OpenSSL FIPS Object Module

Version 1.2

By the

Open Source Software Institute

http://www.oss-institute.org/

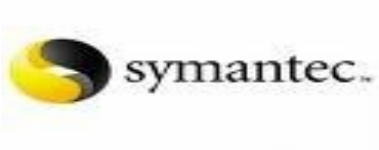
OpenSSL FIPS 140-2 User Guide

June 25, 2012

Copyright Notice

Sponsored by

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



Acknowledgments

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

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

John Weathersby
Executive Director
Open Source Software Institute
8 Woodstone Plaza
Ste. 101
Hattiesburg, MS 39402
USA
601-427-0152 office/601-818-7161 cell
jmw@oss-institute.org
601-427-0156 fax
http://oss-institute.org/

with technical work by:

Stephen Henson
4 Monaco Place,
Westlands, Newcastle-under-Lyme
Staffordshire. ST5 2QT.
England, United Kingdom
shenson@drh-consultancy.co.uk
http://www.drh-consultancy.co.uk/

in coordination with

Andy Polyakov
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
appro@fy.chalmers.se

Tim Hudson
P.O. Box 6389
Fairfield Gardens 4103
Australia
ACN 074 537 821
+61 7 3103 0321
tjh@cryptsoft.com
http://www.cryptsoft.com/

and the OpenSSL Team at www.openssl.org.

Validation testing relating to this software was performed by The DOMUS IT Security Laboratory...

Christian Brych
FIPS 140 Program Manager
DOMUS IT Security Laboratory
400 March Road, Suite 190
Kanata, Ontario
Canada, K2K 3H4
613-726-5091 office
613-867-1241 cell
cbrych@domusitsl.com
http://www.domusitsl.com/

...and by Aspect Labs, a division of BKP Security, Inc.:

Aspect Labs
3080 Olcott Street, Suite 110-A
Santa Clara, CA 95054-3221
USA

888-347-7140
info@aspectlabs.com
http://www.aspectlabs.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.

Date	Description
2012-06-25	Document new Apple iOS and OS X support
2009-11-21	Extensive changes to cover the uClinux change letter update and incorporate numerous editorial suggestions from Tim Hudson.
2009-10-25	Removed superfluous and confusing footnote in section 4.2.2 and correction to command line in section 4.2.1, reported by Patrick Rael; updated contact info in section 6.
2009-10-22	Updated acknowledgments contact info; added announcement URL to section 1; correction in 5.5 para 3, reported by Henry Unger and Patrick Rael
2009-05-08	Several corrections in Appendix B.
2009-04-25	Corrected typos in 2.4, reported by Steve Weymann.
2009-04-18	Added discussion of new *fipsalgtest.pl* to Appendix B.
2009-01-11	Fix references to allowed buildtime options in 4.2, more error coded in Appendix D.
2008-12-17	Fix error in Example 5.2b.
2008-12-07	Multiple corrections from initial draft release.
2008-09-26	Finalized draft at validation award for initial release.
2008-02-29	Incorporated extensive input from a detailed review by Conrad G. Welling.
2007-09-22	Initial draft for openssl-fips-1.2.tar.gz.

Table of Contents

1. INTRODUCTION....8

2. BACKGROUND....9

2.1 Terminology....9
2.2 The FIPS Module and Integrity Test....11
2.3 The FIPS Integrity Test....11
2.3.1 Requirement for Exclusive Integrity Test....12
2.3.2 Requirement for Fixed Object Code Order....12
2.4 The File Integrity Chain....13
2.4.1 Source File (Build Time) Integrity....13
2.4.2 Object Module (Link Time) Integrity....13
2.4.3 Application Executable Object (Run Time) Integrity....14
2.5 Relationship to the OpenSSL API....14
2.6 FIPS Mode of Operation....15
2.6.1 Initialization....16
2.6.2 Algorithms Available in FIPS Mode....16

3. COMPATIBLE PLATFORMS....18

3.1 Build Environment Requirements....18
3.2 Known Supported Platforms....19
3.2.1 Code Paths and Command Sets....20
3.2.2 Assembler Optimizations....21
3.2.3 32 versus 64 Bit Architectures....22

4. GENERATING THE FIPS OBJECT MODULE....23

4.1 Delivery of Source Code....23
4.1.1 Creation of a FIPS Object Module from Other Source Code....23
4.1.2 Verifying Integrity of Distribution....23
4.1.3 Verifying Integrity of the Full Distribution for the FIPS Object Module....26
4.2 Building and Installing the FIPS Object Module with OpenSSL (Unix/Linux)....26
4.2.1 Building the FIPS Object Module from Source....26
4.2.2 Installing and Protecting the FIPS Object Module....27
4.2.3 Building a FIPS Capable OpenSSL....28
4.3 Building and Installing the FIPS Object Module with OpenSSL (Windows)....29
4.3.1 Building the FIPS Object Module from Source....29
4.3.2 Installing and Protecting the FIPS Object Module....29
4.3.3 Building a FIPS Capable OpenSSL....30

5. CREATING APPLICATIONS WHICH REFERENCE THE FIPS OBJECT MODULE..32

5.1 Exclusive Use of the FIPS Object Module for Cryptography....32
5.2 FIPS Mode Initialization....32

5.3 Generate Application Executable Object....34
5.3.1 Linking under Unix/Linux....34
5.3.2 Linking under Windows....36
5.4 Application Implementation Recommendations....37
5.5 Documentation and Record-keeping Recommendations....38
5.6 When is a Separate FIPS 140-2 Validation Required?....39

6. FUTURE PLANS....43

7. REFERENCES....45

APPENDIX A OPENSSL DISTRIBUTION SIGNING KEYS....46

APPENDIX B CMVP TEST PROCEDURE....47

B.1 Building the Software - Linux/Unix....47
B.2 Algorithm Tests - Linux/Unix....47
B.3 Building the Software - Windows....49
B.4 Algorithm Tests - Windows....49
B.5 FIPS 140-2 Test - All Platforms....50
B.6 Testvector Data Files and the mkfipsscr.pl Utility....51
B.7 Files for a Runtime Validation....57
B.8 Retrofitting the fipsalgtest.pl Utility....58

