the initial validation there were 257 unique file names with 2 duplicate names, for a total of 259 files:

Algorithm	Number of *.req files
AES	108

Algorithm	Number of *.req files
AES_GCM	6
CCM	15
CMAC	8
DES	0
DRBG	4
DSA	5
DSA2	5
ECDSA	4
ECDSA2	4
HMAC	1
KAS	1
RNG	6
RSA	9
SHA	15
TDES	66
XTS	2
Total	259

In order to facilitate the processing of test vector data a series of utilities were developed, culminating in the **`fipsalgtest.pl`** program. This program searches a target directory for the known **`*.rsp`** files and generates a script referencing the actual pathnames for those files. That script can then be executed to perform the algorithm tests that generate the **`*.rsp`** result files. The **`fipsalgtest.pl`** program reports unrecognized duplicate **`*.rsp`** files and any files that were expected but not found.

Testvector data sets are generally received as **`*.zip`** files, more rarely as **`*.tgz`**. A typical pathname structure (for this validation) is as follows:

```
./OSF_2464_Template
./OSF_2464_Template/AES
./OSF_2464_Template/AES/resp
./OSF_2464_Template/AES/req
./OSF_2464_Template/AES/req/CBCGFSbox128.req
./OSF_2464_Template/AES/req/CFB128MMT192.req
./OSF_2464_Template/AES/req/CBCVarKey192.req
./OSF_2464_Template/AES/req/CFB1VarTxt256.req
./OSF_2464_Template/AES/req/CBCMMT128.req
./OSF_2464_Template/AES/req/CBCKeySbox256.req
./OSF_2464_Template/AES/req/ECBVarTxt192.req
./OSF_2464_Template/AES/req/CFB128VarKey256.req
./OSF_2464_Template/AES/req/OFBVarTxt128.req
./OSF_2464_Template/AES/req/CFB1MCT192.req
```

```
./OSF_2464_Template/AES/req/CBCVarKey128.req
./OSF_2464_Template/AES/req/CFB8VarTxt128.req
./OSF_2464_Template/AES/req/ECBMMT128.req
./OSF_2464_Template/AES/req/CBCGFSbox192.req
./OSF_2464_Template/AES/req/CFB128MCT192.req
./OSF_2464_Template/AES/req/OFBMCT128.req
./OSF_2464_Template/AES/req/CFB1GFSbox256.req
       .
       .
       .
```

Note directory names may contain embedded spaces. The data files will generally (though not necessarily) be carriage return-line feed delimited.

If multiple platforms are involved in a validation the test vector files for several platforms may be interspersed in the same directory tree. We have also received test vector files for a single platform in multiple different **`*.zip`** files, so the **`fipsalgtest.pl`** program must be able to filter the relevant **`*.rsp`** files out of multiple subdirectories.

The following **`fipsalgtest.pl`** options can be used to accommodate various representations of test vector files:

```
fipsalgtest.pl: generate run CAVP algorithm tests
       --debug                         Enable debug output
       --dir=<dirname>                 Optional root for *.req file search
       --filter=<regexp>               Regex for input files of interest
       --onedir <dirname>              Assume all components in current directory
       --rspdir=<dirname>              Name of subdirectories containing *.rsp
                                          files, default "resp"
       --tprefix=<prefix>              Pathname prefix for directory containing test
                                          programs
       --ignore-bogus                  Ignore duplicate or bogus files
       --ignore-missing                Ignore missing test files
       --quiet                         Shhh....
       --quiet-bogus                   Skip unrecognized file warnings
       --quiet-missing                 Skip missing request file warnings
       --generate                      Generate algorithm test output
       --generate-script=<filename>    Generate script to call algorithm programs
       --minimal-script                Simplest possible output for
                                          --generate-script
       --win32                         Win32 environment
       --compare-all                   Verify unconditionally for all tests
       --list-tests                    Show individual tests
       --mkdir=<cmd>                   Specify "mkdir" command
       --notest                        Exit before running tests
       --rm=<cmd>                      Specify "rm" command
       --script-tprefix                Pathname prefix for --generate-script output
       --enable-<alg>                  Enable algorithm set <alg>.
       --disable-<alg>                 Disable algorithm set <alg>.
       Where <alg> can be one of:
              aes-ccm                (disabled by default)
              rand-aes               (enabled by default)
              ecdsa                  (disabled by default)
```

```
hmac                (enabled by default)
dh                  (disabled by default)
aes-cfb1            (disabled by default)
ecdh                (disabled by default)
des3-cfb1           (disabled by default)
drbg                (disabled by default)
des3                (enabled by default)
dsa                 (enabled by default)
dsa-pqgver          (disabled by default)
rsa-pss0            (disabled by default)
sha                 (enabled by default)
aes                 (enabled by default)
dsa2                (disabled by default)
aes-gcm             (disabled by default)
rsa-pss62           (enabled by default)
cmac                (disabled by default)
aes-xts             (disabled by default)
rsa                 (enabled by default)
v2                  (enabled by default)
rand-des2           (disabled by default)
```

Simply run

perl fipsalgtest.pl --dir=*testvectors* --generate

to generate the ***.rsp** files for submission to the test lab.

Subsequently running **fipsalgtest.pl** without the --**generate** option will compare the generated output with the previously existing ***.rsp** files, and thus provides a comprehensive (though unofficial) check of the algorithm tests.

Individual algorithm tests can be selectively specified with options of the form --**enable-*xxx*** or **--disable-*xxx*** where ***xxx*** is one of the **<alg>** algorithm specifications

The **--ignore-bogus** and **--ignore-missing** options suppress the error exit if the target test vector directory contains more or fewer ***.rsp** files than expected (a not uncommon occurrence in validation testing.

For target platforms that do not support a perl interpreter, but which do provide a basic command line shell, a simple shell script can be generated, for instance:

```
perl ./fips/fipsalgtest.pl --generate-script=fipstest.sh --tprefix=./test/
```

will create a file fipstest.sh script file that successively invokes each of the algorithm test driver programs with the appropriate input and output file names:

```
#!/bin/sh
```

```
# Test vector run script
# Auto generated by fipsalgtest.pl script
# Do not edit

echo Running Algorithm Tests

RM="rm -rf";
MKDIR="mkdir";
TPREFIX=./test/

echo "Running DSA tests"
$RM "./testvectors/tv/OSF_2464_Template/DSA/resp"
$MKDIR "./testvectors/tv/OSF_2464_Template/DSA/resp"

echo "    running PQGGen test"
${TPREFIX}fips_dssvs pqg
"./testvectors/tv/OSF_2464_Template/DSA/req/PQGGen.req"
"./testvectors/tv/OSF_2464_Template/DSA/resp/PQGGen.rsp"
echo "    running KeyPair test"
${TPREFIX}fips_dssvs keypair
"./testvectors/tv/OSF_2464_Template/DSA/req/KeyPair.req"
"./testvectors/tv/OSF_2464_Template/DSA/resp/KeyPair.rsp"
echo "    running SigGen test"
${TPREFIX}fips_dssvs siggen
"./testvectors/tv/OSF_2464_Template/DSA/req/SigGen.req"
"./testvectors/tv/OSF_2464_Template/DSA/resp/SigGen.rsp"
echo "    running SigVer test"
${TPREFIX}fips_dssvs sigver
"./testvectors/tv/OSF_2464_Template/DSA/req/SigVer.req"
"./testvectors/tv/OSF_2464_Template/DSA/resp/SigVer.rsp"
echo "    running PQGVer test"
${TPREFIX}fips_dssvs pqgver
"./testvectors/tv/OSF_2464_Template/DSA/req/PQGVer.req"
"./testvectors/tv/OSF_2464_Template/DSA/resp/PQGVer.rsp"
      .
      .
      .
```

For very simple shells the **--minimal-script** option will omit use of the **rm** and **mkdir** commands to manage the output directories, in which case the empty **req** subdirectories will need to be created beforehand.

To process only a subset of the test vectors file, use the **--filter=*XXX*** option to recognize only certain pathnames and the **--disable-all --enable-*xxx*** options to enable processing of only the algorithm(s) in that selected set for files. For instance:

```
perl ./fips/fipsalgtest.pl --generate-script=fipstestsha.sh --tprefix=./test/
      --disable-all --enable-sha --dir=testvectors --filter=SHA
```

B.6 Documentation

This section discussed the major components of the documentation set for a FIPS 140-2 validation.

Finite State Model

FIPS 140-2 validation requires a Finite State Module (FSM), something that doesn't make much sense for a general purpose cryptographic library. This cosmetic requirement is satisfied by an arbitrary generic diagram and possibly an associated listing or spreadsheet of the states and transitions. Each test lab will typically have a generic template or sample that can be used. The FSM used for this validation can be found in the two files:

http://openssl.com/testing/validation-2.0/docs/FSM.pdf
http://openssl.com/testing/validation-2.0/docs/FSM_main.pdf

The FSM does not contain any information of actual technical value.

Vendor Evidence Document

The test lab must answer the assertions in the Derived Test Requirements (DTR) document (Reference 4). Some labs chose to do so by directly listing all of the assertions with corresponding responses in the order those assertions appear in the DTR. Others respond to the assertions in analysis document structured along more functional lines with many of the redundant an overlapping assertions grouped together with a consolidated response. As with the formal test report (see following section) the test lab will typically want to claim this document as proprietary. The relevant content of the analysis document for this validation has been extracted as Appendix E.

Formal Test Report

The test lab submits a formal test report document to the CMVP. Test labs are uniformly adverse to releasing this document but can usually be persuaded to do so under a non-disclosure agreement (such release should be negotiated *prior* to executing a contract). OSF has seen some test reports but cannot publish them due to the non-disclosure restrictions. Note that those test reports would be of limited value as different test labs can take significantly different approaches to presenting the same module to the CMVP. FIPS 140-2 validation is a highly subjective process and each test lab, and even different reviewers at the CMVP, have distinctive styles. Mixing components from multiple submissions, even of exactly the same software, would result in significant discrepancies and conflicts.

Appendix C Example OpenSSL Based Application

This example shows a simple application using OpenSSL cryptography which will qualify as FIPS 140-2 validated when built and installed in accordance with the procedures in §5. In this application all cryptography is provided through the FIPS Object Module and the FIPS mode initialization is performed via the **`FIPS_mode_set()`** call. The command generates a HMAC-SHA-1 digest of an input stream or a file, using the same arbitrary key as the OpenSSL FIPS Module file integrity check:

```
$ ./fips_hmac -v fips_hmac.c
FIPS mode enabled
8f2c8e4f60607613471c11287423f8429b068eb2
$
$ ./hmac < hmac.c
8f2c8e4f60607613471c11287423f8429b068eb2
$
```

Note this sample command is functionally equivalent to:

```
env OPENSSL_FIPS=1 openssl -hmac etaonrishdlcupfm hmac.c
```

or

```
openssl dgst -fips-fingerprint filename.tar.gz
```

for an **`openssl`** command built from a FIPS capable OpenSSL distribution. The **`OPENSSL_FIPS=1`** environment variable enables FIPS mode for a **`openssl`** command generated from a FIPS capable OpenSSL distribution.

C.1 Native Compilation of Statically Linked Program

Makefile

```
CC = gcc
OPENSSLDIR = /usr/local/ssl
LIBCRYPTO = $(OPENSSLDIR)/lib/libcrypto.a
INCLUDES = -I$(OPENSSLDIR)/include
CMD     = fips_hmac
OBJS = $(CMD).o

$(CMD): $(OBJS)
    FIPSLD_CC=$(CC) $(OPENSSLDIR)/bin/fipsld -o $(CMD) $(OBJS) \
      $(LIBCRYPTO)

$(OBJS): $(CMD).c
    $(CC) -c $(CMD).c $(INCLUDES)
```

```
clean:
      rm $(OBJS)
```

Note the line

```
    $(OPENSSLDIR)/fips/fipsld -o $(CMD) $(OBJS) ...
```

uses the **fipsld** command from the distribution source tree to perform the function of verifying the **fipscanister.o** digest and generating the new embedded digest in the application executable object.

Source File

```
/*
   Sample application using FIPS mode OpenSSL.

   This application will qualify as FIPS 140-2 validated when built,
   installed, and utilized as described in the "OpenSSL FIPS 140-2
   Security Policy" manual.

   This command calculates a HMAC-SHA-1 digest of a file or input data
   stream using the same arbitrary hard-coded key as the FIPS 140-2
   source file build-time integrity checks and runtime executable
   file integrity check.
*/

#include <stdio.h>
#include <string.h>
#include <openssl/hmac.h>

static char label[] = "@(#)FIPS approved SHA1 HMAC";

static void dofile(FILE *fp)
    {
    HMAC_CTX ctx;
    unsigned char hmac_value[EVP_MAX_MD_SIZE];
    int hmac_len, i;
    char key[] = "etaonrishdlcupfm";
    char buf[256];

    /* Initialise context */
    HMAC_CTX_init(&ctx);
    /* Set digest type and key in context */
    HMAC_Init_ex(&ctx, key, strlen(key), EVP_sha1(), NULL);
    /* Process input stream */
    while(i = fread(buf,sizeof(char),sizeof(buf),fp)) {
        if(!HMAC_Update(&ctx, buf, i)) exit(3);
    }
```

```
    /* Generate digest */
    if(!HMAC_Final(&ctx, hmac_value, &hmac_len)) exit(4);
    HMAC_CTX_cleanup(&ctx);

    /* Display digest in hex */
    for(i = 0; i < hmac_len; i++) printf("%02x", hmac_value[i]);
        printf("\n");
    return;
}

main(int argc, char *argv[])
    {
    char *opt = NULL;
    int verbose = 0;
    int fipsmode = 1;
    FILE *fp = stdin;
    int i;

    /* Process command line arguments */
    i = 0;
    while(++i < argc) {
        opt = argv[i];
        if (!strcmp(opt,"-v")) verbose = 1;
        else if (!strcmp(opt,"-c")) fipsmode = 0;
        else if ('-' == opt[0]) {
            printf("Usage: %s <filename>\n", argv[0]);
            puts("Options:");
            puts("\t-c\tUse non-FIPS mode");
            puts("\t-v\tVerbose output");
            exit(1);
        }
        else break;
    }

    /* Enter FIPS mode by default */
    if (fipsmode) {
        if(FIPS_mode_set(1)) {
            verbose && fputs("FIPS mode enabled\n",stderr);
        }
        else {
            ERR_load_crypto_strings();
            ERR_print_errors_fp(stderr);
            exit(1);
        }
    }

    if (i >= argc) {
        dofile(fp);
    }
    else {
```

```
        while(i < argc) {
            opt = argv[i];
            if ((fp = fopen(opt,"rb")) == NULL) {
                fprintf(stderr,"Unable to open file \"%s\"\n", opt);
                exit(1);
            }
        dofile(fp);
        fclose(fp);
        i++;
        }
    }

    exit(0);
}
```

C.2 Cross-compilation of "FIPS capable" Shared OpenSSL Libraries

Here is an example of building and executing the same example program on an Android 4.0 device using a shared **libcrypto** library. The NDK and SDK are from the files

```
android-sdk_r18-linux.tgz
android-ndk-r7c-linux-x86.zip
```

downloaded from http://developer.android.com/sdk/index.html.

```
# Establish the cross-compilation environment
export ANDROID_NDK=$PWD/android-ndk-r7c
export FIPS_SIG=$PWD/openssl-fips-2.0/util/incore
PATH=$ANDROID_NDK/toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-
x86/bin:$PATH
export PATH
export MACHINE=armv7l
export RELEASE=2.6.39
export SYSTEM=android
export ARCH=arm
export CROSS_COMPILE="arm-linux-androideabi-"
export ANDROID_DEV="$ANDROID_NDK/platforms/android-14/arch-arm/usr"
export HOSTCC=gcc

# Build the FIPS module
gunzip -c openssl-fips-2.0.tar.gz | tar xf -
cd openssl-fips-2.0/
./config
make
make install INSTALLTOP=$PWD/../fips
cd ..
```

```
# Build the "FIPS capable" OpenSSL
gunzip -c openssl-1.0.1c.tar.gz | tar xf -
cd openssl-1.0.1c/
./config fips shared --with-fipsdir=$PWD/../fips
make depend
make

# Build the example program
arm-linux-androideabi-gcc -o fips_hmac fips_hmac.c \
  -Iopenssl-1.0.1c/include/ -Lopenssl-1.0.1c/ -lcrypto -Iopenssl-1.0.1c \
  -Iandroid-ndk-r7c/platforms/android-14/arch-arm/usr/include \
  -Bandroid-ndk-r7c/platforms/android-14/arch-arm/usr/lib

# Copy the program and shared library to the Android device
./android-sdk-linux/platform-tools/adb push fips_hmac /data/local/tmp/
./android-sdk-linux/platform-tools/adb push openssl-
1.0.1c/libcrypto.so.1.0.0 /data/local/tmp/

# Execute the program on the Android device
./android-sdk-linux/platform-tools/adb push fips_hmac shell
cd /data/local/tmp
LD_LIBRARY_PATH=openssl-1.0.1c ./fips_hmac -v fips_hmac.c
```

Appendix D FIPS API Documentation

D.1 FIPS Mode

NAME

FIPS mode - NIST FIPS 140-2 Approved mode of operation

DESCRIPTION

When built with the *fips* config option in accordance with some additional procedural requirements the OpenSSL FIPS Object Module can be used to satisfy requirements for FIPS 140-2 validated cryptography.

OVERVIEW

The OpenSSL FIPS Object Module must be built with the *fips* config option. The application must call FIPS_mode_set() to enable FIPS mode. When in FIPS mode only the FIPS approved encryption algorithms
are usable:

+RSA

+DSA

+3DES in CBC, (CFB1), CFB8, CFB64, ECB, OFB modes

+DH

+AES in CBC, (CFB1), CFB8, CFB128, ECB, OFB modes with 128/192/256 bit keys

+SHA-1, SHA-2

+HMAC

Other non-FIPS approved algorithms such a Blowfish, MD5, IDEA, RC4, etc. are disabled in FIPS mode.

To determine the mode of operation in a running program, an application can call FIPS_mode(3). A non-zero return indicates FIPS mode; a 0 indicates non-FIPS mode.

If the FIPS power-up self-test fails subsequent cryptographic operations are disabled and the application will have to exit.

To be considered FIPS 140-2 validated the OpenSSL FIPS Object Module must use the validated version of the FIPS specific OpenSSL source code.

While most platforms and applications can use the OpenSSL FIPS Object Module to satisfy NIST requirements for FIPS 140-2 validated cryptography there are additional additional requirements beyond the call to FIPS_mode_set(). A more complete discussion of the OpenSSL FIPS mode can be found in the *OpenSSL FIPS 140-2 Security Policy* which can be found at http://csrc.nist.gov/cryptval/140-1/140sp/140sp1051.pdf.

Information about FIPS 140 can be found at http://csrc.nist.gov/cryptval/.

NOTES

3DES is also known as TDEA, or Triple Data Encryption Algorithm.
The power-up self-test can take a significant amount of time on slower systems.

HISTORY

FIPS mode support was introduced in version 0.9 of OpenSSL.

SEE ALSO

FIPS_mode_set(3), FIPS_mode(3)

D.2 FIPS_mode_set(), FIPS_selftest()

NAME

FIPS_mode_set, FIPS_selftest - perform FIPS power-up self-test

SYNOPSIS

```
#include <openssl/crypto.h>

int FIPS_mode_set(int ONOFF)

int FIPS_selftest(void)
```

DESCRIPTION

FIPS_mode_set() enables the FIPS mode of operation for applications that have complied with all the provisions of the *OpenSSL FIPS 140-2 Security Policy*. Successful execution of this function call with non-zero **ONOFF** is the only way to enable FIPS mode. After verifying the integrity of the executable object code using the stored digest FIPS_mode_set() performs the power-up self-test.

When invoked with **ONOFF** of zero FIPS_mode_set() exits FIPS mode.

To determine the mode of operation in a running program, an application can call FIPS_mode(3). A non-zero return indicates FIPS mode; a 0 indicates non-FIPS mode.

FIPS_selftest() can be called at any time to perform the FIPS power-up self-test.

If the power-up self-test fails subsequent cryptographic operations are disabled. The only possible recovery is a successful re-invocation of FIPS_mode_set() which is unlikely to work unless the original path was incorrect.

RETURN VALUES

A return value of 1 indicates success, 0 failure.

SEE ALSO

FIPS_mode(3), ERR_get_error(3)

NOTES

FIPS_mode_set() and FIPS_selftest() were formerly included with <openssl/fips/fips.h>.

HISTORY

FIPS support was introduced in version 0.9 of OpenSSL.

D.3 FIPS_mode()

NAME

FIPS_mode – returns the current FIPS mode of operation.

SYNOPSIS

```
#include <openssl/crypto.h>

int FIPS_mode()
```

DESCRIPTION

FIPS_mode() is used to determine the FIPS mode of operation of the running program.

FIPS_mode() currently returns 1 to indicate FIPS mode. Future return values might include 2 to indicate exclusive use of the NSA's Suite B algorithms.

RETURN VALUES

A return code of non-zero indicates FIPS mode, 0 indicates non-FIPS mode.

SEE ALSO

FIPS_mode_set(3)

NOTES

FIPS_mode() was formerly included with <openssl/fips/fips.h>.

HISTORY

FIPS support was introduced in version 0.9 of OpenSSL.

D.4 Error Codes

In order to minimize the size of the FIPS module only numeric error codes are returned. When used in conjunction with a FIPS capable OpenSSL distribution these numeric codes will automatically be converted to the usual text format for display, but the FIPS specific standalone utilities print out numerical error codes. These can be interpreted with the **`openssl errstr`** command or by checking the source file at the referenced location:

```
$ ../util/shlib_wrap.sh ./fips_shatest
ERROR:2d06c071:lib=45,func=108,reason=113:file=fips.c:line=274:1,129d0
$
$ openssl errstr 2d06c071
error:2D06C071:FIPS routines:FIPS_mode_set:unsupported platform
$
```

These error codes are defined in the include file *fips_err.h*.

The **`FIPS_mode_set()`** call or other function calls in FIPS mode can return any of the following errors:

Return Code	Meaning and Comment
CRYPTO_R_FIPS_MODE_NOT_SUPPORTED	“fips mode not supported” You likely linked against a non-FIPS Capable library. Ensure `config fips <options>` was executed when configuring.
FIPS_R_CANNOT_READ_EXE	"cannot read exe"
FIPS_R_CANNOT_READ_EXE_DIGEST	"cannot read exe digest"
FIPS_R_CONTRADICTING_EVIDENCE	"contradicting evidence"
FIPS_R_EXE_DIGEST_DOES_NOT_MATCH	"exe digest does not match"
FIPS_R_FINGERPRINT_DOES_NOT_MATCH	"fingerprint does not match” The integrity test has failed.
FIPS_R_FINGERPRINT_DOES_NOT_MATCH_-NONPIC_RELOCATED	"fingerprint does not match nonpic relocated" This Microsoft Windows specific error indicates that there might be a DLL address conflict which needs to be addressed by re-basing the offending DLL.
FIPS_R_FINGERPRINT_DOES_NOT_MATCH_-SEGMENT_ALIASING	“fingerprint does not match segment aliasing" This error is returned when a defective compiler has merged **`.rodata`** (read-only) and **`.data`** (writable) segments. This situation effectively degrades the read-only status of constant tables and leaves them without hardware protection, thus jeopardizing the FIPS mode of operation.
FIPS_R_FIPS_MODE_ALREADY_SET	"fips mode already set"
FIPS_R_INVALID_KEY_LENGTH	"invalid key length"
FIPS_R_KEY_TOO_SHORT	"key too short"
FIPS_R_NON_FIPS_METHOD	"non fips method” Attempted non FIPS-compliant DSA usage.
FIPS_R_PAIRWISE_TEST_FAILED	"pairwise test failed" One or more of the algorithm pairwise consistency tests has failed.
FIPS_R_RSA_DECRYPT_ERROR	"rsa decrypt error"
FIPS_R_RSA_ENCRYPT_ERROR	"rsa encrypt error"
FIPS_R_SELFTEST_FAILED	"selftest failed" One or more of the algorithm known answer tests has failed.
FIPS_R_TEST_FAILURE	"test failure"
FIPS_R_UNSUPPORTED_PLATFORM	"unsupported platform" Indicates the validity of the digest test is unknown for the current platform.

Appendix E Platform Specific Notes

Note: the material present in this appendix for earlier versions of this document has been removed and relocated to http://www.openssl.com/fips/tech/.

E.1 Apple OS X Support

<TBD>

E.2 Apple iOS Support

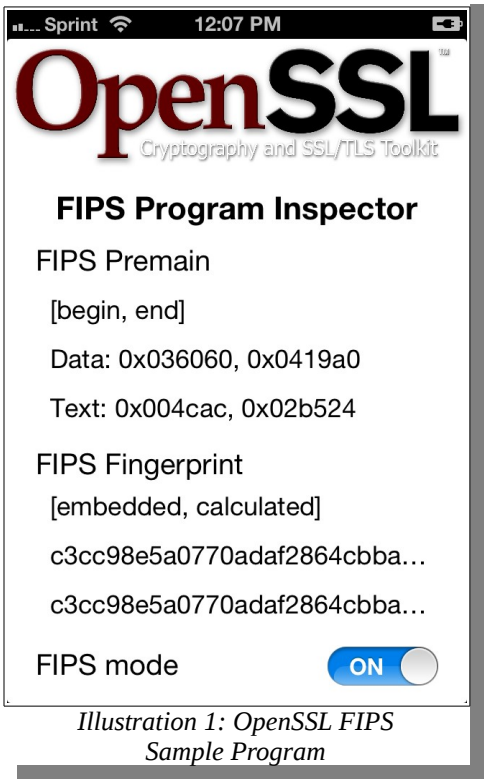

Illustration 1: OpenSSL FIPS Sample Program

OpenSSL fully supports building the FIPS Object Module and FIPS Capable library for iOS devices. There are five logical steps to build the OpenSSL FIPS Object Module and FIPS Capable Library for use in an Xcode/iOS project . The steps are outlined below:

1. Acquire the required files
2. Build the Incore utility
3. Build the FIPS Object Module
4. Build the FIPS Capable Library
5. Create an Xcode Project

The procedures for each logical step are detailed below. The sample Xcode project is offered at the end of the chapter.

Acquire Required Files

First, obtain the base files from http://www.openssl.org/source/:

- **`openssl-1.0.1c.tar.gz`**
- **`openssl-fips-2.0.1.tar.gz`**

Next, acquire the auxiliary files, which can be obtained from http://openssl.com/fips/2.0/platforms/ios/:

- **`setenv-reset.sh`**
- **`setenv-darwin-i386.sh`**
- **`setenv-ios-11.sh`**
- **`ios-incore-2.0.1.tar.gz`**

In addition to the required core files listed above, http://openssl.com/fips/2.0/platforms/ios/ includes a sample program:

- **`fips-pi.tar.gz`**

`openssl-fips-2.0.1.tar.gz` includes the FIPS Object Module.

openssl-1.0.1c.tar.gz has the FIPS Capable OpenSSL library.

ios-incore-2.0.1.tar.gz contains OS X and iOS specific Incore utility to determine the object code digest.

setenv-darwin-i386.sh and **setenv-ios-11.sh** are used to set the proper environments for the task at hand, while **setenv-reset.sh** is used to reset the environment.

> *Note: as of this writing (January, 2013), the scripts have a **PWD** dependency and do not alert the user of failures such as missing or errant paths. I was not able to get hardened/updated scripts placed on web for download. Please accept my sincerest apologies (JW).*

After collecting the required files, your working directory will look similar to below.

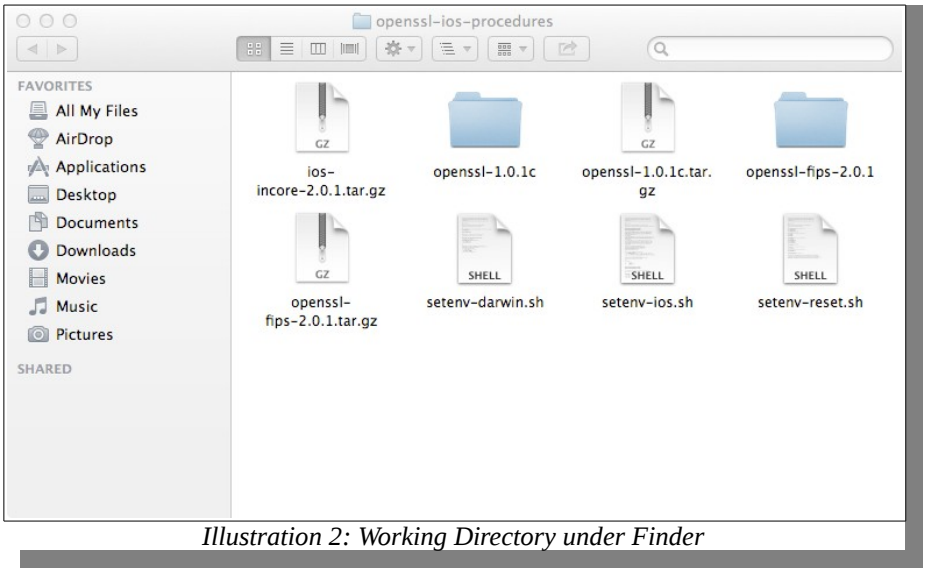

Illustration 2: Working Directory under Finder

After acquiring the files, perform the following in the working directory to remove quarantine bit and ensure the execute bit is set:

```
$ xattr  -r -d "com.apple.quarantine" *.tar.gz *.sh
$ chmod  +x *.sh
```

Build the Incore Utility

The Incore utility is a native application used to embed the FIPS Object Module's fingerprint in the ARM library. Building Incore is a two step process – first, build a native version of **libcrypto.a**, and then build Incore using the previously built native **libcrypto.a**.

To compile the **incore_macho** utility for the native platform, perform the following steps:

```
$ rm -rf  openssl-fips-2.0.1/              (delete old artifacts)

$ tar xzf  openssl-fips-2.0.1.tar.gz         (unpack fresh files)
$ tar xzf  ios-incore-2.0.1.tar.gz

$ .  ./setenv-reset.sh                  (note the leading dot ".")
$ .  ./setenv-darwin-i386.sh            (note the leading dot ".")

$ cd  openssl-fips-2.0.1/              (perform `cd` after setenv)

$ ./config                                (several screens of output)
$ make                           (build libcrypto.a, lots of output)

$ cd  iOS/                          (switch to incore's subdirectory)
$ make                          (build incore_macho, lots of output)
```

Note: as of this writing (January, 2013), ***setenv-darwin-i386.sh*** *could silently fail due to* ***PWD*** *dependencies. Please execute the* ***`env`*** *command and verify the paths placed in the environment by the script.*

Confirm the utility works:

```
$ ./incore_macho
usage:
        ./incore_macho [--debug] [-exe|-dso] executable
```

If the utility does not work, delete the **openssl-fips-2.0.1/** directory and start over.

Once the utility has been verified on the native platform, install the **incore_macho** utility in a location on path, such as **/usr/local/bin**. The instructions below offer a second choice, and place **incore_macho** in your home directory.

```
$ mkdir "$HOME/bin"
$ cp incore_macho "$HOME/bin"
$ PATH="$HOME/bin":$PATH
```

Finally, delete the **openssl-fips-2.0.1/** directory in preparation for the ARM build of the FIPS Capable library. This is done to keep cross contamination to a minimum since **openssl-fips-2.0.1/** is essentially reused.

```
$ cd ..
$ rm -rf openssl-fip-2.0.1/
```

This instructions from this point assume the build environment has been prepared, including the creation of the **incore_macho** utility, as documented in the previous section, and that **incore_macho** is on path.

Build the FIPS Object Module

This section of the document will guide you through the creation of the FIPS Object Module. The Module is governed by the FIPS 140-2 program requirements and you cannot deviate from the Security Policy during any stage during handling, from acquisition, through building, to installation. In case of a discrepancy between this document and the Security Policy, the Security Policy will prevail.

While these commands look similar to those recently executed for the generation of the **incore_macho** utility, there are subtle differences. This time you are cross-compiling for the an iOS device.

While it is not readily apparent, the iOS tools used via **IOS_TOOLS** environmental variable are available from **ios-incore-2.0.1.tar.gz**, so you must unpack it again. The tools unpack into **openssl-fips-2.0.1/**.

```
$ rm  -rf  openssl-fips-2.0.1/              (delete old artifacts)
$ tar  xzf  openssl-fips-2.0.1.tar.gz          (unpack fresh files)
$ tar  xzf  ios-incore-2.0.1.tar.gz            (unpack fresh files)

$ cd   openssl-fips-2.0.1/                    (perform `cd` first)

$ .  ../setenv-reset.sh                 (note the leading dot ".")
$ .  ../setenv-ios-11.sh                (note the leading dot ".")

$ llvm-gcc   -v                         (verify expected compiler)
Using built-in specs.
Target: i686-apple-darwin10
Configured with:/private/var/tmp/llvmgcc42_Embedded/
llvmgcc42_Embedded-2377~4/src/configure
...
```

```
gcc version 4.2.1 (Based on Apple Inc. build 5658)
(LLVM build 2377.00)
```

Note: as of this writing (January, 2013), `setenv-ios-11.sh` could silently fail due to `PWD` dependencies. Please execute the `` `env` `` command and verify the paths placed in the environment by the script.