APPENDIX C EXAMPLE OPENSSL BASED APPLICATION....61

APPENDIX D FIPS API DOCUMENTATION....64

D.1 FIPS_mode....64
D.2 FIPS_mode_set(), FIPS_selftest()....65
D.3 Error Codes....65

APPENDIX E PLATFORM SPECIFIC NOTES....67

E.0 Nomenclature for ABIs:....67
E.1 Compiler placement of read-only data....69
E.2 Bugs in Microsoft TLS Implementation....70
E.3 Solaris and gcc Problems....71
E.5 HP-UX Vendor Support....72
E.6 Apple OS X Support....72
E.7 Apple iOS Support....74

APPENDIX F RESTRICTIONS ON THE EXPORT OF CRYPTOGRAPHY....77

F.1 Open Source Software....77
F.2 "Export Jobs, Not Crypto"....78

APPENDIX G SECURITY POLICY ERRATA....79

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 product. 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/140sp1051.pdf.

For more information on OSSI see http://www.oss-institute.org/. 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.

2. Background

For the purposes of FIPS 140-2 validation the OpenSSL FIPS Object Module v1.2 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 this object code in an isolated and separated form that is used to provide a cryptographic API (Application Programming Interface) to external applications. The term *FIPS Object Module* elsewhere in this document refers 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 0.9.8 distributions beginning with 0.9.8j. Note: OpenSSL 1.0.0 is not supported for use with the OpenSSL FIPS Object Module. These standard OpenSSL 0.9.8 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 other than DH 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 requirements. As such, 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 in sections 4 and 5 .

The FIPS Object Module provides confidentiality, integrity and message digest services. The FIPS Object Module supports the following algorithms: Triple DES, AES, RSA (for digital signatures), DH, DSA, 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 an ANSI X9.31 compliant pseudo-random number generator.

2.1 Terminology

During the course of the OpenSSL validation it became clear that a mapping between the tterminology used by the OpenSSL developers and cryptographers and that of the CMVP and FIPS 140-2 specialists. In this section some of these distinctions are discussed.

Approved Mode

[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 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[3] 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

The concept of the cryptographic module is an important one for FIPS 140-2, with subtle nuances . In the context of FIPS 140-2 "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 1340-2 context.

Conceptually the Module is the binary object code and data in the FIPS Object Module for a running process. This binary object as it resides in memory is verified by the FIPS integrity test.

Note that traditionally the executable (or shared library) file on disk corresponding to this Module as a running process is also considered by the CMVP to be a Module[4]; 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[5] 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.

[3]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.
[4]Presumably because the transformations of the disk resident file contents performed by the runtime loader are considered to be well understood and sufficiently minimal.
[5]The definition of what constitutes "extraneous" is not formally specified and subject to interpretation.

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 continuing with each step in the transformation of that source code into a running process is established to provide the equivalent check 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.

2.2 The FIPS Module and Integrity Test

The FIPS Object Module is generated in binary file form, with an embedded pre-calculated HMAC-SHA-1 digest covering the module[6] 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 necessarily resides in an area not included in the calculated digest[7]. This "in-core hashing" integrity test is designed to be both executable format independent and fail-safe.

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

[6]Specifically, the text and read-only data segments which constitute the initialized components of the module.
[7]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.

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 not affect the cryptographic module integrity test[8].

This requirement is met with an in-core integrity test that carefully excludes any extraneous 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
```

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
```

[8]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.

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. In the case of validated pre-built binary executables neither the FIPS 140-2 CMVP test lab nor the end user has an independent means of directly verifying that the vendor supplied software is actually derived from the validated code. 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.

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 use with the OpenSSL API. Applications linked with the FIPS Object Module and with the separate OpenSSL libraries can use both the FIPS validated cryptographic functions of the FIPS Object Module and the high level functions of OpenSSL.

First, some important definitions to avoid confusion. The *FIPS Object Module* is the special monolithic object module built from the special source distribution identified in the *Security Policy*. It is *not* the same as the OpenSSL product or any specific official OpenSSL distribution release.

A version of the OpenSSL product that is suitable for reference by an application along with the FIPS Object Module is a *FIPS compatible OpenSS*L.

When the FIPS Object Module and a FIPS compatible OpenSSL are separately built and installed on a system, 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 the object modules comprising the FIPS Object Module could easily be incorporated directly in the OpenSSL `libcrypto.a` library, but for the fact that such a combination (linking to a static library) is specifically forbidden by FIPS 140-2 and the CMVP[9].

Various non-FIPS algorithms such as Blowfish, IDEA, CAST, MD2, etc. will be included in the OpenSSL libraries (depending on the `./config` options specified in addition to `fips`). For applications that do not care about FIPS 140-2 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; a non-FIPS compatible OpenSSL library cannot be substituted for the FIPS compatible library for an application using FIPS mode because the FIPS specific function calls, in particular `FIPS_mode_set()`, will not be present.

Applications that do wish to use FIPS mode must call the `FIPS_mode_set()` function, and after the runtime FIPS mode initialization the non-FIPS algorithms are disabled by default.

In this sense the combination of the FIPS Object Module and the usual OpenSSL libraries constitutes a "FIPS capable API", capable of being used for either FIPS mode operation or for conventional non-FIPS purposes as before.

2.6 FIPS Mode of Operation

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. The enabling of runtime FIPS mode is explicitly performed by a function call.

---

[9]Actually, to encourage use of `fipscanister.o` even in non-FIPS mode application, 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.

2.6.1 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 table 4.5a 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.

By design, the OpenSSL API attempts to disable non-FIPS algorithms, when in FIPS mode, at the EVP level and via 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 a 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 no non-FIPS algorithms are used when in FIPS mode.

OpenSSL provides a mechanism, the "ENGINE" component, for interfacing with external cryptographic devices such as accelerator cards. 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 compliant. In general, the OpenSSL FIPS 140-2 validation still holds true in the case that other separately FIPS 140-2 validated hardware or software modules are utilized in conjunction with the OpenSSL FIPS validated cryptographic module.