The output of interest from **`llvm-gcc -v`** are (1) **`llvm-gcc`** is on path; (2) gcc version 4.2.1; and (3) the compiler is for an embedded platform.

At this point you are ready to commence the standard FIPS canister build for the target platform. Note that "fips canister" is implied, so there is no need for either **`./config fipscanisterbuild`** or **`./config fips`** (nor is it allowed by the Security Policy).

```
$ ./config                                  (several screens of output)
$ make                                                  (lots of output)
```

Confirm the binaries are for the iOS target device:

```
$ lipo  -info ./fips/fipscanister.o
Non-fat file: ./fips/fipscanister.o is architecture: armv7
```

After confirming the target architecture, complete the installation procedure by performing an install:

```
$ sudo  make  install
```

The default installation directory is **`/usr/local/ssl/Release-iphoneos/`**.

After installation, delete the **`openssl-fips-2.0.1/`** directory since its no longer needed:

```
$ rm  -rf  openssl-fips-2.0.1/
```

Recall from Section 2.4.2 Object Module (Link Time) Integrity that applications link against **`libcrypto.a`**, and not directly to f**`ipscanister.o`**. You will build **`libcrypto.a`** and **`libssl.a`** next in Build the FIPS Capable Library[59].

Build the FIPS Capable Library

This section of the document will guide you through the creation of the The FIPS Capable Library. The capable library is a standard OpenSSL distribution that is "FIPS Aware". The "aware" library handles all the details of operation while in FIPS mode after you successfully call **`FIPS_mode_set()`** (see D.2 FIPS_mode_set(), FIPS_selftest()). If you don't call

[59]There is some hand waiving here, but the details are not important at the moment for these procedures.

`FIPS_mode_set()`, the library will still operate as expected; but it will not be using validated cryptography.

Recall the FIPS Object Module is governed by the FIPS 140-2 program requirements, and you could not deviate from the Security Policy. The FIPS Capable Library does not endure the same requirements, and you are free to modify the environment and sources within reason.

To build the FIPS Capable library, you must issue **`./config fips`**, but other options are up to you. Some suggested options for configure include:

Option	Comment
--openssldir	Base of the OpenSSL installation. Default value is `--openssldir=/usr/local/ssl/Release-iphoneos`
--with-fipsdir	Location of fipscanister.o, if not located at `/usr/local/ssl/Release-iphoneos/lib`.
-no-sslv2	Disable SSLv2. SSLv2 is defective[60]
-no-sslv3	Disable SSLv3. SSLv3 is defective[61]
-no-comp	Disable compression independent of zlib. Compression is known to leak session information via CRIME attacks[62]
-no-shared	Disable shared library output. Apple only allows static linking, and dynamic linking is not supported on iOS.
-no-dso	Disable the OpenSSL DSO API (the library offers a shared object abstraction layer). iOS only uses static linking.
-no-hw	Disable hardware support.
-no-engines	Disable engine support.

To begin, clean old artifacts and set the environment for cross compilation.

```
$ rm  -rf  openssl-1.0.1c/                  (delete old artifacts)
$ tar  xzf  openssl-1.0.1c.tar.gz             (unpack fresh files)

$ cd   openssl-fips-1.0.1c/                   (perform `cd` first)

$ .  ../setenv-reset.sh                  (note the leading dot ".")
$ .  ../setenv-ios-11.sh                 (note the leading dot ".")
```

[60] Bruce Schneier and David Wagner, *Analysis of the SSL 3.0 Protocol*, www.schneier.com/paper-ssl-revised.pdf
[61] Loren Weith, *Differences Between SSLv2, SSLv3, and TLS*, http://www.yaksman.org/~lweith/ssl.pdf
[62] Mozilla's NSS accidentally disabled compression long before CRIME attacks due to compile/link conflicts (https://bugzilla.mozilla.org/show_bug.cgi?id=580679). Mozilla's Firefox did not support compression on clients. Many other browsers, such as Android (com.android.browser), did not support compression.

Next, configure and make the FIPS Capable library, where you pick your favorite options. No options are also acceptable:

```
$ ./config  fips  <options>            (several screens of output)
$ make  <options>                                  (lots of output)
```

Confirm the binaries are for the iOS target device:

```
$ lipo  -info  ./libcrypto.a  ./libssl.a
Non-fat file: ./libcrypto.a is architecture: armv7
Non-fat file: ./libssl.a is architecture: armv7
```

After confirming the target architecture, complete the installation procedure by performing an install:

```
$ sudo  make  install
```

The default installation directory is **/usr/local/ssl/Release-iphoneos/**.

After installation, delete the **openssl-fips-2.0.1/** directory since its no longer needed:

```
$ rm  -rf  openssl-fips-1.0.1c/
```

You might encounter issues due to the configuration options. The issues have been cleared in the version control system, but the tarballs maybe dated. If so, the issues and the fixes are listed below. Recall you have latitude in changing source files because the OpenSSL FIPS Capable Library is outside the Cryptographic Module (CM) boundary.

Issue	Remedy
Built-in tools not on path	Open **setenv-ios-11.sh**, and change the **CROSS_COMPILE** variable to **CROSS_COMPILE="$CROSS_CHAIN"**
No valid iOS SDK	Open the **setenv-ios-11.sh**, and change the **for** loop to include 6.2, 6.1, and 6.0
makedepend: warning: cannot open "armv7" **makedepend: error: ...**	Open the **Makefile**, and change **MAKEDEPPROG=makedepend** to **MAKEDEPPROG=$(CC) -M**
Undefined symbols for architecture armv7: "_ERR_load_COMP_strings"	Open **err_all.c**, and delete **all** declarations of **ERR_load_COMP_strings()**

OpenSSL Xcode Application

OpenSSL offers a sample Xcode project to test your installation. The minimal project demonstrates linking against the FIPS Capable Library, enabling FIPS Mode, disabling FIPS mode, displaying the embedded and calculated fingerprint, and displaying critical values from **fips_premain.c**. A screen capture from the device is shown in Illustration 1: OpenSSL FIPS Sample Program.

The essence of the sample code is shown in the listing below. The code toggles FIPS mode by way of **FIPS_mode()** and **FIPS_mode_set();** and retrieves error information via **ERR_get_error**(). The functions are available from **<openssl/crypto.h>** and **<openssl/err.h>** respectively. In the case of an error, error values were discussed in Appendix D FIPS API Documentation.

```
int mode = FIPS_mode(), ret = 0;
unsigned long err = 0;

if(mode == 0)
{
    ret = FIPS_mode_set(1 /*on*/);
    err = ERR_get_error();
}
else
{
    ret = FIPS_mode_set(0 /*off*/);
    err = ERR_get_error();
}

if(1 != ret)
    DisplayError("FIPS_mode_set failed", err);

...
```

After creating an Xcode project, you must add **fips_premain.c** to the project. Copy **fips_premain.c** from its location at /**usr/local/ssl/Release-iphoneos/lib/** into your project's working directory. Since the file is outside the Cryptographic Module (CM) boundary, you can check it in to revision control and even modify it if desired (within reason).

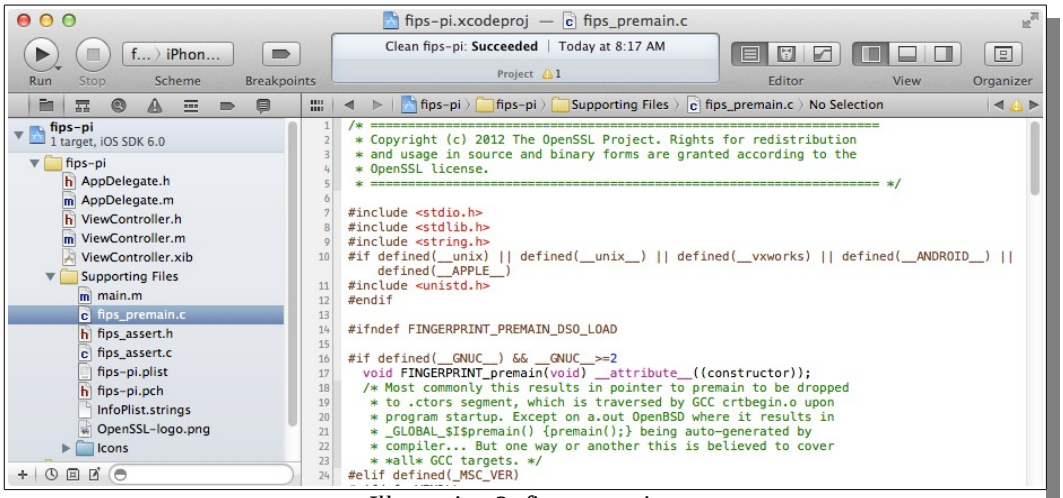

Illustration 3: fips_premain.c

The Xcode Build Settings to compile an OpenSSL dependent program are discussed below. The Build Setting should be set on the **Project**, and not the Target (all targets inherit from the project). The sample project has screen captures of the relevant changes under Xcode in the top level **`settings/`** directory.

Build Setting	Value
Architectures (**`ARCHS`**)	armv7 (remove armv6 and/or armv7s, unless you built for the architecture).
Always Search User Paths (**`ALWAYS_SEARCH_USER_PATHS`**)	Yes (due to **`#include <openssl/crypto.h>`** in non-standard location)
User header Search Paths (**`USER_HEADER_SEARCH_PATHS`**)	/usr/local/ssl/Release-iphoneos/include/
Other Linker Flags (**`OTHER_LDFLAGS`**)	/usr/local/ssl/Release-iphoneos/libcrypto.a (use the fully specified pathname, without **`-l`** or **`-L`**)
Build Setting	**Value**
Valid Architectures (**`VALID_ARCHS`**)	armv7 (remove armv6 and/or armv7s, unless you built for the architecture).
Always Search User Paths (**`ALWAYS_SEARCH_USER_PATHS`**)	Yes (due to **`#include <openssl/crypto.h>`** in non-standard location)
User header Search Paths (**`USER_HEADER_SEARCH_PATHS`**)	/usr/local/ssl/Release-iphoneos/include/
Other Linker Flags (**`OTHER_LDFLAGS`**)	/usr/local/ssl/Release-iphoneos/libcrypto.a (use the fully specified filename, without **`-l`** or **`-L`**)

The final modification is a Build Phase Script on the **Target** (not the Project) to embed the Module's expected signature using **`incore_macho`**. The full command to embed the signature is **`/usr/local/bin/incore_macho -exe "$CONFIGURATION_BUILD_DIR/$EXECUTABLE_PATH"`**.

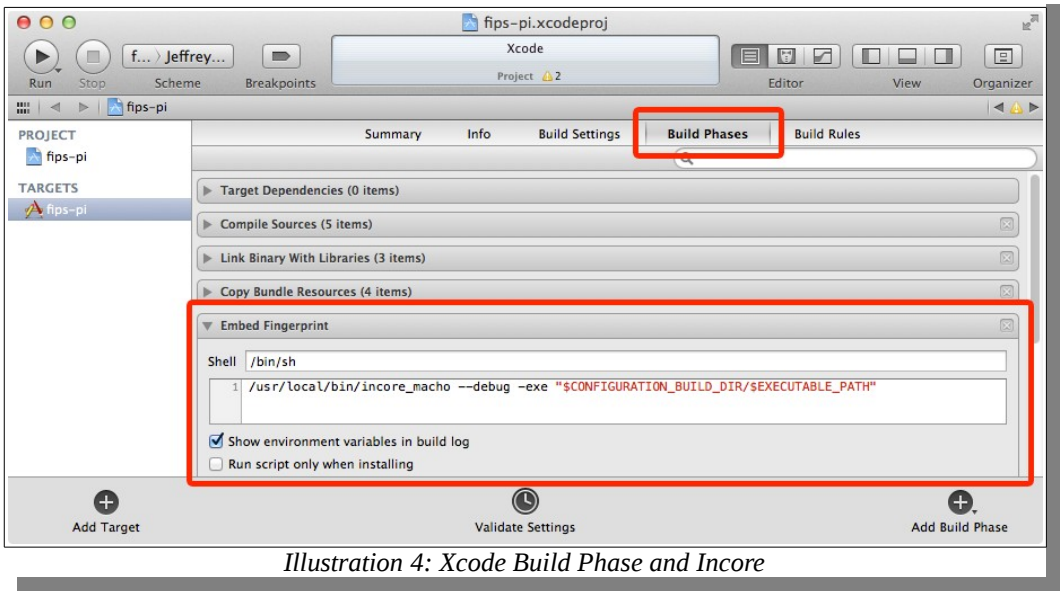

Illustration 4: Xcode Build Phase and Incore

E.3 Windows CE Support

NOTE: This section is incomplete

The Microsoft Windows mobile operating systems are among the most challenging platform for the FIPS Object Module, due to the wide variation among individual system configurations.
Representative Build

These instructions are necessarily only representative of one specific configuration and may require substantial modification for specific Windows CE or EC platforms.

Typically a version of Visual Studio will be used. In this representative example the following environment variables are defined in a .BAT file, *setenv-wince6.bat*:

```
@rem
@rem setenv_wince.cmd
@rem
@rem Paths for Visual Studio 2008 on command line (on-64-bit-
host)
```

```
@call "c:\Program Files\Microsoft Visual Studio
9.0\VC\"vcvarsall.bat

@set OSVERSION=WCE600
@set PLATFORM=MACKEREL
@set TARGETCPU=ARMV4I

@set WCECOMPAT=C:\wcecompat
@SET MACKERELSDK=C:\Program Files\Windows CE
Tools\wce600\Mackerel SDK

@set PATH=%VSINSTALLDIR%\Common7\IDE;%VCINSTALLDIR
%\ce\bin\x86_arm;%VCINSTALLDIR%\bin;%NASMINSTALLDIR%;%PATH%
@set INCLUDE=%MACKERELSDK%\Include\Armv4i;%VCINSTALLDIR
%\ce\include;%INCLUDE%
@set LIB=%MACKERELSDK%\Lib\ARMV4I;%VCINSTALLDIR
%\ce\lib\armv4i;%LIB%
@set LIBPATH=%MACKERELSDK%\Lib\ARMV4I;%VCINSTALLDIR
%\ce\lib\armv4i;%LIBPATH%

@set FIPS_SHA1_PATH=perl /openssl-fips-
2.0/util/fips_standalone_sha1
@set FIPS_SIG=perl /openssl-fips-2.0/util/msincore
```

On the Windows build system, invoke a DOS Command Prompt and in that shell enter the following:

```
X:\>setenv-wince6
X:\>cl
Microsoft (R) C/C++ Optimizing Compiler Version 15.00.20720
for ARM
Copyright (C) Microsoft Corporation. All rights reserved.

usage: cl [ option... ] filename... [ /link linkoption... ]

X:\>
X:\>cd  openssl-fips-2.0
X:\openssl-fips-2.0>ms\do_fips
X:\openssl-fips-2.0>nmake  -f ms\cedll.mak  build_algvs
```

In either case a "Press any key to continue . . . " prompt will be seen.

At this point the FIPS Object Module and *fips_algvs* utility program have been created.

General Considerations

DLLs present on CE versions prior to 6.0 take away a portion of precious 32MB address space from *all* processes[63]. This means that unlike "normal" Windows, where DLL load address availability is a per-process attribute, it's a per-system attribute for CE pre-6.0. In more practical terms the determination of the load address can be dependent on the order in which processes are started. In general the static link method is preferred on CE, unless the DLL is ROM-based, and use of ce[dll].mak instead of nt[dll].mak.

Note that the two-step link is not necessary for Windows, as use of the *msincore* utility after a conventional link is sufficient. For the runtime integrity test (fingerprint verification) to succeed a binary module, either .exe or .dll, must be loaded at a predefined address or not contain any relocations. As there is virtually no control over the load address for CE, fingerprint verification in a DLL will fail. The only solution is to statically link the FIPS Object Module into an .exe executable and not as a DLL.