However, if any cryptographic operations are performed by a non-FIPS validated device the result is not FIPS compliant. It is the responsibility of the application developer to ensure that any use of devices via the OpenSSL ENGINE support are for FIPS validated devices only (when in FIPS mode).

3. Compatible Platforms

The FIPS Object Module was built from source and tested on specific hardware/software environments (See the *Security Policy*) . The FIPS Object Module maintains vendor affirmed FIPS 140-2 compliance on other operating systems provided that the conditions described in Section 4 of the *Security Policy* apply.

The FIPS Object Module is designed to run on a wide range of hardware and software platforms. Any such 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 v 1.2* was developed all Unix®[10]-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 Note also that future releases of OpenSSL may support additional platforms requiring new or changed source from that of the current FIPS source distribution, in which case use of OpenSSL with the FIPS Object Module will not be validated for those new platforms.

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 only for the x86/Intel®[11] ELF machine architecture. The FIPS Object Module build process will automatically select and include these assembly routines by default when performed on a x86 platform. This assembly language code was included in the validation testing and hence a FIPS Object Module built using the x86/Intel® assembly language routines is a FIPS 140-2 validated FIPS Object Module.

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®.

[10]UNIX is a registered trademark of The Open Group
[11]Intel is a registered trademark of the Intel Corporation

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 the patch openss fips1.2.crossbuild.diff.gz as noted in the Security Policy is applied and build option "U2" is used for cross-compilation support where the in core hash is calculated on the build host.

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[12]). 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 written:

- Linux®[13] on x86, x86_64, IA64
- SPARC® Solaris®[14], both 32 and 64 bit ABIs
- x86 Solaris® on 32 and 64 bit ABIs
- IRIX®[15] 6.5, n32, and 64-bit ABIs
- Tru64®[16]
- HP-UX®[17], 32 and 64 bit PA-RISC®, 32 and 64 bit IA64
- Mac OS X®[18] on PowerPC®[19]
- Windows®[20] with Cygwin®[21] and Mingw

[12]A convenient way of generating a chared library containing `fipscanister.o` is discussed in Appendix B
[13]Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
[14]SPARC and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.
[15]IRIX is a trademark of Silicon Graphics, Inc.
[16]Tru64 is a registered trademark of Compaq Computer Corporation.
[17]HP-UX is a registered trademark of Hewlett-Packard Company.
[18]Mac OS X is a registered trademark of Apple Computer, Inc.
[19]PowerPC is a trademarks of International Business Machines Corporation in the United States, other countries, or both.
[20]Windows is a registered trademark of Microsoft Corporation in the United States and other countries.
[21]Cygwin is a registered trademark of Red Hat, Inc.

- OpenBSD®[22] and FreeBSD®[23] on x86
- uClinux®[24] on ARM

Among the platforms known to not be supported are VMS[25] and Windows CE.

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 the assembler optimizations for x86 and x86_64 were tested, so only those optimizations are available for building the FIPS Object Module. Four separate sets of source code were identified to cover plain C (no assembler) for both 32 and 64 bits, and optimizations for x86 and x86_64.

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

The resulting eight 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
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 fipscanisterbuild no-asm`

[22]OpenBSD is a registered trademark of Theo de Raadt.
[23]FreeBSD is a registered trademark of the FreeBSD Foundation.
[24]uClinux is a registered trademark of Arcturus Networks Inc.
[25]VMS is a registered trademark of Digital Equipment Corporation.

Command Set Name		Build Commands
		`make` `make install`

u2	Linux/Unix with x86/x86_64 optimizations	`./config fipscanisterbuild`
		`make`
		`make install`

w1	Windows, pure C	`ms\do_fips no-asm`
w2	Windows with x86/x86_64 optimizations	ms\do_fips

u1	Linux/Unix, pure C, cross-compiled	`./config fipscanisterbuild`
		`make`
		`make install`

3.2.1b - Command Sets

The actual representative systems tested for the validation were:

Generic System		***Actual System***
U1	Linux x86 no-asm	Linux.2.6.18_i686_gcc-4.1.2 (OpenSuSE 10.2)
U2	Linux x86-64 no-asm	Linux.2.6.20_x86-64_gcc-4.1.2 (F7)
U3	Linux x86 asm	Linux.2.6.18_i686_gcc-4.1.2 (OpenSuSE 10.2)
U4	Linux x86-64 asm	Linux.2.6.20_x86-64_gcc-4.1.2 (F7)
W1	Windows x86 no-asm	WinXP.SP2_i386_MSVC.8.0 no-asm
W2	Windows x64 no-asm	WinXP.SP2_x86-64_MSVC.8.0 no-asm
W3	Windows x86 asm	WinXP.SP2_i386_MSVC.8.0 NASM, SSE2
W4	Windows x64 asm	WinXP.SP2_x86-64_MSVC.8.0
C1	Linux ARM no-asm	Linux.2.4.32-uc0_armv4l gcc-3.4.4 (uCLinux)

Table 3.2.1c - Representative Systems

3.2.2 Assembler Optimizations

The only option for processor architectures other than x86 and x86_64 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 x86 or x86_64 processors either the plain C or hand-coded assembler build can be performed, though there would be no advantage (from a FIPS perspective) in building the plain C version.

3.2.3 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 out of the scope of this document.

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)

4. Generating the FIPS Object Module

This section describes the creation of a FIPS Object Module for subsequent use by an application.

4.1 Delivery of Source Code

The OpenSSL FIPS Object Module software is available only in source format. The specific distribution was taken from the FIPS development branch which was a mid-release version of OpenSSL after 0.9.8e and before 0.9.8f, . This specific distribution can be found at http://www.openssl.org/source/openssl-fips-1.2.tar.gz.

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.

Note this downloaded file is untrusted for any purpose until verified as described in §4.1.2, and untrusted for the purposes of building a FIPS Object Module until verified as described in §4.1.3.

Note that while the specific FIPS Object Module Distribution *could* be used to build the conventional OpenSSL libraries in addition to the FIPS Object Module, it should *not* be used for that purpose. Use of a newer version of OpenSSL (0.9.8.j+) linking against the FIPS module is effectively mandatory due to several compilation issues with `gcc` versions 4.2 which were present at the parent 0.9.8e/f baseline and not addressed until later.