The build for the formally tested Win CE 5 platform used a ROM-based DLL and some flags set in Platform Builder. A normal DLL would not work as it ignored the load address and setting /FIXED stopped it loading altogether.

Note the *fipslink.pl* utility can handle even statically linked applications.

Note that Windows and Linux cannot be compared in this context, because Linux can generate position-independent code which means we avoid any difficulties with base addresses, relocations, etc. For Windows a consistent load address is needed for the DLL. If that DLL isn't ROM-based then things like the load order can result in different addresses which will result in an invalid signature.

So one (messy) solution is to set up platform builder to get that consistent load address: as long as it doesn't change it doesn't matter what it is. The process viewer tool can be used to check the load address. Then once a fixed address has been established it can be used to build the FIPS capable OpenSSL to embed the signature; this is the `--with-baseaddr=<address>` option to Configure.

[63] CE DLLs steal memory from all processes, so if only one application needs to operate in validated mode then a statically linked module is preferable.

Appendix F Restrictions on the Export of Cryptography

Government restrictions and regulations on the use, acquisition, and distribution of cryptographic products are a matter of concern for some potential users.

F.1 Open Source Software

In the United States the current export regulations appear to more or less leave open source software in source code format alone, except for a reporting requirement to the Bureau of Industry and Security (BIS) of the U.S. Department of Commerce; see http://bxa.doc.gov/Encryption/pubavailencsourcecodenofify.html.

When in doubt consultation with legal experts would be appropriate. An example of an E-mail message sent to comply with this reporting requirement is:

```
To: crypt@bis.doc.gov, enc@nsa.gov, web_site@bis.doc.gov
Subject: TSU NOTIFICATION

SUBMISSION TYPE: TSU

SUBMITTED BY: Steve Marquess
SUBMITTED FOR: OpenSSL Software Foundation, Inc.
POINT OF CONTACT: Steve Marquess
PHONE and/or FAX: 877-673-6775
MANUFACTURER: N/A
PRODUCT NAME/MODEL #: OpenSSL
ECCN: 5D002

NOTIFICATION: http://cvs.openssl.org/dir

Employee(s), subcontractor(s), and/or agent(s) of the OpenSSL Software
Foundation, Inc. (OSF) are participating in the development of the freely
available open source OpenSSL product by providing feedback on new
releases, by requesting new features, by correspondence either to the
developer and user mailing lists or directly with the product developers,
and by subcontracting software development services to one or more of
the OpenSSL developers.  This correspondence may include suggested
source code fragments or patches.  All versions of any such contributions
incorporated, or software implemented, in any of the OpenSSL software
will be publicly accessible at http://cvs.openssl.org/dir.

--
Steve Marquess
OpenSSL Software Foundation, Inc.
1829 Mount Ephraim Road
Adamstown, MD  21710
USA
+1 877-673-6775
```

marquess@marquess@openssl.com

No response was received (or expected).

Other links of interest:

http://bxa.doc.gov/Encryption/ChecklistInstr.htm

F.2 "Export Jobs, Not Crypto"

For software exported in binary form the situation is far less certain. As incredible and unbelievably opposed to common sense as it seems, current U.S. export controls appear to restrict the export from the U.S. of software products that use the OpenSSL product, even if OpensSSL is used exclusively for all cryptographic functionality.

From what has been relayed from several vendors affected by these export restrictions, export approval for software utilizing OpenSSL is contingent on a number of factors including the type of linking (static build-time linking or dynamic run-time linking). Static linking is more desirable, apparently something to do with the concept of an "open cryptographic interface". Evidently a product where the end user can easily substitute a new cryptographic library (a newer version of OpenSSL, say) is not permissible.

Needless to say the written regulations and expert commentary are varied, so advice of legal counsel is recommended. The only other safe course of action would be to pay non-U.S. citizens to develop the cryptographic software overseas and *import* it into the U.S., as imports are not restricted. Foreigners who benefit financially from this situation refer to the U.S. "export jobs, not crypto" policy.

Links of interest:

http://www.axsmith.com/Encryption_Law.htm
http://library.findlaw.com/2000/Jan/1/128443.html
http://cryptome.org/bxa-bernstein.htm

APPENDIX G Security Policy Errata

The formal Security Policy (http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1747.pdf is a controlled document and so, as with the validated software proper, cannot readily be changed. This section lists known errors in that document.

- Table 2: The operating system for platform 9 is listed as "Android 2.2". That device was the Motorola Xoom running Android 3.0, the earliest version of Android that device shipped with. During the period the validation was in process that version of Android on that device was superseded by Android 4.0 which was tested as platform 39, so platform 9 is of academic interest only (note platform essentially 9 duplicates platform 2). The error was reported to the test lab even prior to the formal validation award, but since correction of errors in completed validations is difficult we elected not to press the issue.

Appendix H DTR Analysis

[TBD]

Appendix I API Entry Points by Source File

The API entry points in the Module are listed here, organized by source file.

FIPS 140-2 requires that logical interfaces have to be identified as one of "data input", "data output", "control input", or "status output". Functions with multiple arguments and the C language argument passing mechanism do not naturally match these categories, especially where pointers to structures are used. This table designates each function as primarily serving one of the four purposes, with the individual arguments also designated as input, output, or both.

The function names are in **bold**. Input arguments are highlighted in Grey and listed with a right pointing arrow (->). Output arguments are listed with a left pointing arrow (<-). Pointer arguments referencing structures containing both input and output data elements are listed with a double arrow (<->). The function return value is denoted in the list of arguments as "Return".

Note that many of these Module API functions calls are rarely if ever referenced directly by applications, instead they are referenced from the separate OpenSSL product by a non-cryptographic abstraction layer such as the EVP interface (see Reference 11). Some external symbols defined in the Module but not intended for reference by calling applications are omitted. Also note that the API as documented below may vary slightly by platform due to the use of assembly language optimizations.

Some general notes:

The POST code is contained in the *./fips/* subdirectory, beginning with the **`FIPS_module_mode_set()`** function in *./fips/fips.c* and leading directly functions defined in *./fips/fips_post.c*.

The best way to trace each of the algorithm implementations is from the respective algorithm test drivers, as they start with the CAVS test vector request file data and make the appropriate API calls to perform the algorithm processing. Those are found in the *./fips/XXX/* directories, for "*XXX*" the algorithm, and are also symlinked from the *./test/* subdirectory:

test/fips_aesavs.c -> ../fips/aes/fips_aesavs.c
test/fips_cmactest.c -> ../fips/cmac/fips_cmactest.c
test/fips_desmovs.c -> ../fips/des/fips_desmovs.c
test/fips_dhvs.c -> ../fips/dh/fips_dhvs.c
test/fips_drbgvs.c -> ../fips/rand/fips_drbgvs.c
test/fips_dsatest.c -> ../fips/dsa/fips_dsatest.c
test/fips_dssvs.c -> ../fips/dsa/fips_dssvs.c
test/fips_ecdhvs.c -> ../fips/ecdh/fips_ecdhvs.c
test/fips_ecdsavs.c -> ../fips/ecdsa/fips_ecdsavs.c

test/fips_gcmtest.c -> ../fips/aes/fips_gcmtest.c
test/fips_hmactest.c -> ../fips/hmac/fips_hmactest.c
test/fips_randtest.c -> ../fips/rand/fips_randtest.c
test/fips_rngvs.c -> ../fips/rand/fips_rngvs.c
test/fips_rsagtest.c -> ../fips/rsa/fips_rsagtest.c
test/fips_rsastest.c -> ../fips/rsa/fips_rsastest.c
test/fips_rsavtest.c -> ../fips/rsa/fips_rsavtest.c
test/fips_shatest.c -> ../fips/sha/fips_shatest.c

Note the algorithm test drivers themselves are not part of the FIPS module.

Symbol renaming: Some symbol names as defined in the source code are dynamically redefined at build time. This API documentation shows both the original (source code) and build time (object code) symbol names, for instance:

```
FIPS_bn_bn2bin (renames BN_bn2bin) in file
                ./crypto/bn/bn_lib.[o|c]
```

which indicates that the *FIPS_bn_bn2bin()* function as seen in the compiled code (*./crypto/bn/bn_lib.o*) is found in the source code as function *BN_bn2bin()* in source file *./crypto/bn/bn_lib.c*.

Some functions are not renamed, for instance:

```
FIPS_module_mode_set in file ./fips/fips.[o|c]
```

indicates that *FIPS_module_mode_set()* is defined in *./fips/fips.c* and *./fips/fips.o* with the same symbol name. Likewise,

```
FIPS_add_lock (reimplements CRYPTO_add_lock) in file
          ./fips/utl/fips_lck.[o|c]
```

indicates that **`FIPS_add_lock()`** is defined by that name in both *./fips/utl/fips_lck.o* and *./fips/utl/fips_lck.c*; the "reimplements" notation refers to the redefinition of this FIPS module specific function to replace a similar known function from the original OpenSSL distribution from which the FIPS module was derived.

This list was produced by the **`api_list.pl`** tool in the *./fips/tools/* subdirectory of the source code distribution, using supporting files also in that directory:

`api_fns.pm`	a perl module that for api_list.pl
`declarations.dat`	a file of information about public fips symbols

This utility attempts to "direction of use" for each function parameter, i.e. whether that parameter is referenced as input, as output, or both. That determination is far from clear in some cases, as for some types of parameters there is no clear answer -- consider for instance a pointer to a structure containing a callback to a function that is only called as an exception. In any event that information is stored in the file **`declarations.dat`** and can be manually corrected by replacing the value for the key 'direction' where the value contains a question mark. Those values can be changed as appropriate, to one of:

```
<-      output
->      input
<->     both
```

and the manually changed values will be preserved in the **`declarations.dat`** file. The **`api_list.pl`** utility has no command line options and is invoked from the root of the source code work area:

```
perl fips/tools/api_list.pl > <outfile>
```

The HTML formatted contents of the output file can be lightly edited for inclusion in documents such as this one.

This following list shows the functions in alphabetical order by the runtime symbol name.

FIPS_add_error_data (reimplements ERR_add_error_data) in file ./fips/utl/fips_err.[o|c]

void **FIPS_add_error_data**(int num, ...)
-> num
-> ...

FIPS_add_lock (reimplements CRYPTO_add_lock) in file ./fips/utl/fips_lck.[o|c]

int **FIPS_add_lock**(int *pointer, int amount, int type, const char *file, int line)
<- pointer
-> amount
-> type
-> file
-> line
<- Return

FIPS_bn_bin2bn (renames BN_bin2bn) in file ./crypto/bn/bn_lib.[o|c]

BIGNUM ***FIPS_bn_bin2bn**(const unsigned char *s, int len, BIGNUM *ret)
-> s
-> len
<-> ret
<- Return

FIPS_bn_bn2bin (renames BN_bn2bin) in file ./crypto/bn/bn_lib.[o|c]

int **FIPS_bn_bn2bin**(const BIGNUM *a, unsigned char *to)
-> a
<- to
<- Return

FIPS_bn_clear (renames BN_clear) in file ./crypto/bn/bn_lib.[o|c]

void **FIPS_bn_clear**(BIGNUM *a)
<-> a

FIPS_bn_clear_free (renames BN_clear_free) in file ./crypto/bn/bn_lib.[o|c]

void **FIPS_bn_clear_free**(BIGNUM *a)
<-> a

FIPS_bn_free (renames BN_free) in file ./crypto/bn/bn_lib.[o|c]

void **FIPS_bn_free**(BIGNUM *a)
<-> a

FIPS_bn_generate_prime_ex (renames BN_generate_prime_ex) in file ./crypto/bn/bn_prime.[o|c]

int **FIPS_bn_generate_prime_ex**(BIGNUM *ret, int bits, int safe, const BIGNUM *add, const BIGNUM *rem, BN_GENCB *cb)
<-> ret
-> bits
-> safe
-> add

-> rem
<-> cb
<- Return

FIPS_bn_get_word (renames BN_get_word) in file ./crypto/bn/bn_lib.[o|c]

BN_ULONG **FIPS_bn_get_word**(const BIGNUM *a)
-> a
<- Return

FIPS_bn_is_bit_set (renames BN_is_bit_set) in file ./crypto/bn/bn_lib.[o|c]

int **FIPS_bn_is_bit_set**(const BIGNUM *a, int n)
-> a
-> n
<- Return

FIPS_bn_is_prime_ex (renames BN_is_prime_ex) in file ./crypto/bn/bn_prime.[o|c]

int **FIPS_bn_is_prime_ex**(const BIGNUM *p, int nchecks, BN_CTX *ctx, BN_GENCB *cb)
-> p
-> nchecks
<- ctx
<-> cb
<- Return

FIPS_bn_is_prime_fasttest_ex (renames BN_is_prime_fasttest_ex) in file ./crypto/bn/bn_prime.[o|c]

int **FIPS_bn_is_prime_fasttest_ex**(const BIGNUM *p, int nchecks, BN_CTX *ctx, int do_trial_division, BN_GENCB *cb)
-> p
-> nchecks
<- ctx
-> do_trial_division
<-> cb
<- Return

FIPS_bn_new (renames BN_new) in file ./crypto/bn/bn_lib.[o|c]

BIGNUM ***FIPS_bn_new**()
<- Return

FIPS_bn_num_bits (renames BN_num_bits) in file ./crypto/bn/bn_lib.[o|c]

int **FIPS_bn_num_bits**(const BIGNUM *a)
-> a
<- Return

FIPS_bn_num_bits_word (renames BN_num_bits_word) in file ./crypto/bn/bn_lib.[o|c]

int **FIPS_bn_num_bits_word**(BN_ULONG l)
-> l
<- Return

FIPS_bn_pseudo_rand (renames BN_pseudo_rand) in file ./crypto/bn/bn_rand.[o|c]

int **FIPS_bn_pseudo_rand**(BIGNUM *rnd, int bits, int top, int bottom)
<-> rnd
-> bits
-> top
-> bottom
<- Return

FIPS_bn_pseudo_rand_range (renames BN_pseudo_rand_range) in file ./crypto/bn/bn_rand.[o|c]

int **FIPS_bn_pseudo_rand_range**(BIGNUM *rnd, const BIGNUM *range)
<-> rnd
-> range
<- Return

FIPS_bn_rand (renames BN_rand) in file ./crypto/bn/bn_rand.[o|c]

int **FIPS_bn_rand**(BIGNUM *rnd, int bits, int top, int bottom)
<-> rnd
-> bits

-> top
-> bottom
<- Return

FIPS_bn_rand_range (renames BN_rand_range) in file ./crypto/bn/bn_rand.[o|c]

int **FIPS_bn_rand_range**(BIGNUM *rnd, const BIGNUM *range)
<-> rnd
-> range
<- Return

FIPS_bn_set_bit (renames BN_set_bit) in file ./crypto/bn/bn_lib.[o|c]

int **FIPS_bn_set_bit**(BIGNUM *a, int n)
<-> a
-> n
<- Return

FIPS_bn_x931_derive_prime_ex (renames BN_X931_derive_prime_ex) in file ./crypto/bn/bn_x931p.[o|c]

int **FIPS_bn_x931_derive_prime_ex**(BIGNUM *p, BIGNUM *p1, BIGNUM *p2, const BIGNUM *Xp, const BIGNUM *Xp1, const BIGNUM *Xp2, const BIGNUM *e, BN_CTX *ctx, BN_GENCB *cb)
<-> p
<-> p1
<-> p2
-> Xp
-> Xp1
-> Xp2
-> e
<- ctx
<-> cb
<- Return

FIPS_bn_x931_generate_prime_ex (renames BN_X931_generate_prime_ex) in file ./crypto/bn/bn_x931p.[o|c]

int **FIPS_bn_x931_generate_prime_ex**(BIGNUM *p, BIGNUM *p1, BIGNUM *p2, BIGNUM

*Xp1, BIGNUM *Xp2, const BIGNUM *Xp, const BIGNUM *e, BN_CTX *ctx, BN_GENCB *cb)
<-> p
<-> p1
<-> p2
<-> Xp1
<-> Xp2
-> Xp
-> e
<- ctx
<-> cb
<- Return

FIPS_bn_x931_generate_xpq (renames BN_X931_generate_Xpq) in file ./crypto/bn/bn_x931p.[o|c]

int **FIPS_bn_x931_generate_xpq**(BIGNUM *Xp, BIGNUM *Xq, int nbits, BN_CTX *ctx)
<-> Xp
<-> Xq
-> nbits
<- ctx
<- Return

FIPS_check_incore_fingerprint in file ./fips/fips.[o|c]

int **FIPS_check_incore_fingerprint**()
<- Return

FIPS_cipher (reimplements EVP_Cipher) in file ./fips/utl/fips_enc.[o|c]

__owur int **FIPS_cipher**(EVP_CIPHER_CTX *c, unsigned char *out, const unsigned char *in, unsigned int inl)
<- c
<- out
-> in
-> inl
<- Return

FIPS_cipher_ctx_cleanup (reimplements EVP_CIPHER_CTX_cleanup) in file ./fips/utl/fips_enc.[o|c]

int **FIPS_cipher_ctx_cleanup**(EVP_CIPHER_CTX *a)
<- a
<- Return

FIPS_cipher_ctx_copy (reimplements EVP_CIPHER_CTX_copy) in file ./fips/utl/fips_enc.[o|c]

int **FIPS_cipher_ctx_copy**(EVP_CIPHER_CTX *out, const EVP_CIPHER_CTX *in)
<- out
-> in
<- Return

FIPS_cipher_ctx_ctrl (reimplements EVP_CIPHER_CTX_ctrl) in file ./fips/utl/fips_enc.[o|c]

int **FIPS_cipher_ctx_ctrl**(EVP_CIPHER_CTX *ctx, int type, int arg, void *ptr)
<- ctx
-> type
-> arg
<-> ptr
<- Return

FIPS_cipher_ctx_free (reimplements EVP_CIPHER_CTX_free) in file ./fips/utl/fips_enc.[o|c]

void **FIPS_cipher_ctx_free**(EVP_CIPHER_CTX *a)
<- a

FIPS_cipher_ctx_init (reimplements EVP_CIPHER_CTX_init) in file ./fips/utl/fips_enc.[o|c]

void **FIPS_cipher_ctx_init**(EVP_CIPHER_CTX *a)
<- a

FIPS_cipher_ctx_new (reimplements EVP_CIPHER_CTX_new) in file ./fips/utl/fips_enc.[o|c]

EVP_CIPHER_CTX ***FIPS_cipher_ctx_new**()
<- Return

FIPS_cipher_ctx_set_key_length (reimplements EVP_CIPHER_CTX_set_key_length) in file ./fips/utl/fips_enc.[o|c]

int **FIPS_cipher_ctx_set_key_length**(EVP_CIPHER_CTX *x, int keylen)
<- x
-> keylen
<- Return

FIPS_cipherinit (reimplements EVP_CipherInit) in file ./fips/utl/fips_enc.[o|c]

__owur int **FIPS_cipherinit**(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, const unsigned char *key, const unsigned char *iv, int enc)
<- ctx
-> cipher
-> key
-> iv
-> enc
<- Return

FIPS_cmac_ctx_cleanup (renames CMAC_CTX_cleanup) in file ./crypto/cmac/cmac.[o|c]

void **FIPS_cmac_ctx_cleanup**(CMAC_CTX *ctx)
<- ctx

FIPS_cmac_ctx_copy (renames CMAC_CTX_copy) in file ./crypto/cmac/cmac.[o|c]

int **FIPS_cmac_ctx_copy**(CMAC_CTX *out, const CMAC_CTX *in)
<- out
-> in
<- Return

FIPS_cmac_ctx_free (renames CMAC_CTX_free) in file ./crypto/cmac/cmac.[o|c]

void **FIPS_cmac_ctx_free**(CMAC_CTX *ctx)
<- ctx

FIPS_cmac_ctx_get0_cipher_ctx (renames CMAC_CTX_get0_cipher_ctx) in file ./crypto/cmac/cmac.[o|c]

EVP_CIPHER_CTX ***FIPS_cmac_ctx_get0_cipher_ctx**(CMAC_CTX *ctx)
<- ctx
<- Return

FIPS_cmac_ctx_new (renames CMAC_CTX_new) in file ./crypto/cmac/cmac.[o|c]

CMAC_CTX ***FIPS_cmac_ctx_new**()
<- Return

FIPS_cmac_final (renames CMAC_Final) in file ./crypto/cmac/cmac.[o|c]

int **FIPS_cmac_final**(CMAC_CTX *ctx, unsigned char *out, size_t *poutlen)
<- ctx
<- out
<- poutlen
<- Return

FIPS_cmac_init (renames CMAC_Init) in file ./crypto/cmac/cmac.[o|c]

int **FIPS_cmac_init**(CMAC_CTX *ctx, const void *key, size_t keylen, const EVP_CIPHER *cipher, ENGINE *impl)
<- ctx
-> key
-> keylen
-> cipher
<-> impl
<- Return

FIPS_cmac_resume (renames CMAC_resume) in file ./crypto/cmac/cmac.[o|c]

int **FIPS_cmac_resume**(CMAC_CTX *ctx)
<- ctx
<- Return

FIPS_cmac_update (renames CMAC_Update) in file ./crypto/cmac/cmac.[o|c]

int **FIPS_cmac_update**(CMAC_CTX *ctx, const void *data, size_t dlen)

<- ctx
-> data
-> dlen
<- Return

FIPS_crypto_get_id_callback (renames CRYPTO_get_id_callback) in file ./crypto/thr_id.[o|c]

unsigned long (***FIPS_crypto_get_id_callback**())(void)
<- Return

FIPS_crypto_set_id_callback (renames CRYPTO_set_id_callback) in file ./crypto/thr_id.[o|c]

void **FIPS_crypto_set_id_callback**(unsigned long (*func)(void))
<-> func

FIPS_crypto_thread_id (renames CRYPTO_thread_id) in file ./crypto/thr_id.[o|c]

unsigned long **FIPS_crypto_thread_id**()
<- Return

FIPS_crypto_threadid_get_callback (renames CRYPTO_THREADID_get_callback) in file ./crypto/thr_id.[o|c]

void (***FIPS_crypto_threadid_get_callback**())(CRYPTO_THREADID *)
<- Return

FIPS_crypto_threadid_hash (renames CRYPTO_THREADID_hash) in file ./crypto/thr_id.[o|c]

unsigned long **FIPS_crypto_threadid_hash**(const CRYPTO_THREADID *id)
-> id
<- Return

FIPS_crypto_threadid_set_callback (renames CRYPTO_THREADID_set_callback) in file ./crypto/thr_id.[o|c]

int **FIPS_crypto_threadid_set_callback**(void (*threadid_func)(CRYPTO_THREADID *))

<-> threadid_func
<- Return

FIPS_crypto_threadid_set_numeric (renames CRYPTO_THREADID_set_numeric) in file ./crypto/thr_id.[o|c]

void **FIPS_crypto_threadid_set_numeric**(CRYPTO_THREADID *id, unsigned long val)
<-> id
-> val

FIPS_crypto_threadid_set_pointer (renames CRYPTO_THREADID_set_pointer) in file ./crypto/thr_id.[o|c]

void **FIPS_crypto_threadid_set_pointer**(CRYPTO_THREADID *id, void *ptr)
<-> id
<-> ptr

FIPS_des_check_key_parity (renames DES_check_key_parity) in file ./crypto/des/set_key.[o|c]

int **FIPS_des_check_key_parity**(const_DES_cblock *key)
-> key
<- Return

FIPS_dh_check (renames DH_check) in file ./crypto/dh/dh_check.[o|c]

int **FIPS_dh_check**(const DH *dh, int *codes)
-> dh
<- codes
<- Return

FIPS_dh_check_pub_key (renames DH_check_pub_key) in file ./crypto/dh/dh_check.[o|c]

int **FIPS_dh_check_pub_key**(const DH *dh, const BIGNUM *pub_key, int *codes)
-> dh
-> pub_key
<- codes
<- Return

FIPS_dh_compute_key (renames DH_compute_key) in file ./crypto/dh/dh_key.[o|c]

int **FIPS_dh_compute_key**(unsigned char *key, const BIGNUM *pub_key, DH *dh)
<- key
-> pub_key
<-> dh
<- Return

FIPS_dh_compute_key_padded (renames DH_compute_key_padded) in file ./crypto/dh/dh_key.[o|c]

int **FIPS_dh_compute_key_padded**(unsigned char *key, const BIGNUM *pub_key, DH *dh)
<- key
-> pub_key
<-> dh
<- Return

FIPS_dh_free in file ./fips/dh/fips_dh_lib.[o|c]

void **FIPS_dh_free**(DH *dh)
<-> dh

FIPS_dh_generate_key (renames DH_generate_key) in file ./crypto/dh/dh_key.[o|c]

int **FIPS_dh_generate_key**(DH *dh)
<-> dh
<- Return

FIPS_dh_generate_parameters_ex (renames DH_generate_parameters_ex) in file ./crypto/dh/dh_gen.[o|c]

int **FIPS_dh_generate_parameters_ex**(DH *dh, int prime_len, int generator, BN_GENCB *cb)
<-> dh
-> prime_len
-> generator
<-> cb
<- Return

FIPS_dh_new in file ./fips/dh/fips_dh_lib.[o|c]

DH * **FIPS_dh_new**()
<- Return

FIPS_dh_openssl (renames DH_OpenSSL) in file ./crypto/dh/dh_key.[o|c]

const DH_METHOD ***FIPS_dh_openssl**()
<- Return

FIPS_digest (reimplements EVP_Digest) in file ./fips/utl/fips_md.[o|c]

__owur int **FIPS_digest**(const void *data, size_t count, unsigned char *md, unsigned int *size, const EVP_MD *type, ENGINE *impl)
-> data
-> count
<- md
<- size
-> type
<-> impl
<- Return

FIPS_digestfinal (reimplements EVP_DigestFinal_ex) in file ./fips/utl/fips_md.[o|c]

__owur int **FIPS_digestfinal**(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *s)
<- ctx
<- md
<- s
<- Return

FIPS_digestinit (reimplements EVP_DigestInit) in file ./fips/utl/fips_md.[o|c]

__owur int **FIPS_digestinit**(EVP_MD_CTX *ctx, const EVP_MD *type)
<- ctx
-> type
<- Return

FIPS_digestupdate (reimplements EVP_DigestUpdate) in file ./fips/utl/fips_md.[o|c]

__owur int **FIPS_digestupdate**(EVP_MD_CTX *ctx, const void *d, size_t cnt)
<- ctx
-> d
-> cnt
<- Return

FIPS_drbg_free in file ./fips/rand/fips_drbg_lib.[o|c]

void **FIPS_drbg_free**(DRBG_CTX *dctx)
<- dctx

FIPS_drbg_generate in file ./fips/rand/fips_drbg_lib.[o|c]

int **FIPS_drbg_generate**(DRBG_CTX *dctx, unsigned char *out, size_t outlen, int prediction_resistance, const unsigned char *adin, size_t adinlen)
<- dctx
<- out
-> outlen
-> prediction_resistance
-> adin
-> adinlen
<- Return

FIPS_drbg_get_app_data in file ./fips/rand/fips_drbg_lib.[o|c]

void ***FIPS_drbg_get_app_data**(DRBG_CTX *ctx)
<- ctx
<- Return

FIPS_drbg_get_blocklength in file ./fips/rand/fips_drbg_lib.[o|c]

size_t **FIPS_drbg_get_blocklength**(DRBG_CTX *dctx)
<- dctx
<- Return

FIPS_drbg_get_strength in file ./fips/rand/fips_drbg_lib.[o|c]

int **FIPS_drbg_get_strength**(DRBG_CTX *dctx)
<- dctx
<- Return

FIPS_drbg_health_check in file ./fips/rand/fips_drbg_selftest.[o|c]

int **FIPS_drbg_health_check**(DRBG_CTX *dctx)
<- dctx
<- Return

FIPS_drbg_init in file ./fips/rand/fips_drbg_lib.[o|c]

int **FIPS_drbg_init**(DRBG_CTX *dctx, int type, unsigned int flags)
<- dctx
-> type
-> flags
<- Return

FIPS_drbg_instantiate in file ./fips/rand/fips_drbg_lib.[o|c]

int **FIPS_drbg_instantiate**(DRBG_CTX *dctx, const unsigned char *pers, size_t perslen)
<- dctx
-> pers
-> perslen
<- Return

FIPS_drbg_method in file ./fips/rand/fips_drbg_rand.[o|c]

const RAND_METHOD ***FIPS_drbg_method**()
<- Return

FIPS_drbg_new in file ./fips/rand/fips_drbg_lib.[o|c]

DRBG_CTX ***FIPS_drbg_new**(int type, unsigned int flags)
-> type

-> flags
<- Return

FIPS_drbg_reseed in file ./fips/rand/fips_drbg_lib.[o|c]

int **FIPS_drbg_reseed**(DRBG_CTX *dctx, const unsigned char *adin, size_t adinlen)
<- dctx
-> adin
-> adinlen
<- Return

FIPS_drbg_set_app_data in file ./fips/rand/fips_drbg_lib.[o|c]

void **FIPS_drbg_set_app_data**(DRBG_CTX *ctx, void *app_data)
<- ctx
<-> app_data

FIPS_drbg_set_callbacks in file ./fips/rand/fips_drbg_lib.[o|c]

int **FIPS_drbg_set_callbacks**(DRBG_CTX *dctx, size_t (*get_entropy)(DRBG_CTX *ctx, unsigned char **pout, int entropy, size_t min_len, size_t max_len), void (*cleanup_entropy)(DRBG_CTX *ctx, unsigned char *out, size_t olen), size_t entropy_blocklen, size_t (*get_nonce)(DRBG_CTX *ctx, unsigned char **pout, int entropy, size_t min_len, size_t max_len), void (*cleanup_nonce)(DRBG_CTX *ctx, unsigned char *out, size_t olen))
<- dctx
<- get_entropy
<- cleanup_entropy
-> entropy_blocklen
<- get_nonce
<- cleanup_nonce
<- Return

FIPS_drbg_set_check_interval in file ./fips/rand/fips_drbg_lib.[o|c]

void **FIPS_drbg_set_check_interval**(DRBG_CTX *dctx, int interval)
<- dctx
-> interval

FIPS_drbg_set_rand_callbacks in file ./fips/rand/fips_drbg_lib.[o|c]

int **FIPS_drbg_set_rand_callbacks**(DRBG_CTX *dctx, size_t (*get_adin)(DRBG_CTX *ctx, unsigned char **pout), void (*cleanup_adin)(DRBG_CTX *ctx, unsigned char *out, size_t olen), int (*rand_seed_cb)(DRBG_CTX *ctx, const void *buf, int num), int (*rand_add_cb)(DRBG_CTX *ctx, const void *buf, int num, double entropy))
<- dctx
<- get_adin
<- cleanup_adin
-> rand_seed_cb
-> rand_add_cb
<- Return

FIPS_drbg_set_reseed_interval in file ./fips/rand/fips_drbg_lib.[o|c]

void **FIPS_drbg_set_reseed_interval**(DRBG_CTX *dctx, int interval)
<- dctx
-> interval

FIPS_drbg_stick in file ./fips/rand/fips_drbg_lib.[o|c]

void **FIPS_drbg_stick**(int onoff)
-> onoff

FIPS_drbg_uninstantiate in file ./fips/rand/fips_drbg_lib.[o|c]

int **FIPS_drbg_uninstantiate**(DRBG_CTX *dctx)
<- dctx
<- Return

FIPS_dsa_free in file ./fips/dsa/fips_dsa_lib.[o|c]

void **FIPS_dsa_free**(DSA *r)
<-> r

FIPS_dsa_generate_key (renames DSA_generate_key) in file ./crypto/dsa/dsa_key.[o|c]

int **FIPS_dsa_generate_key**(DSA *a)

<-> a
<- Return

FIPS_dsa_generate_parameters_ex (renames DSA_generate_parameters_ex) in file ./crypto/dsa/dsa_gen.[o|c]

int **FIPS_dsa_generate_parameters_ex**(DSA *dsa, int bits, const unsigned char *seed, int seed_len, int *counter_ret, unsigned long *h_ret, BN_GENCB *cb)
<-> dsa
-> bits
-> seed
-> seed_len
<- counter_ret
<-> h_ret
<-> cb
<- Return

FIPS_dsa_new in file ./fips/dsa/fips_dsa_lib.[o|c]

DSA * **FIPS_dsa_new**()
<- Return

FIPS_dsa_openssl (renames DSA_OpenSSL) in file ./crypto/dsa/dsa_ossl.[o|c]

const DSA_METHOD ***FIPS_dsa_openssl**()
<- Return

FIPS_dsa_sig_free (reimplements DSA_SIG_free) in file ./fips/dsa/fips_dsa_lib.[o|c]

void **FIPS_dsa_sig_free**(DSA_SIG *a)
<-> a

FIPS_dsa_sig_new (reimplements DSA_SIG_new) in file ./fips/dsa/fips_dsa_lib.[o|c]