4.1.1 Creation of a FIPS Object Module from Other Source Code

Some OpenSSL distributions other than the specific distribution 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

The integrity and authenticity of the complete OpenSSL distribution should be validated manually with the PGP signatures[26] 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 specific FIPS 140-2 source file integrity check (§4.1.3).

[26]Note this PGP/GPG signature check is *not* related to any of the FIPS integrity checks!

The PGP signatures are contained in the file

```
openssl-fips-1.2.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/)[27]. 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 be 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-0.9.8z.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
$
```

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.

[27]Note that although PGP and GPG are functionally interoperable, some versions of PGP are currently FIPS 140 validated and no versions of GPG are. For the purposes of FIPS 140-2 validation a validated version of PGP must be used. The examples given here are applicable to both GPG and PGP.

```
$ 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

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

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-0.9.8z.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 is verified, i.e. the target file *`openssl-fips-1.2.tar.gz`* was signed by the key with id `49A563D9`. The warning message in

this example is noting the fact that the key is not already 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.

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

At this point the signature of the distribution has been verified as belonging to the signing key, and the authenticity of the signing key has been verified against a reference (this document) downloaded from the NIST web site.

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

A separate source file integrity check is required to meet the requirements of FIPS 140-2.

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-1.2.tar.gz
```

where the `openssl` command is from a recent version of OpenSSL that supports the `-hmac` option[28]. If you don't have the `openssl` command yet it will be generated by the build process.

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.

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 generated FIPS Object Module when the `fips` configuration option is

[28]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 fipscanisterbuild
or
      ./config fipscanisterbuild no-asm
```

where the specific option used depends on the platform (see §3.2.1).

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

```
      ./config fipscanisterbuild 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[29].

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 fipscanisterbuild`" or "`make`" commands are not permitted. If the original distribution is modified, or if anything than those three specified commands are used, or if any intermediate files are modified, the result is *not* FIPS validated.

4.2.2 Installing and Protecting the FIPS Object Module

[29] 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 H for one known exception.

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 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-1.0/lib/* for Unix® based or Linux® systems.) with the appropriate permissions to satisfy the security requirement. These four files constitute the validated FIPS Object Module, the (many) 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 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

At this point a full OpenSSL library has been installed. However, the special distribution required to generate the validated FIPS Object Module does not correspond exactly to any official OpenSSL releases. Once the validated FIPS Object Module has been generated the other OpenSSL components can be replaced with components from a different OpenSSL distributions. Any 0.9.8 releases from j onwards (i.e. 0.9.8j or above) can be used for this purpose. OpenSSL 1.0 is not compatible. The commands

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

will install the new OpenSSL without overwriting the validated FIPS Object Module files. The `--with-fipslibdir` 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 0.9.8j 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.obj`, `fipscanister.obj.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 0.9.8 distribution, 0.9.8j or later. Follow the standard Windows® build procedure except that instead of the command:

```
perl Configure VC-WIN32
```

do:

```
perl Configure VC-WIN32 fips --with-fipslibdir=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_RELATED
```

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.o` 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. This checklist 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.

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()

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 a `alg_section` specification naming a section that can contain an optional "`fips_mode = yes`" statement.

```
 # Default section
XXXX_conf = XXXX_options

...

[ XXXX_options ]
alg_section = algs

...

[ algs ]
fips_mode = yes

...
```

Example 5.2c – Sample openssl.conf File

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[30] 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.

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

[30]Note that where `fipscanister.o` 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.

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[31]:

```
OPENSSLDIR = /usr/local/ssl/fips-1.0
FIPSMODULE = $(OPENSSLDIR)/lib/fipscanister.o
     .
     .
     .
<application>: $(OBJS) $(FIPSMODULE)
    env FIPSLD_CC=$(CC) fipsld $(CFLAGS) -o $@ $(OBJS) \
         $(LIBS) $(LIBCRYPTO)
```

[31]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.