DSA_SIG * **FIPS_dsa_sig_new**()
<- Return

FIPS_dsa_sign in file ./fips/dsa/fips_dsa_sign.[o|c]

DSA_SIG * **FIPS_dsa_sign**(DSA *dsa, const unsigned char *msg, size_t msglen, const EVP_MD *mhash)
<-> dsa
-> msg
-> msglen
-> mhash
<- Return

FIPS_dsa_sign_ctx in file ./fips/dsa/fips_dsa_sign.[o|c]

DSA_SIG * **FIPS_dsa_sign_ctx**(DSA *dsa, EVP_MD_CTX *ctx)
<-> dsa
<- ctx
<- Return

FIPS_dsa_sign_digest in file ./fips/dsa/fips_dsa_sign.[o|c]

DSA_SIG * **FIPS_dsa_sign_digest**(DSA *dsa, const unsigned char *dig, int dlen)
<-> dsa
-> dig
-> dlen
<- Return

FIPS_dsa_verify in file ./fips/dsa/fips_dsa_sign.[o|c]

int **FIPS_dsa_verify**(DSA *dsa, const unsigned char *msg, size_t msglen, const EVP_MD *mhash, DSA_SIG *s)
<-> dsa
-> msg
-> msglen
-> mhash
<-> s
<- Return

FIPS_dsa_verify_ctx in file ./fips/dsa/fips_dsa_sign.[o|c]

int **FIPS_dsa_verify_ctx**(DSA *dsa, EVP_MD_CTX *ctx, DSA_SIG *s)

<-> dsa
<- ctx
<-> s
<- Return

FIPS_dsa_verify_digest in file ./fips/dsa/fips_dsa_sign.[o|c]

int **FIPS_dsa_verify_digest**(DSA *dsa, const unsigned char *dig, int dlen, DSA_SIG *s)
<-> dsa
-> dig
-> dlen
<-> s
<- Return

FIPS_ec_get_builtin_curves (renames EC_get_builtin_curves) in file ./crypto/ec/ec_curve.[o|c]

size_t **FIPS_ec_get_builtin_curves**(EC_builtin_curve *r, size_t nitems)
<-> r
-> nitems
<- Return

FIPS_ec_group_clear_free (renames EC_GROUP_clear_free) in file ./crypto/ec/ec_lib.[o|c]

void **FIPS_ec_group_clear_free**(EC_GROUP *group)
<-> group

FIPS_ec_group_get0_generator (renames EC_GROUP_get0_generator) in file ./crypto/ec/ec_lib.[o|c]

const EC_POINT ***FIPS_ec_group_get0_generator**(const EC_GROUP *group)
-> group
<- Return

FIPS_ec_group_get0_seed (renames EC_GROUP_get0_seed) in file ./crypto/ec/ec_lib.[o|c]

unsigned char ***FIPS_ec_group_get0_seed**(const EC_GROUP *x)
-> x
<- Return

FIPS_ec_group_get_asn1_flag (renames EC_GROUP_get_asn1_flag) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_get_asn1_flag**(const EC_GROUP *group)
-> group
<- Return

FIPS_ec_group_get_cofactor (renames EC_GROUP_get_cofactor) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_get_cofactor**(const EC_GROUP *group, BIGNUM *cofactor, BN_CTX *ctx)
-> group
<-> cofactor
<- ctx
<- Return

FIPS_ec_group_get_curve_gf2m (renames EC_GROUP_get_curve_GF2m) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_get_curve_gf2m**(const EC_GROUP *group, BIGNUM *p, BIGNUM *a, BIGNUM *b, BN_CTX *ctx)
-> group
<-> p
<-> a
<-> b
<- ctx
<- Return

FIPS_ec_group_get_curve_gfp (renames EC_GROUP_get_curve_GFp) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_get_curve_gfp**(const EC_GROUP *group, BIGNUM *p, BIGNUM *a, BIGNUM *b, BN_CTX *ctx)
-> group
<-> p
<-> a
<-> b
<- ctx
<- Return

FIPS_ec_group_get_curve_name (renames EC_GROUP_get_curve_name) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_get_curve_name**(const EC_GROUP *group)
-> group
<- Return

FIPS_ec_group_get_degree (renames EC_GROUP_get_degree) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_get_degree**(const EC_GROUP *group)
-> group
<- Return

FIPS_ec_group_get_order (renames EC_GROUP_get_order) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_get_order**(const EC_GROUP *group, BIGNUM *order, BN_CTX *ctx)
-> group
<-> order
<- ctx
<- Return

FIPS_ec_group_method_of (renames EC_GROUP_method_of) in file ./crypto/ec/ec_lib.[o|c]

const EC_METHOD ***FIPS_ec_group_method_of**(const EC_GROUP *group)
-> group
<- Return

FIPS_ec_group_new (renames EC_GROUP_new) in file ./crypto/ec/ec_lib.[o|c]

EC_GROUP ***FIPS_ec_group_new**(const EC_METHOD *meth)
-> meth
<- Return

FIPS_ec_group_new_by_curve_name (renames EC_GROUP_new_by_curve_name) in file ./crypto/ec/ec_curve.[o|c]

EC_GROUP ***FIPS_ec_group_new_by_curve_name**(int nid)

-> nid
<- Return

FIPS_ec_group_new_curve_gf2m (renames EC_GROUP_new_curve_GF2m) in file ./crypto/ec/ec_cvt.[o|c]

EC_GROUP *__FIPS_ec_group_new_curve_gf2m__(const BIGNUM *p, const BIGNUM *a, const BIGNUM *b, BN_CTX *ctx)
-> p
-> a
-> b
<- ctx
<- Return

FIPS_ec_group_new_curve_gfp (renames EC_GROUP_new_curve_GFp) in file ./crypto/ec/ec_cvt.[o|c]

EC_GROUP *__FIPS_ec_group_new_curve_gfp__(const BIGNUM *p, const BIGNUM *a, const BIGNUM *b, BN_CTX *ctx)
-> p
-> a
-> b
<- ctx
<- Return

FIPS_ec_group_precompute_mult (renames EC_GROUP_precompute_mult) in file ./crypto/ec/ec_lib.[o|c]

int __FIPS_ec_group_precompute_mult__(EC_GROUP *group, BN_CTX *ctx)
<-> group
<- ctx
<- Return

FIPS_ec_group_set_asn1_flag (renames EC_GROUP_set_asn1_flag) in file ./crypto/ec/ec_lib.[o|c]

void __FIPS_ec_group_set_asn1_flag__(EC_GROUP *group, int flag)
<-> group
-> flag

FIPS_ec_group_set_curve_gf2m (renames EC_GROUP_set_curve_GF2m) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_set_curve_gf2m**(EC_GROUP *group, const BIGNUM *p, const BIGNUM *a, const BIGNUM *b, BN_CTX *ctx)
<-> group
-> p
-> a
-> b
<- ctx
<- Return

FIPS_ec_group_set_curve_gfp (renames EC_GROUP_set_curve_GFp) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_set_curve_gfp**(EC_GROUP *group, const BIGNUM *p, const BIGNUM *a, const BIGNUM *b, BN_CTX *ctx)
<-> group
-> p
-> a
-> b
<- ctx
<- Return

FIPS_ec_group_set_curve_name (renames EC_GROUP_set_curve_name) in file ./crypto/ec/ec_lib.[o|c]

void **FIPS_ec_group_set_curve_name**(EC_GROUP *group, int nid)
<-> group
-> nid

FIPS_ec_group_set_generator (renames EC_GROUP_set_generator) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_group_set_generator**(EC_GROUP *group, const EC_POINT *generator, const BIGNUM *order, const BIGNUM *cofactor)
<-> group
-> generator
-> order
-> cofactor
<- Return

FIPS_ec_group_set_point_conversion_form (renames EC_GROUP_set_point_conversion_form) in file ./crypto/ec/ec_lib.[o|c]

void **FIPS_ec_group_set_point_conversion_form**(EC_GROUP *group, point_conversion_form_t form)
<-> group
-> form

FIPS_ec_key_check_key (renames EC_KEY_check_key) in file ./crypto/ec/ec_key.[o|c]

int **FIPS_ec_key_check_key**(const EC_KEY *key)
-> key
<- Return

FIPS_ec_key_clear_flags (renames EC_KEY_clear_flags) in file ./crypto/ec/ec_key.[o|c]

void **FIPS_ec_key_clear_flags**(EC_KEY *key, int flags)
<-> key
-> flags

FIPS_ec_key_copy (renames EC_KEY_copy) in file ./crypto/ec/ec_key.[o|c]

EC_KEY ***FIPS_ec_key_copy**(EC_KEY *dst, const EC_KEY *src)
<-> dst
-> src
<- Return

FIPS_ec_key_dup (renames EC_KEY_dup) in file ./crypto/ec/ec_key.[o|c]

EC_KEY ***FIPS_ec_key_dup**(const EC_KEY *src)
-> src
<- Return

FIPS_ec_key_free (renames EC_KEY_free) in file ./crypto/ec/ec_key.[o|c]

void **FIPS_ec_key_free**(EC_KEY *key)
<-> key

FIPS_ec_key_generate_key (renames EC_KEY_generate_key) in file ./crypto/ec/ec_key.[o|c]

int **FIPS_ec_key_generate_key**(EC_KEY *key)
<-> key
<- Return

FIPS_ec_key_get0_group (renames EC_KEY_get0_group) in file ./crypto/ec/ec_key.[o|c]

const EC_GROUP ***FIPS_ec_key_get0_group**(const EC_KEY *key)
-> key
<- Return

FIPS_ec_key_get0_private_key (renames EC_KEY_get0_private_key) in file ./crypto/ec/ec_key.[o|c]

const BIGNUM ***FIPS_ec_key_get0_private_key**(const EC_KEY *key)
-> key
<- Return

FIPS_ec_key_get0_public_key (renames EC_KEY_get0_public_key) in file ./crypto/ec/ec_key.[o|c]

const EC_POINT ***FIPS_ec_key_get0_public_key**(const EC_KEY *key)
-> key
<- Return

FIPS_ec_key_get_conv_form (renames EC_KEY_get_conv_form) in file ./crypto/ec/ec_key.[o|c]

point_conversion_form_t **FIPS_ec_key_get_conv_form**(const EC_KEY *key)
-> key
<- Return

FIPS_ec_key_get_enc_flags (renames EC_KEY_get_enc_flags) in file ./crypto/ec/ec_key.[o|c]

unsigned **FIPS_ec_key_get_enc_flags**(const EC_KEY *key)
-> key
<- Return

FIPS_ec_key_get_flags (renames EC_KEY_get_flags) in file ./crypto/ec/ec_key.[o|c]

int **FIPS_ec_key_get_flags**(const EC_KEY *key)
-> key
<- Return

FIPS_ec_key_get_key_method_data (renames EC_KEY_get_key_method_data) in file ./crypto/ec/ec_key.[o|c]

void ***FIPS_ec_key_get_key_method_data**(EC_KEY *key, void *(*dup_func)(void *), void (*free_func)(void *), void (*clear_free_func)(void *))
<-> key
<-> dup_func
<-> free_func
<-> clear_free_func
<- Return

FIPS_ec_key_insert_key_method_data (renames EC_KEY_insert_key_method_data) in file ./crypto/ec/ec_key.[o|c]

void **FIPS_ec_key_insert_key_method_data**(EC_KEY *key, void *data, void *(*dup_func)(void *), void (*free_func)(void *), void (*clear_free_func)(void *))
<-> key
<-> data
<-> dup_func
<-> free_func
<-> clear_free_func

FIPS_ec_key_new (renames EC_KEY_new) in file ./crypto/ec/ec_key.[o|c]

EC_KEY ***FIPS_ec_key_new**()
<- Return

FIPS_ec_key_new_by_curve_name (renames EC_KEY_new_by_curve_name) in file ./crypto/ec/ec_key.[o|c]

EC_KEY ***FIPS_ec_key_new_by_curve_name**(int nid)
-> nid
<- Return

FIPS_ec_key_precompute_mult (renames EC_KEY_precompute_mult) in file ./crypto/ec/ec_key.[o|c]

int **FIPS_ec_key_precompute_mult**(EC_KEY *key, BN_CTX *ctx)
<-> key
<- ctx
<- Return

FIPS_ec_key_set_asn1_flag (renames EC_KEY_set_asn1_flag) in file ./crypto/ec/ec_key.[o|c]

void **FIPS_ec_key_set_asn1_flag**(EC_KEY *eckey, int asn1_flag)
<-> eckey
-> asn1_flag

FIPS_ec_key_set_conv_form (renames EC_KEY_set_conv_form) in file ./crypto/ec/ec_key.[o|c]

void **FIPS_ec_key_set_conv_form**(EC_KEY *eckey, point_conversion_form_t cform)
<-> eckey
-> cform

FIPS_ec_key_set_enc_flags (renames EC_KEY_set_enc_flags) in file ./crypto/ec/ec_key.[o|c]

void **FIPS_ec_key_set_enc_flags**(EC_KEY *eckey, unsigned int flags)
<-> eckey
-> flags

FIPS_ec_key_set_flags (renames EC_KEY_set_flags) in file ./crypto/ec/ec_key.[o|c]

void **FIPS_ec_key_set_flags**(EC_KEY *key, int flags)
<-> key
-> flags

FIPS_ec_key_set_group (renames EC_KEY_set_group) in file ./crypto/ec/ec_key.[o|c]

int **FIPS_ec_key_set_group**(EC_KEY *key, const EC_GROUP *group)
<-> key
-> group
<- Return

FIPS_ec_key_set_private_key (renames EC_KEY_set_private_key) in file ./crypto/ec/ec_key.[o|c]

int **FIPS_ec_key_set_private_key**(EC_KEY *key, const BIGNUM *prv)
<-> key
-> prv
<- Return

FIPS_ec_key_set_public_key (renames EC_KEY_set_public_key) in file ./crypto/ec/ec_key.[o|c]

int **FIPS_ec_key_set_public_key**(EC_KEY *key, const EC_POINT *pub)
<-> key
-> pub
<- Return

FIPS_ec_key_set_public_key_affine_coordinates (renames EC_KEY_set_public_key_affine_coordinates) in file ./crypto/ec/ec_key.[o|c]

int **FIPS_ec_key_set_public_key_affine_coordinates**(EC_KEY *key, BIGNUM *x, BIGNUM *y)
<-> key
<-> x
<-> y
<- Return

FIPS_ec_key_up_ref (renames EC_KEY_up_ref) in file ./crypto/ec/ec_key.[o|c]

int **FIPS_ec_key_up_ref**(EC_KEY *key)
<-> key
<- Return

FIPS_ec_method_get_field_type (renames EC_METHOD_get_field_type) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_method_get_field_type**(const EC_METHOD *meth)
-> meth
<- Return

FIPS_ec_point_clear_free (renames EC_POINT_clear_free) in file ./crypto/ec/ec_lib.[o|c]

void **FIPS_ec_point_clear_free**(EC_POINT *point)
<-> point

FIPS_ec_point_free (renames EC_POINT_free) in file ./crypto/ec/ec_lib.[o|c]

void **FIPS_ec_point_free**(EC_POINT *point)
<-> point

FIPS_ec_point_get_affine_coordinates_gf2m (renames EC_POINT_get_affine_coordinates_GF2m) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_point_get_affine_coordinates_gf2m**(const EC_GROUP *group, const EC_POINT *p, BIGNUM *x, BIGNUM *y, BN_CTX *ctx)
-> group
-> p
<-> x
<-> y
<- ctx
<- Return

FIPS_ec_point_get_affine_coordinates_gfp (renames EC_POINT_get_affine_coordinates_GFp) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_point_get_affine_coordinates_gfp**(const EC_GROUP *group, const EC_POINT *p, BIGNUM *x, BIGNUM *y, BN_CTX *ctx)
-> group
-> p
<-> x
<-> y

<- ctx
<- Return

FIPS_ec_point_get_jprojective_coordinates_gfp (renames EC_POINT_get_Jprojective_coordinates_GFp) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_point_get_jprojective_coordinates_gfp**(const EC_GROUP *group, const EC_POINT *p, BIGNUM *x, BIGNUM *y, BIGNUM *z, BN_CTX *ctx)
-> group
-> p
<-> x
<-> y
<-> z
<- ctx
<- Return

FIPS_ec_point_is_at_infinity (renames EC_POINT_is_at_infinity) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_point_is_at_infinity**(const EC_GROUP *group, const EC_POINT *p)
-> group
-> p
<- Return

FIPS_ec_point_is_on_curve (renames EC_POINT_is_on_curve) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_point_is_on_curve**(const EC_GROUP *group, const EC_POINT *point, BN_CTX *ctx)
-> group
-> point
<- ctx
<- Return

FIPS_ec_point_make_affine (renames EC_POINT_make_affine) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_point_make_affine**(const EC_GROUP *group, EC_POINT *point, BN_CTX *ctx)
-> group
<-> point
<- ctx
<- Return

FIPS_ec_point_method_of (renames EC_POINT_method_of) in file ./crypto/ec/ec_lib.[o|c]

const EC_METHOD ***FIPS_ec_point_method_of**(const EC_POINT *point)
-> point
<- Return

FIPS_ec_point_mul (renames EC_POINT_mul) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_point_mul**(const EC_GROUP *group, EC_POINT *r, const BIGNUM *n, const EC_POINT *q, const BIGNUM *m, BN_CTX *ctx)
-> group
<-> r
-> n
-> q
-> m
<- ctx
<- Return

FIPS_ec_point_new (renames EC_POINT_new) in file ./crypto/ec/ec_lib.[o|c]

EC_POINT ***FIPS_ec_point_new**(const EC_GROUP *group)
-> group
<- Return

FIPS_ec_point_set_to_infinity (renames EC_POINT_set_to_infinity) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_point_set_to_infinity**(const EC_GROUP *group, EC_POINT *point)
-> group
<-> point
<- Return

FIPS_ec_points_make_affine (renames EC_POINTs_make_affine) in file ./crypto/ec/ec_lib.[o|c]

int **FIPS_ec_points_make_affine**(const EC_GROUP *group, size_t num, EC_POINT *points, BN_CTX *ctx)
-> group
-> num

<-> points
<- ctx
<- Return

FIPS_ecdh_compute_key (renames ECDH_compute_key) in file ./crypto/ecdh/ech_key.[o|c]

int **FIPS_ecdh_compute_key**(void *out, size_t outlen, const EC_POINT *pub_key, EC_KEY *ecdh, void *(*KDF)(const void *in, size_t inlen, void *out, size_t *outlen))
<-> out
-> outlen
-> pub_key
<-> ecdh
-> KDF
<- Return

FIPS_ecdh_openssl (renames ECDH_OpenSSL) in file ./crypto/ecdh/ech_ossl.[o|c]

const ECDH_METHOD ***FIPS_ecdh_openssl**()
<- Return

FIPS_ecdsa_openssl (renames ECDSA_OpenSSL) in file ./crypto/ecdsa/ecs_ossl.[o|c]

const ECDSA_METHOD ***FIPS_ecdsa_openssl**()
<- Return

FIPS_ecdsa_sig_free (reimplements ECDSA_SIG_free) in file ./fips/ecdsa/fips_ecdsa_lib.[o|c]

void **FIPS_ecdsa_sig_free**(ECDSA_SIG *sig)
<-> sig

FIPS_ecdsa_sig_new (reimplements ECDSA_SIG_new) in file ./fips/ecdsa/fips_ecdsa_lib.[o|c]

ECDSA_SIG ***FIPS_ecdsa_sig_new**()
<- Return

FIPS_ecdsa_sign in file ./fips/ecdsa/fips_ecdsa_sign.[o|c]

ECDSA_SIG * **FIPS_ecdsa_sign**(EC_KEY *key, const unsigned char *msg, size_t msglen, const EVP_MD *mhash)
<-> key
-> msg
-> msglen
-> mhash
<- Return

FIPS_ecdsa_sign_ctx in file ./fips/ecdsa/fips_ecdsa_sign.[o|c]

ECDSA_SIG * **FIPS_ecdsa_sign_ctx**(EC_KEY *key, EVP_MD_CTX *ctx)
<-> key
<- ctx
<- Return

FIPS_ecdsa_sign_digest in file ./crypto/ecdsa/ecs_ossl.[o|c]

ECDSA_SIG * **FIPS_ecdsa_sign_digest**(EC_KEY *key, const unsigned char *dig, int dlen)
<-> key
-> dig
-> dlen
<- Return

FIPS_ecdsa_verify in file ./fips/ecdsa/fips_ecdsa_sign.[o|c]

int **FIPS_ecdsa_verify**(EC_KEY *key, const unsigned char *msg, size_t msglen, const EVP_MD *mhash, ECDSA_SIG *s)
<-> key
-> msg
-> msglen
-> mhash
<-> s
<- Return

FIPS_ecdsa_verify_ctx in file ./fips/ecdsa/fips_ecdsa_sign.[o|c]

int **FIPS_ecdsa_verify_ctx**(EC_KEY *key, EVP_MD_CTX *ctx, ECDSA_SIG *s)
<-> key
<- ctx

<-> s
<- Return

FIPS_ecdsa_verify_digest in file ./crypto/ecdsa/ecs_ossl.[o|c]

int **FIPS_ecdsa_verify_digest**(EC_KEY *key, const unsigned char *dig, int dlen, ECDSA_SIG *s)
<-> key
-> dig
-> dlen
<-> s
<- Return

FIPS_evp_aes_128_cbc (renames EVP_aes_128_cbc) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_128_cbc**()
<- Return

FIPS_evp_aes_128_ccm (renames EVP_aes_128_ccm) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_128_ccm**()
<- Return

FIPS_evp_aes_128_cfb1 (renames EVP_aes_128_cfb1) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_128_cfb1**()
<- Return

FIPS_evp_aes_128_cfb128 (renames EVP_aes_128_cfb128) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_128_cfb128**()
<- Return

FIPS_evp_aes_128_cfb8 (renames EVP_aes_128_cfb8) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_128_cfb8**()
<- Return

FIPS_evp_aes_128_ctr (renames EVP_aes_128_ctr) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_128_ctr__()
<- Return

FIPS_evp_aes_128_ecb (renames EVP_aes_128_ecb) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_128_ecb__()
<- Return

FIPS_evp_aes_128_gcm (renames EVP_aes_128_gcm) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_128_gcm__()
<- Return

FIPS_evp_aes_128_ofb (renames EVP_aes_128_ofb) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_128_ofb__()
<- Return

FIPS_evp_aes_128_xts (renames EVP_aes_128_xts) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_128_xts__()
<- Return

FIPS_evp_aes_192_cbc (renames EVP_aes_192_cbc) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_192_cbc__()
<- Return

FIPS_evp_aes_192_ccm (renames EVP_aes_192_ccm) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_192_ccm__()
<- Return

FIPS_evp_aes_192_cfb1 (renames EVP_aes_192_cfb1) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_192_cfb1**()
<- Return

FIPS_evp_aes_192_cfb128 (renames EVP_aes_192_cfb128) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_192_cfb128**()
<- Return

FIPS_evp_aes_192_cfb8 (renames EVP_aes_192_cfb8) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_192_cfb8**()
<- Return

FIPS_evp_aes_192_ctr (renames EVP_aes_192_ctr) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_192_ctr**()
<- Return

FIPS_evp_aes_192_ecb (renames EVP_aes_192_ecb) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_192_ecb**()
<- Return

FIPS_evp_aes_192_gcm (renames EVP_aes_192_gcm) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_192_gcm**()
<- Return

FIPS_evp_aes_192_ofb (renames EVP_aes_192_ofb) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_192_ofb**()
<- Return

FIPS_evp_aes_256_cbc (renames EVP_aes_256_cbc) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_256_cbc**()
<- Return

FIPS_evp_aes_256_ccm (renames EVP_aes_256_ccm) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_256_ccm**()
<- Return

FIPS_evp_aes_256_cfb1 (renames EVP_aes_256_cfb1) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_256_cfb1**()
<- Return

FIPS_evp_aes_256_cfb128 (renames EVP_aes_256_cfb128) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_256_cfb128**()
<- Return

FIPS_evp_aes_256_cfb8 (renames EVP_aes_256_cfb8) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_256_cfb8**()
<- Return

FIPS_evp_aes_256_ctr (renames EVP_aes_256_ctr) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_256_ctr**()
<- Return

FIPS_evp_aes_256_ecb (renames EVP_aes_256_ecb) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER ***FIPS_evp_aes_256_ecb**()
<- Return

FIPS_evp_aes_256_gcm (renames EVP_aes_256_gcm) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_256_gcm__()
<- Return

FIPS_evp_aes_256_ofb (renames EVP_aes_256_ofb) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_256_ofb__()
<- Return

FIPS_evp_aes_256_xts (renames EVP_aes_256_xts) in file ./crypto/evp/e_aes.[o|c]

const EVP_CIPHER *__FIPS_evp_aes_256_xts__()
<- Return

FIPS_evp_des_ede (renames EVP_des_ede) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER *__FIPS_evp_des_ede__()
<- Return

FIPS_evp_des_ede3 (renames EVP_des_ede3) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER *__FIPS_evp_des_ede3__()
<- Return

FIPS_evp_des_ede3_cbc (renames EVP_des_ede3_cbc) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER *__FIPS_evp_des_ede3_cbc__()
<- Return

FIPS_evp_des_ede3_cfb1 (renames EVP_des_ede3_cfb1) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER *__FIPS_evp_des_ede3_cfb1__()
<- Return

FIPS_evp_des_ede3_cfb64 (renames EVP_des_ede3_cfb64) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER ***FIPS_evp_des_ede3_cfb64**()
<- Return

FIPS_evp_des_ede3_cfb8 (renames EVP_des_ede3_cfb8) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER ***FIPS_evp_des_ede3_cfb8**()
<- Return

FIPS_evp_des_ede3_ecb (renames EVP_des_ede3_ecb) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER ***FIPS_evp_des_ede3_ecb**()
<- Return

FIPS_evp_des_ede3_ofb (renames EVP_des_ede3_ofb) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER ***FIPS_evp_des_ede3_ofb**()
<- Return

FIPS_evp_des_ede_cbc (renames EVP_des_ede_cbc) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER ***FIPS_evp_des_ede_cbc**()
<- Return

FIPS_evp_des_ede_cfb64 (renames EVP_des_ede_cfb64) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER ***FIPS_evp_des_ede_cfb64**()
<- Return

FIPS_evp_des_ede_ecb (renames EVP_des_ede_ecb) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER ***FIPS_evp_des_ede_ecb**()
<- Return

FIPS_evp_des_ede_ofb (renames EVP_des_ede_ofb) in file ./crypto/evp/e_des3.[o|c]

const EVP_CIPHER ***FIPS_evp_des_ede_ofb**()
<- Return

FIPS_evp_dss (renames EVP_dss) in file ./crypto/evp/m_dss.[o|c]

const EVP_MD ***FIPS_evp_dss**()
<- Return

FIPS_evp_dss1 (renames EVP_dss1) in file ./crypto/evp/m_dss1.[o|c]

const EVP_MD ***FIPS_evp_dss1**()
<- Return

FIPS_evp_ecdsa (renames EVP_ecdsa) in file ./crypto/evp/m_ecdsa.[o|c]

const EVP_MD ***FIPS_evp_ecdsa**()
<- Return

FIPS_evp_enc_null (renames EVP_enc_null) in file ./crypto/evp/e_null.[o|c]

const EVP_CIPHER ***FIPS_evp_enc_null**()
<- Return

FIPS_evp_sha1 (renames EVP_sha1) in file ./crypto/evp/m_sha1.[o|c]

const EVP_MD ***FIPS_evp_sha1**()
<- Return

FIPS_evp_sha224 (renames EVP_sha224) in file ./crypto/evp/m_sha1.[o|c]

const EVP_MD ***FIPS_evp_sha224**()
<- Return

FIPS_evp_sha256 (renames EVP_sha256) in file ./crypto/evp/m_sha1.[o|c]

const EVP_MD *__FIPS_evp_sha256__()
<- Return

FIPS_evp_sha384 (renames EVP_sha384) in file ./crypto/evp/m_sha1.[o|c]

const EVP_MD *__FIPS_evp_sha384__()
<- Return

FIPS_evp_sha512 (renames EVP_sha512) in file ./crypto/evp/m_sha1.[o|c]

const EVP_MD *__FIPS_evp_sha512__()
<- Return

FIPS_free (reimplements CRYPTO_free) in file ./fips/utl/fips_mem.[o|c]

void __FIPS_free__(void *ptr)
<-> ptr

FIPS_get_cipherbynid in file ./fips/utl/fips_enc.[o|c]

const struct evp_cipher_st *__FIPS_get_cipherbynid__(int nid)
-> nid
<- Return

FIPS_get_default_drbg in file ./fips/rand/fips_drbg_rand.[o|c]

DRBG_CTX *__FIPS_get_default_drbg__()
<- Return

FIPS_get_digestbynid in file ./fips/utl/fips_md.[o|c]

const struct env_md_st *__FIPS_get_digestbynid__(int nid)
-> nid
<- Return

FIPS_get_timevec in file ./fips/rand/fips_rand.[o|c]

void **FIPS_get_timevec**(unsigned char *buf, unsigned long *pctr)
<- buf
<-> pctr

FIPS_hmac (renames HMAC) in file ./crypto/hmac/hmac.[o|c]

unsigned char ***FIPS_hmac**(const EVP_MD *evp_md, const void *key, int key_len, const unsigned char *d, size_t n, unsigned char *md, unsigned int *md_len)
-> evp_md
-> key
-> key_len
-> d
-> n
<- md
<- md_len
<- Return

FIPS_hmac_ctx_cleanup (renames HMAC_CTX_cleanup) in file ./crypto/hmac/hmac.[o|c]

void **FIPS_hmac_ctx_cleanup**(HMAC_CTX *ctx)
<- ctx

FIPS_hmac_ctx_copy (renames HMAC_CTX_copy) in file ./crypto/hmac/hmac.[o|c]

__owur int **FIPS_hmac_ctx_copy**(HMAC_CTX *dctx, HMAC_CTX *sctx)
<- dctx
<- sctx
<- Return

FIPS_hmac_ctx_init (renames HMAC_CTX_init) in file ./crypto/hmac/hmac.[o|c]

void **FIPS_hmac_ctx_init**(HMAC_CTX *ctx)
<- ctx

FIPS_hmac_ctx_set_flags (renames HMAC_CTX_set_flags) in file ./crypto/hmac/hmac.[o|c]

void **FIPS_hmac_ctx_set_flags**(HMAC_CTX *ctx, unsigned long flags)
<- ctx
-> flags

FIPS_hmac_final (renames HMAC_Final) in file ./crypto/hmac/hmac.[o|c]

__owur int **FIPS_hmac_final**(HMAC_CTX *ctx, unsigned char *md, unsigned int *len)
<- ctx
<- md
<- len
<- Return

FIPS_hmac_init (renames HMAC_Init) in file ./crypto/hmac/hmac.[o|c]

__owur int **FIPS_hmac_init**(HMAC_CTX *ctx, const void *key, int len, const EVP_MD *md)
<- ctx
-> key
-> len
-> md
<- Return

FIPS_hmac_init_ex (renames HMAC_Init_ex) in file ./crypto/hmac/hmac.[o|c]

__owur int **FIPS_hmac_init_ex**(HMAC_CTX *ctx, const void *key, int len, const EVP_MD *md, ENGINE *impl)
<- ctx
-> key
-> len
-> md
<-> impl
<- Return

FIPS_hmac_update (renames HMAC_Update) in file ./crypto/hmac/hmac.[o|c]

__owur int **FIPS_hmac_update**(HMAC_CTX *ctx, const unsigned char *data, size_t len)
<- ctx
-> data
-> len
<- Return

FIPS_incore_fingerprint in file ./fips/fips.[o|c]

unsigned int **FIPS_incore_fingerprint**(unsigned char *sig, unsigned int len)
<- sig
-> len
<- Return

FIPS_lock (reimplements CRYPTO_lock) in file ./fips/utl/fips_lck.[o|c]

void **FIPS_lock**(int mode, int type, const char *file, int line)
-> mode
-> type
-> file
-> line

FIPS_malloc (reimplements CRYPTO_malloc) in file ./fips/utl/fips_mem.[o|c]

void ***FIPS_malloc**(int num, const char *file, int line)
-> num
-> file
-> line
<- Return

FIPS_md_ctx_cleanup (reimplements EVP_MD_CTX_cleanup) in file ./fips/utl/fips_md.[o|c]

int **FIPS_md_ctx_cleanup**(EVP_MD_CTX *ctx)
<- ctx
<- Return

FIPS_md_ctx_copy (reimplements EVP_MD_CTX_copy_ex) in file ./fips/utl/fips_md.[o|c]

__owur int **FIPS_md_ctx_copy**(EVP_MD_CTX *out, const EVP_MD_CTX *in)
<- out
-> in
<- Return

FIPS_md_ctx_create (reimplements EVP_MD_CTX_create) in file ./fips/utl/fips_md.[o|c]

EVP_MD_CTX ***FIPS_md_ctx_create**()
<- Return

FIPS_md_ctx_destroy (reimplements EVP_MD_CTX_destroy) in file ./fips/utl/fips_md.[o|c]

void **FIPS_md_ctx_destroy**(EVP_MD_CTX *ctx)
<- ctx

FIPS_md_ctx_init (reimplements EVP_MD_CTX_init) in file ./fips/utl/fips_md.[o|c]

void **FIPS_md_ctx_init**(EVP_MD_CTX *ctx)
<- ctx

FIPS_module_mode in file ./fips/fips.[o|c]

int **FIPS_module_mode**()
<- Return

FIPS_module_mode_set in file ./fips/fips.[o|c]

int **FIPS_module_mode_set**(int onoff, const char *auth)
-> onoff
-> auth
<- Return

FIPS_module_version in file ./fips/fips.[o|c]

unsigned long **FIPS_module_version**()
<- Return

FIPS_module_version_text in file ./fips/fips.[o|c]

const char ***FIPS_module_version_text**()
<- Return

FIPS_openssl_cleanse (renames OPENSSL_cleanse) in file ./crypto/x86cpuid.[o|c]

void **FIPS_openssl_cleanse**(void *ptr, size_t len)
<-> ptr
-> len

FIPS_openssl_showfatal (renames OPENSSL_showfatal) in file ./crypto/cryptlib.[o|c]

void **FIPS_openssl_showfatal**(const char *fmta, ...)
-> fmta
-> ...