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

```
<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 functions 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 #1051. Instead a statement similar to the following is recommended:

> *Product XXXX uses an embedded FIPS 140-2-validated cryptographic module (Certificate #1051) 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”).

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-1.2.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 fipscanisterbuild [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 0.9.8j+ 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/cryptval/140-1/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.

A typical network protocol, such as IPSec/IKE, TLS, SSH, S-MIME or 802.11 protocol family may provide a complex variety of services. Some aspects of such services may have a cryptographic nature and utilize Approved or "allowed for use" cryptographic algorithms, such as encryption, decryption, signatures, hashes, message digests and others. Other services provided by a network protocol may be of non-cryptographic nature, such as packet routing, packet assembly/disassembly, defragmentation, radio and link layer communications, firewalling, network address translation, address resolution, quality of service, re-transmission and others. While the ultimate verdict for a particular solution belongs to the CMVP, it is generically logical to assume that non-cryptographic services of a particular network protocol or a set of protocols may be implemented outside of a validated cryptographic module. This is also logical having in mind that in many cases non-cryptographic services of a particular protocol may be delegated to other devices in the IT solution. For instance, in some wireless LAN access systems an implementation of the 802.11 protocol set is provided jointly by a wireless access switch and a wireless access point, where the wireless access point may provide non-cryptographic services of the 802.11 protocol set such as radio transmissions, frequency and signal strength control, initial wireless client association and others. Another widely used example is a web server offloading cryptographic functionality of the HTTPS/TLS protocol to a FIPS 140-2 validated cryptographic accelerator card (many such cards are available on the market).

In addition to consulting the written CMVP guidance, it is then also important to consider industry-wide interpretation patterns and precedents in this field. After performing a review of the FIPS 140-2 validated products list http://csrc.nist.gov/cryptval/140-1/140val-all.htm one may conclude that implementing non-cryptographic services of a particular network protocol outside of a validated cryptographic module can in many cases be considered as an industry trend. There are multiple examples which illustrate such a trend. For illustration purposes only we can take a look at the example of the Microsoft Kernel Module

http://csrc.nist.gov/cryptval/140-1/140sp/140sp241.pdf

Note that there are many other modules which follow a similar trend, this module is just one example out of many. The analysis here is generic, applies to a large number of validated modules, and is not intended to make any specific statements as to the validation of this particular module.

As specified by the vendor, the Kernel Module is used by the vendor implementation of the IPSec/IKE protocol

http://www.microsoft.com/technet/archive/security/topics/issues/fipseval.mspx?mfr=true

In particular it is stated that

> *"Both IPSEC and EFS in Windows 2000, XP, and Server 2003 use the FIPS-140-1 or FIPS 140-2 (as appropriate) evaluated Kernel Mode Cryptographic Module to encrypt the traffic packet data and file contents respectively if configured appropriately with the selections of FIPS compliant algorithms."*

A review of the Kernel Module Security Policy then shows that the module's services are specified as services performing cryptographic algorithms supported by IPSec/IKE(such as encryption/decryption and key agreement) and not as providing a full IPSec/IKE protocol implementation. This could again serve as an illustration of the fact that non-cryptographic services of a particular protocol are in many cases implemented outside of a cryptographic module. A similar analysis could be performed for other protocols specified in

http://www.microsoft.com/technet/archive/security/topics/issues/fipseval.mspx?mfr=true

such as S/MIME (used in Outlook), TLS (used in Explorer), Remote Desktop Protocol and Encrypting File System. Other examples can be discussed by analyzing the list of historically validated products http://csrc.nist.gov/cryptval/140-1/140val-all.htm. In general the vendor of a product based on the OpenSSL FIPS Object Module should be able to find an existing validated module, and similar products claiming validation by virtue of that module, for one of several large well-known IT companies.

In conclusion, both the historical perspective and the current CMVP guidance point to a possibility of non-cryptographic services in an IT solution being implemented outside of a validated cryptographic module. We are not aware of any CMVP regulations explicitly denying use of embedded validated cryptographic modules to satisfy the requirements of FIPS 140-2 statement, provided that the set of conditions specified in the CMVP FAQ and other relevant documentation is satisfied. With this in mind, the ultimate decision for a particular product/protocol belongs to the CMVP and the analysis presented here can serve for discussion purposes only.

The main job of a FIPS 140-2 testing lab is to consistently apply the FIPS 140-2 standard, the Derived Test Requirements and the Implementation Guidance to all modules which the testing lab checks. Testing labs can have suggestions for further improvements which could be communicated to the CMVP and potentially incorporated in the future versions of the standard or in the implementation guidance.

Since the CMVP does not have a formal program for validation of IT solutions with embedded FIPS 140-2 modules, the question is how is the actual compliance/non-compliance 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, OSSI can perform a review and provide an opinion letter stating whether, based on information provided by the vendor, that product appears to OSSI 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.

6. Future Plans

The current validated FIPS Object Module has a number of limitations, among them:

1. Not all FIPS compliant cryptographic algorithms are validated (ECC, for example). An algorithm test involves coding a test driver (`dsa/fips_dsatest.c`, `hmac/fips_hmactest.c`, etc. in the `./fips/` subdirectory) to process input test vectors in a format described in a standards document. Most of these were relatively straightforward, if tedious, to code, though a few required considerable trial-and-error experimentation to interpret.
2. Not compatible with OpenSSL 1.0 where a number of data structures are not binary compatible with OpenSSL 0.9.8 on which the FIPS module is based.

These first OpenSSL FIPS validations were pioneer efforts that confronted may uncertainties in the validation process with limited resources. Now that the initial validations have been awarded and the requirements for a source code validation are better understood these limitations can be addressed in future validations.

Any such future validation will have one fixed cost, the CMVP testing laboratory fee, plus whatever costs are associated with the specific software development items. For that reason the more financial co-sponsors are involved in a given validation, the lower the cost to each sponsor.

Any interested parties willing to consider co-sponsoring a future OpenSSL FIPS 140-2 or 140-3 validation effort should contact

```
Steve Marquess                      877-673-6775 office
OpenSSL Software Foundation         marquess@opensslfoundation.com
1829 Mount Ephraim Road             http://opensslfoundation.com/
Adamstown, MD  21710
USA
```

or

```
John Weathersby                     601-427-0152 office
Executive Director                  601-818-7161 cell
Open Source Software Institute      601-427-0156 fax
8 Woodstone Plaza                   jmw@oss-institute.org
```

```
Ste. 101                                http://oss-institute.org/
Hattiesburg, MS  39402
USA
```

for information on current plans and the opportunities for participation. Note plans for new validations will be announced at http://www.openssl.org/docs/fips/fipsnotes.html.

7. REFERENCES

1. *OpenSSL FIPS 140-2 Security Policy*, Version 1.2, Open Source Software Institute. This document is available at http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1051.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. *Network Security with OpenSSL*, John Viega et. al., 15 June 2002, O'Reilly & Associates

The algorithm certificates:

RSA	http://csrc.nist.gov/groups/STM/cavp/documents/aes/aesval.html#323
DSA	http://csrc.nist.gov/groups/STM/cavp/documents/dss/dsaval.htm#264
3DES	http://csrc.nist.gov/groups/STM/cavp/documents/des/tripledesval.html#627
AES	http://csrc.nist.gov/groups/STM/cavp/documents/aes/aesval.html#695
HMAC	http://csrc.nist.gov/groups/STM/cavp/documents/mac/hmacval.html#373
SHA-1,2	http://csrc.nist.gov/groups/STM/cavp/documents/shs/shaval.htm#723
RNG	http://csrc.nist.gov/groups/STM/cavp/documents/rng/rngval.html#407

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.

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**
49A563D9	Mark Cox <mark@awe.com>
Fingerprint	7B 79 19 FA 71 6B 87 25 0E 77 21 E5 52 D9 83 BF
26BB437D	Ralf S. Engelschall <rse@engelschall.com>
Fingerprint	00 C9 21 8E D1 AB 70 37 DD 67 A2 3A 0A 6F 8D A5
F295C759	Dr Stephen Henson <shenson@drh-consultancy.co.uk>
Fingerprint	D0 5D 8C 61 6E 27 E6 60 41 EC B1 B8 D5 7E E5 97
9C58A66D	Lutz Janicke <Lutz.Jaenicke@aet.TU-Cottbus.DE>
Fingerprint	13 D0 B8 9D 37 30 C3 ED AC 9C 24 7D 45 8C 17 67
2118CF83	Ben Laurie <ben@cryptix.org>
Fingerprint	7656 55DE 62E3 96FF 2587 EB6C 4F6D E156 2118 CF83
E06D2CB1	Richard Levitte <levitte@lp.se>
Fingerprint	35 3E 6C 9E 8C 97 85 24 BD 9F D1 9E 8F 75 23 6B
5A6A9B85	Bodo Moeller <2004@bmoeller.de>
Fingerprint	C7 AC 7E AD 56 6A 65 EC F6 16 66 83 7E 86 68 28

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://www.openssl.org/docs/fips/.

B.1 Building the Software - Linux/Unix

1. Copy the OpenSSL distribution (openssl-fips-1.2.tar.gz) to a directory on the test system. Approximately 80Mb free space is needed.

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

   ```
   gunzip -c openssl-fips-1.2.tar.gz | tar xf -
   cd openssl
   ./config fipscanisterbuild [no-asm]
   make
   ```

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

3. Optionally run

   ```
   make test
   ```

   to perform the standard integrity test. A large amount of output is generated. If an error occurs processing will be aborted.

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 mkfipsscr.pl --dir=testvectors --rspdir=resp
sh fipstests.sh
```