FIPS_openssldie (renames OpenSSLDie) in file ./crypto/cryptlib.[o|c]

void **FIPS_openssldie**(const char *file, int line, const char *assertion)
-> file
-> line
-> assertion

FIPS_post_set_callback in file ./fips/fips_post.[o|c]

void **FIPS_post_set_callback**(int (*post_cb)(int op, int id, int subid, void *ex))
<- post_cb

FIPS_put_error (reimplements ERR_put_error) in file ./fips/utl/fips_err.[o|c]

void **FIPS_put_error**(int lib, int func, int reason, const char *file, int line)
-> lib
-> func
-> reason
-> file
-> line

FIPS_rand_add (reimplements RAND_add) in file ./fips/rand/fips_rand_lib.[o|c]

void **FIPS_rand_add**(const void *buf, int num, double entropy)
-> buf

-> num
-> entropy

FIPS_rand_bytes (reimplements RAND_bytes) in file ./fips/rand/fips_rand_lib.[o|c]

int **FIPS_rand_bytes**(unsigned char *buf, int num)
<- buf
-> num
<- Return

FIPS_rand_get_method in file ./fips/rand/fips_rand_lib.[o|c]

const RAND_METHOD ***FIPS_rand_get_method**()
<- Return

FIPS_rand_pseudo_bytes (reimplements RAND_pseudo_bytes) in file ./fips/rand/fips_rand_lib.[o|c]

int **FIPS_rand_pseudo_bytes**(unsigned char *buf, int num)
<- buf
-> num
<- Return

FIPS_rand_seed (reimplements RAND_seed) in file ./fips/rand/fips_rand_lib.[o|c]

void **FIPS_rand_seed**(const void *buf, int num)
-> buf
-> num

FIPS_rand_set_bits in file ./fips/rand/fips_rand_lib.[o|c]

void **FIPS_rand_set_bits**(int nbits)
-> nbits

FIPS_rand_set_method in file ./fips/rand/fips_rand_lib.[o|c]

int **FIPS_rand_set_method**(const RAND_METHOD *meth)

-> meth
<- Return

FIPS_rand_status (reimplements RAND_status) in file ./fips/rand/fips_rand_lib.[o|c]

int **FIPS_rand_status**()
<- Return

FIPS_rand_strength in file ./fips/rand/fips_rand_lib.[o|c]

int **FIPS_rand_strength**()
<- Return

FIPS_rsa_blinding_off (renames RSA_blinding_off) in file ./crypto/rsa/rsa_crpt.[o|c]

void **FIPS_rsa_blinding_off**(RSA *rsa)
<-> rsa

FIPS_rsa_blinding_on (renames RSA_blinding_on) in file ./crypto/rsa/rsa_crpt.[o|c]

int **FIPS_rsa_blinding_on**(RSA *rsa, BN_CTX *ctx)
<-> rsa
<- ctx
<- Return

FIPS_rsa_flags (renames RSA_flags) in file ./crypto/rsa/rsa_crpt.[o|c]

int **FIPS_rsa_flags**(const RSA *r)
-> r
<- Return

FIPS_rsa_free in file ./fips/rsa/fips_rsa_lib.[o|c]

void **FIPS_rsa_free**(struct rsa_st *r)
<-> r

FIPS_rsa_generate_key_ex (renames RSA_generate_key_ex) in file ./crypto/rsa/rsa_gen.[o|c]

int **FIPS_rsa_generate_key_ex**(RSA *rsa, int bits, BIGNUM *e, BN_GENCB *cb)
<-> rsa
-> bits
<-> e
<-> cb
<- Return

FIPS_rsa_new in file ./fips/rsa/fips_rsa_lib.[o|c]

struct rsa_st ***FIPS_rsa_new**()
<- Return

FIPS_rsa_pkcs1_ssleay (renames RSA_PKCS1_SSLeay) in file ./crypto/rsa/rsa_eay.[o|c]

const RSA_METHOD ***FIPS_rsa_pkcs1_ssleay**()
<- Return

FIPS_rsa_private_decrypt (renames RSA_private_decrypt) in file ./crypto/rsa/rsa_crpt.[o|c]

int **FIPS_rsa_private_decrypt**(int flen, const unsigned char *from, unsigned char *to, RSA *rsa, int padding)
-> flen
-> from
<- to
<-> rsa
-> padding
<- Return

FIPS_rsa_private_encrypt (renames RSA_private_encrypt) in file ./crypto/rsa/rsa_crpt.[o|c]

int **FIPS_rsa_private_encrypt**(int flen, const unsigned char *from, unsigned char *to, RSA *rsa, int padding)
-> flen
-> from
<- to
<-> rsa

-> padding
<- Return

FIPS_rsa_public_decrypt (renames RSA_public_decrypt) in file ./crypto/rsa/rsa_crpt.[o|c]

int **FIPS_rsa_public_decrypt**(int flen, const unsigned char *from, unsigned char *to, RSA *rsa, int padding)
-> flen
-> from
<- to
<-> rsa
-> padding
<- Return

FIPS_rsa_public_encrypt (renames RSA_public_encrypt) in file ./crypto/rsa/rsa_crpt.[o|c]

int **FIPS_rsa_public_encrypt**(int flen, const unsigned char *from, unsigned char *to, RSA *rsa, int padding)
-> flen
-> from
<- to
<-> rsa
-> padding
<- Return

FIPS_rsa_sign in file ./fips/rsa/fips_rsa_sign.[o|c]

int **FIPS_rsa_sign**(struct rsa_st *rsa, const unsigned char *msg, int msglen, const struct env_md_st *mhash, int rsa_pad_mode, int saltlen, const struct env_md_st *mgf1Hash, unsigned char *sigret, unsigned int *siglen)
<-> rsa
-> msg
-> msglen
-> mhash
-> rsa_pad_mode
-> saltlen
-> mgf1Hash
<- sigret
<- siglen
<- Return

FIPS_rsa_sign_ctx in file ./fips/rsa/fips_rsa_sign.[o|c]

int **FIPS_rsa_sign_ctx**(struct rsa_st *rsa, struct env_md_ctx_st *ctx, int rsa_pad_mode, int saltlen, const struct env_md_st *mgf1Hash, unsigned char *sigret, unsigned int *siglen)
<-> rsa
<- ctx
-> rsa_pad_mode
-> saltlen
-> mgf1Hash
<- sigret
<- siglen
<- Return

FIPS_rsa_sign_digest in file ./fips/rsa/fips_rsa_sign.[o|c]

int **FIPS_rsa_sign_digest**(struct rsa_st *rsa, const unsigned char *md, int md_len, const struct env_md_st *mhash, int rsa_pad_mode, int saltlen, const struct env_md_st *mgf1Hash, unsigned char *sigret, unsigned int *siglen)
<-> rsa
-> md
-> md_len
-> mhash
-> rsa_pad_mode
-> saltlen
-> mgf1Hash
<- sigret
<- siglen
<- Return

FIPS_rsa_size (renames RSA_size) in file ./crypto/rsa/rsa_crpt.[o|c]

int **FIPS_rsa_size**(const RSA *rsa)
-> rsa
<- Return

FIPS_rsa_verify in file ./fips/rsa/fips_rsa_sign.[o|c]

int **FIPS_rsa_verify**(struct rsa_st *rsa, const unsigned char *msg, int msglen, const struct env_md_st *mhash, int rsa_pad_mode, int saltlen, const struct env_md_st *mgf1Hash, const

unsigned char *sigbuf, unsigned int siglen)
<-> rsa
-> msg
-> msglen
-> mhash
-> rsa_pad_mode
-> saltlen
-> mgf1Hash
-> sigbuf
-> siglen
<- Return

FIPS_rsa_verify_ctx in file ./fips/rsa/fips_rsa_sign.[o|c]

int **FIPS_rsa_verify_ctx**(struct rsa_st *rsa, struct env_md_ctx_st *ctx, int rsa_pad_mode, int saltlen, const struct env_md_st *mgf1Hash, const unsigned char *sigbuf, unsigned int siglen)
<-> rsa
<- ctx
-> rsa_pad_mode
-> saltlen
-> mgf1Hash
-> sigbuf
-> siglen
<- Return

FIPS_rsa_verify_digest in file ./fips/rsa/fips_rsa_sign.[o|c]

int **FIPS_rsa_verify_digest**(struct rsa_st *rsa, const unsigned char *dig, int diglen, const struct env_md_st *mhash, int rsa_pad_mode, int saltlen, const struct env_md_st *mgf1Hash, const unsigned char *sigbuf, unsigned int siglen)
<-> rsa
-> dig
-> diglen
-> mhash
-> rsa_pad_mode
-> saltlen
-> mgf1Hash
-> sigbuf
-> siglen
<- Return

FIPS_rsa_x931_derive_ex (renames RSA_X931_derive_ex) in file ./crypto/rsa/rsa_x931g.[o|c]

int **FIPS_rsa_x931_derive_ex**(RSA *rsa, BIGNUM *p1, BIGNUM *p2, BIGNUM *q1, BIGNUM *q2, const BIGNUM *Xp1, const BIGNUM *Xp2, const BIGNUM *Xp, const BIGNUM *Xq1, const BIGNUM *Xq2, const BIGNUM *Xq, const BIGNUM *e, BN_GENCB *cb)
<-> rsa
<-> p1
<-> p2
<-> q1
<-> q2
-> Xp1
-> Xp2
-> Xp
-> Xq1
-> Xq2
-> Xq
-> e
<-> cb
<- Return

FIPS_rsa_x931_generate_key_ex (renames RSA_X931_generate_key_ex) in file ./crypto/rsa/rsa_x931g.[o|c]

int **FIPS_rsa_x931_generate_key_ex**(RSA *rsa, int bits, const BIGNUM *e, BN_GENCB *cb)
<-> rsa
-> bits
-> e
<-> cb
<- Return

FIPS_selftest in file ./fips/fips_post.[o|c]

int **FIPS_selftest**()
<- Return

FIPS_selftest_aes in file ./fips/aes/fips_aes_selftest.[o|c]

int **FIPS_selftest_aes**()
<- Return

FIPS_selftest_aes_ccm in file ./fips/aes/fips_aes_selftest.[o|c]

int **FIPS_selftest_aes_ccm**()
<- Return

FIPS_selftest_aes_gcm in file ./fips/aes/fips_aes_selftest.[o|c]

int **FIPS_selftest_aes_gcm**()
<- Return

FIPS_selftest_aes_xts in file ./fips/aes/fips_aes_selftest.[o|c]

int **FIPS_selftest_aes_xts**()
<- Return

FIPS_selftest_check in file ./fips/fips.[o|c]

void **FIPS_selftest_check**()

FIPS_selftest_cmac in file ./fips/cmac/fips_cmac_selftest.[o|c]

int **FIPS_selftest_cmac**()
<- Return

FIPS_selftest_des in file ./fips/des/fips_des_selftest.[o|c]

int **FIPS_selftest_des**()
<- Return

FIPS_selftest_drbg in file ./fips/rand/fips_drbg_selftest.[o|c]

int **FIPS_selftest_drbg**()
<- Return

FIPS_selftest_drbg_all in file ./fips/rand/fips_drbg_selftest.[o|c]

int **FIPS_selftest_drbg_all**()
<- Return

FIPS_selftest_dsa in file ./fips/dsa/fips_dsa_selftest.[o|c]

int **FIPS_selftest_dsa**()
<- Return

FIPS_selftest_ecdh in file ./fips/ecdh/fips_ecdh_selftest.[o|c]

int **FIPS_selftest_ecdh**()
<- Return

FIPS_selftest_ecdsa in file ./fips/ecdsa/fips_ecdsa_selftest.[o|c]

int **FIPS_selftest_ecdsa**()
<- Return

FIPS_selftest_failed in file ./fips/fips.[o|c]

int **FIPS_selftest_failed**()
<- Return

FIPS_selftest_hmac in file ./fips/hmac/fips_hmac_selftest.[o|c]

int **FIPS_selftest_hmac**()
<- Return

FIPS_selftest_rsa in file ./fips/rsa/fips_rsa_selftest.[o|c]

int **FIPS_selftest_rsa**()
<- Return

FIPS_selftest_sha1 in file ./fips/sha/fips_sha1_selftest.[o|c]

int **FIPS_selftest_sha1**()
<- Return

FIPS_selftest_x931 in file ./fips/rand/fips_rand_selftest.[o|c]

int **FIPS_selftest_x931**()
<- Return

FIPS_set_error_callbacks in file ./fips/utl/fips_err.[o|c]

void **FIPS_set_error_callbacks**(void (*put_cb)(int lib, int func,int reason,const char *file,int line), void (*add_cb)(int num, va_list args))
-> put_cb
<- add_cb

FIPS_set_locking_callbacks in file ./fips/utl/fips_lck.[o|c]

void **FIPS_set_locking_callbacks**(void (*func)(int mode, int type, const char *file,int line), int (*add_cb)(int *pointer, int amount, int type, const char *file, int line))
-> func
-> add_cb

FIPS_set_malloc_callbacks in file ./fips/utl/fips_mem.[o|c]

void **FIPS_set_malloc_callbacks**(void *(*malloc_cb)(int num, const char *file, int line), void (*free_cb)(void *))
-> malloc_cb
<-> free_cb

FIPS_text_end in file ./fips/fips_end.[o|c]

void ***FIPS_text_end**()
<- Return

FIPS_text_start in file ./fips/fips_start.[o|c]

void ***FIPS_text_start**()
<- Return

FIPS_x931_bytes in file ./fips/rand/fips_rand.[o|c]

int **FIPS_x931_bytes**(unsigned char *out, int outlen)
<- out
-> outlen
<- Return

FIPS_x931_method in file ./fips/rand/fips_rand.[o|c]

const RAND_METHOD ***FIPS_x931_method**()
<- Return

FIPS_x931_reset in file ./fips/rand/fips_rand.[o|c]

void **FIPS_x931_reset**()

FIPS_x931_seed in file ./fips/rand/fips_rand.[o|c]

int **FIPS_x931_seed**(const void *buf, int num)
-> buf
-> num
<- Return

FIPS_x931_set_dt in file ./fips/rand/fips_rand.[o|c]

int **FIPS_x931_set_dt**(unsigned char *dt)
<- dt
<- Return

FIPS_x931_set_key in file ./fips/rand/fips_rand.[o|c]

int **FIPS_x931_set_key**(const unsigned char *key, int keylen)

-> key
-> keylen
<- Return

FIPS_x931_status in file ./fips/rand/fips_rand.[o|c]

int **FIPS_x931_status**()
<- Return

FIPS_x931_stick in file ./fips/rand/fips_rand.[o|c]

void **FIPS_x931_stick**(int onoff)
-> onoff

FIPS_x931_test_mode in file ./fips/rand/fips_rand.[o|c]

int **FIPS_x931_test_mode**()
<- Return