   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.

6. The many (approximately 198) generated `*.rsp` files will be found in the `./testvectors/` directory tree under `./fips/`:

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

7. The generates responses can be tested against another set of known good testvector responses at pathname FIPS_TVOK=`<pathname>` with:

```
make fips_test_diff FIPS_TVOK=<pathname>
```

   This comparison automatically suppresses the whitespace and comment line differences and ignores the seven testvector files that are always different[32].

8. The tree of `*.rsp` files can 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
```

[32]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

These files are listed in the file `./fips/fips-nodiff.txt` that is referenced by the `fips_test_diff` makefile target.

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-1.2.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 mkfipsscr.pl --win32 --dir=testvectors --rspdir=resp
.\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 following is produced:

```
$ ./fips_test_suite
        FIPS-mode test application

1. Non-Approved cryptographic operation test...
        a. Included algorithm (D-H)...successful
2. Automatic power-up self test...successful
3. AES encryption/decryption...successful
4. RSA key generation and encryption/decryption...successful
5. DES-ECB encryption/decryption...successful
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
8. Non-Approved cryptographic operation test...
        a. Included algorithm (D-H)...successful as expected
9. Zero-ization...
 Generated 128 byte RSA private key
        BN key before overwriting:
8E0CD7345733DA2922DFDA0C8FC79F5B7F67567E8391C81FA0A3298DF1CE0C6A33646A0840F5
5F098711075F457943FC340719760851E21DB7918A8B0728D4F33F1210FC22A52EF8BB20353F
98BB3C8C7E2FC5C36B49AC28E0932568CFC948E6F4923F42906BC8B14E4071E8960EF17974C8
70C541241B4F3BB3D5F001E45C01
        BN key after overwriting:
8045F430EAD7D1ADF7E3582517692DC69F958844C62FDE68DF62F31A26F1F319BDE04A62DBCB
B0965B7055BB45B613E428B29F22797884DFC1B51B593346F9B9470FB660F91B8FA487AE469A
B7FFC23135CF5107FD62D5E355D613462F08D5D5235D62A897B398F7089FD911144B3AF33492
BD0C5B7FB93B43D26CE26B60E9DF
        char buffer key before overwriting:
                4850f0a33aedd3af6e477f8302b10968
```

```
        char buffer key after overwriting:
                8ff13f1c2311f716f165e24a5042c47d

All tests completed with 0 errors
$
```

2. To demonstrate the effect of a corrupted Known Answer Test invoke the `fips_test_suite` command with one of the arguments `aes, des, dsa, rsa, sha1, rng`. The command must be invoked separately with each argument for the KAT test to fail for each separate algorithm. Output similar to the following will be produced for each algorithm (AES in this example):

```
$ ./fips_test_suite aes
        FIPS-mode test application

3. AES encryption/decryption with corrupted KAT...
24557:error:24064064:random number generator:SSLEAY_RAND_BYTES:PRNG not
seeded:md_rand.c:512:You need to read the OpenSSL FAQ,
http://www.openssl.org/support/faq.html
24557:error:2A068065:lib(42):FIPS_selftest_aes:selftest
failed:fips_aes_selftest.c:92:
Power-up self test failed
$
```

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

The FIPS 140-2 test labs use CMVP provided Windows based software known as the “CAVS tool” to generate the testvector data files used for the algorithm tests. The CAVS tool performance used to be an issue (requiring days to produce testvector data sets) however it has been improved and this is no longer an issue.. The only constant is the name of the actual `*.rsp` files of input data. For the initial validation there were 196 unique file name with 2 duplicate names, for a total of 198 files:

Algorithm	**Number of *.req files**
AES	108
DSA	4
HMAC	1
RNG	6
RSA	0
SHA	15
TDES	55

Total	198

The specific file names are:

HMAC	HMAC.req
AES	CFB8MMT128.req
AES	CFB128VarTxt128.req
AES	CBCGFSbox128.req
AES	CFB128KeySbox256.req
AES	OFBGFSbox192.req
AES	CBCMMT128.req
AES	OFBVarKey256.req
AES	CFB8VarKey256.req
AES	CFB8MCT256.req
AES	CFB1MMT128.req
AES	CFB8KeySbox128.req
AES	OFBMCT128.req
AES	CFB8GFSbox128.req
AES	CFB128MCT192.req
AES	OFBMMT256.req
AES	OFBGFSbox256.req
AES	CFB8MMT192.req
AES	OFBMMT192.req
AES	ECBGFSbox256.req
AES	CFB8VarTxt192.req
AES	ECBKeySbox128.req
AES	CFB1GFSbox256.req
AES	CFB128VarKey256.req
AES	CFB8VarKey128.req
AES	CFB128VarTxt256.req
AES	CFB8VarTxt256.req
AES	CFB128GFSbox192.req
AES	CFB128KeySbox128.req
AES	CFB8MCT192.req
AES	CFB1MMT256.req
AES	CFB1VarKey128.req
AES	CFB8KeySbox256.req
AES	OFBKeySbox256.req
AES	CFB1KeySbox192.req
AES	CFB1MCT192.req
AES	CFB1VarTxt192.req
AES	CBCVarKey192.req
AES	CBCMCT128.req
AES	CFB1MMT192.req
AES	ECBMMT192.req
AES	CBCVarTxt128.req
AES	CFB128VarKey192.req

AES	OFBGFSbox128.req
AES	OFBMCT256.req
AES	CBCKeySbox128.req
AES	CFB1VarTxt256.req
AES	OFBKeySbox192.req
AES	ECBVarTxt192.req
AES	OFBMMT128.req
AES	ECBMMT128.req
AES	ECBKeySbox192.req
AES	OFBVarTxt256.req
AES	CBCGFSbox192.req
AES	CFB128GFSbox256.req
AES	CBCVarKey128.req
AES	CBCVarKey256.req
AES	CFB1GFSbox128.req
AES	CFB1KeySbox256.req
AES	CBCMMT256.req
AES	CFB8VarKey192.req
AES	CFB1VarKey256.req
AES	ECBGFSbox192.req
AES	CBCMMT192.req
AES	CBCKeySbox256.req
AES	CFB128VarTxt192.req
AES	CFB1MCT128.req
AES	CBCGFSbox256.req
AES	ECBVarKey192.req
AES	CFB128KeySbox192.req
AES	CFB128VarKey128.req
AES	CFB128MMT256.req
AES	CBCKeySbox192.req
AES	CBCVarTxt256.req
AES	CFB1GFSbox192.req
AES	OFBMCT192.req
AES	ECBVarTxt256.req
AES	OFBVarKey192.req
AES	CFB128MMT192.req
AES	ECBMCT192.req
AES	CFB1VarKey192.req
AES	OFBVarKey128.req
AES	ECBVarKey128.req
AES	CFB8GFSbox256.req
AES	ECBVarTxt128.req
AES	CBCMCT192.req
AES	CBCMCT256.req
AES	ECBMCT128.req
AES	ECBMMT256.req
AES	CFB8KeySbox192.req
AES	CFB1VarTxt128.req

AES	ECBVarKey256.req
AES	CFB128MMT128.req
AES	CFB1MCT256.req
AES	ECBGFSbox128.req
AES	CFB8GFSbox192.req
AES	OFBKeySbox128.req
AES	CFB128GFSbox128.req
AES	CFB1KeySbox128.req
AES	CFB128MCT256.req
AES	ECBKeySbox256.req
AES	ECBMCT256.req
AES	CBCVarTxt192.req
AES	CFB8MMT256.req
AES	CFB8VarTxt128.req
AES	OFBVarTxt128.req
AES	CFB8MCT128.req
AES	OFBVarTxt192.req
AES	CFB128MCT128.req
TDES	TOFBinvperm.req
TDES	TCFB64Monte2.req
TDES	TECBMonte1.req
TDES	TCBCvartext.req
TDES	TECBpermop.req
TDES	TCFB8invperm.req
TDES	TCFB8MMT3.req
TDES	TECBMonte2.req
TDES	TCBCsubtab.req
TDES	TCBCinvperm.req
TDES	TCFB8Monte1.req
TDES	TOFBvartext.req
TDES	TOFBMonte3.req
TDES	TOFBMMT2.req
TDES	TOFBsubtab.req
TDES	TCBCpermop.req
TDES	TCBCMonte1.req
TDES	TOFBvarkey.req
TDES	TOFBMonte2.req
TDES	TECBsubtab.req
TDES	TECBvartext.req
TDES	TECBMMT1.req
TDES	TCFB64Monte3.req
TDES	TCBCvarkey.req
TDES	TCFB64varkey.req
TDES	TCFB64MMT2.req
TDES	TCFB8MMT2.req
TDES	TCFB8varkey.req
TDES	TECBvarkey.req
TDES	TCFB64MMT3.req

TDES	TCBCMMT2.req
TDES	TECBMonte3.req
TDES	TOFBpermop.req
TDES	TCBCMMT1.req
TDES	TCFB8Monte2.req
TDES	TCFB8subtab.req
TDES	TCFB64vartext.req
TDES	TCBCMonte3.req
TDES	TECBMMT3.req
TDES	TCFB64subtab.req
TDES	TCFB8vartext.req
TDES	TECBMMT2.req
TDES	TECBinvperm.req
TDES	TCFB64MMT1.req
TDES	TCFB64permop.req
TDES	TOFBMMT1.req
TDES	TCBCMMT3.req
TDES	TCFB8permop.req
TDES	TCFB64invperm.req
TDES	TCFB8Monte3.req
TDES	TCFB8MMT1.req
TDES	TCBCMonte2.req
TDES	TCFB64Monte1.req
TDES	TOFBMMT3.req
TDES	TOFBMonte1.req
RSA	SigVerPSS.req
RSA	KeyGenRSA.req
RSA	SigGenRSA.req
RSA	SigGenPSS.req
RSA	SigVerRSA.req
RSA	SigVer15.req
RSA	SigGen15.req
SHA	SHA384LongMsg.req
SHA	SHA512LongMsg.req
SHA	SHA224ShortMsg.req
SHA	SHA1ShortMsg.req
SHA	SHA512ShortMsg.req
SHA	SHA224LongMsg.req
SHA	SHA224Monte.req
SHA	SHA1LongMsg.req
SHA	SHA384Monte.req
SHA	SHA512Monte.req
SHA	SHA256Monte.req
SHA	SHA1Monte.req
SHA	SHA384ShortMsg.req
SHA	SHA256LongMsg.req
SHA	SHA256ShortMsg.req
DSA	SigGen.req

DSA	KeyPair.req
DSA	SigVer.req
DSA	PQGGen.req
RNG	ANSI931_AES128VST.req
RNG	ANSI931_AES192VST.req
RNG	ANSI931_AES256MCT.req
RNG	ANSI931_AES192MCT.req
RNG	ANSI931_AES256VST.req
RNG	ANSI931_AES128MCT.req
RSA	SigVerPSS.req (SALT=62)
RSA	SigGenPSS.req (SALT=62)

In order to facilitate the processing of testvector data a series of utilities were developed, culminating in the `mkfipssccr.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 `mkfipsscr.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, rarely as `*.tgz`. A typical pathname structure (for this validation) is as follows:

```
OpenSSL vectors (Linux 32 bit No ASM)
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/HMAC
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/HMAC/resp
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/HMAC/req
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/AES
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/AES/resp
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/AES/req
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/TDES
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/TDES/resp
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/TDES/req
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/RSA
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/RSA/resp
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/RSA/req
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/SHA
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/SHA/resp
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/SHA/req
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/DSA
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/DSA/resp
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/DSA/req
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/RNG
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/RNG/resp
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-0 1.2/RNG/req
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-62 1.2
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-62 1.2/RSA
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-62 1.2/RSA/resp
OpenSSL vectors (Linux 32 bit No ASM)/OpenSSL-x86_ noASM_Linux-SALT-62 1.2/RSA/req
```

Note the typical use of embedded spaces in the directory names. The data files will generally (though not necessarily) be carriage return-line feed delimited.

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

The following `mkfipssrc.pl` options can be used to accommodate various representations of testvector files:

```
--win32                   Generate Windows style script (default is *nix)
--onedir                  All executables are in current directory
--quiet                   Suppress warnings
--dir=<dirname>           Only search named directory
--rspdir=<dirname>        Name of subdirs for *.rsp files (default resp)
--tprefix=<prefix>        Path to test executables
--shwrap_prefix=<prefix>  Path to shlib_wrap.sh (*nix only)
--filter=<string>         Only search pathnames containing <string>
```

B.7 Files for a Runtime Validation

Many vendors validate binary code generated from OpenSSL source. The minimal set of files needed to obtain a validation of the binary shared libraries `libfips.so.0.9.8` (Linux/Unix) or `libosslfips.dll` (Windows) are generated as follows:

For Linux/Unix:

```
./config fipsdso
make
mkdir ../<target_dir>
cp test/fips_*test ../<target_dir>/
cp test/fips_*vs ../<target_dir>/
cp test/fips_test_suite ../<target_dir>/
cp util/shlib_wrap.sh ../<target_dir>/
cp libfips.so.0.9.8 ../<target_dir>/
cp fips/mkfipsscr.pl ../<target_dir>/
```

will place all the files needed for the test lab validation testing in the target directory `../<target_dir>/`. The test lab then runs the binaries in that target directory as follows:

```
cd ../<target_dir>/
unzip <zipfile of test vectors>.zip
perl mkfipsscr.pl --onedir --rspdir=resp
sh fipstest.sh
./shlib_wrap.sh ./fips_test_suite
```

The test vectors are sometimes in tar or tat.gz format, in which case the appropriate command would be used such as `<gunzip -c <tarball of test vectors>.tar.gz | tar xf -.`

For Windows[33]:

```
perl Configure VC-WIN32 fipsdso
nmake -f ms\do_fips
copy test\fips_*test <target_dir>
copy test\fips_*vs <target_dir>
copy test\fips_test_suite <target_dir>
copy libosslfips.dll <target_dir>
copy fipslink.pl <target_dir>
```

The test lab then executes the binaries in that target directory:

```
unzip <zipfile of test vectors>.zip
perl mkfipsscr.pl --win32 --onedir --rspdir=resp
fipstests.bat
fips_test_suite
```

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

Note that the Microsoft Visual C++ Redistributable package (http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=200b2fd9-ae1a-4a14-984d-389c36f85647) may be required to execute these programs on a system that does not have a version of Microsoft Visual C++ installed.

B.8 Retrofitting the fipsalgtest.pl Utility

As of the 0.9.8k OpenSSL baseline, which post-dates the OpenSSL FIPS Object Module v1.2 validation, a new more comprehensive replacement for `mkfipsscr.pl` was developed. Since this utility is external to the cryptographic module it can be used for v1.2 based validations.

Copy `fipsalgtest.pl` from a 0.9.8k or later OpenSSL distribution, and in place of the `mkfipsscr.pl` and `fipstest.sh` scripts referenced earlier in this appendix, simply run

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

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

[33]Note that for the original *openssl-fips-1.2.tar.gz* and some subsequent 0.9.8k+ source distributions the *do_fips.bat* file must be modified to change "f*ipscanisterbuild*" to "*fipsdso*". Such modifications are permissible in the context of a complete new validation.

Subsequently running `fipsalgtest.pl` without the --`generate` option will perform some sanity checks on the previously generated `*.rsp` files. If run on the stock OpenSSL FIPS Object Module v1.2 some DSA errors will be reported:

```
$ perl fipsalgtest.pl --dir=testvectors --generate
Running DSA tests
Don't know how to keyver.
WARNING: error executing verify test KeyPair ../util/shlib_wrap.sh
../test/fips_dssvs keyver <2009-04-16-Sample-all//DSA/rsp/KeyPair.tst
>2009-04-16-Sample-all//DSA/rsp/KeyPair.ver
Running RSA tests
Running SHA tests
Running HMAC tests
Running RAND (AES) tests
Running AES tests
Running Triple DES tests
ALGORITHM TEST VERIFY SUMMARY REPORT:
Tests skipped due to missing files:        0
Algorithm test program execution failures: 0
Test comparisons successful:               172
Test comparisons failed:                   0
Test sanity checks successful:             5
Test sanity checks failed:                 0
Sanity check program execution failures:   1
***TEST FAILURE***
$
```

These errors are seen because some of the sanity checks had not yet been written for v1.2. The `fips_dssvs.c` file from OpenSSL 0.9.8k and later provides those checks.

The `fipsalgtest.pl` utility provides a number of options:

```
--debug                     Enable debug output
--dir=<dirname>             Optional root for *.req file search , default “.”
--filter=<regexp>           Optional regexp prefix for *.req file names
--onedir <dirname>          Assume all components in current directory
--rspdir=<dirname>          Name of subdirectories containing *.rsp files
--shwrap_prefix=<prefix>    Path prefix for the shlib_wrap.sh script,
                            default “../util”.
--tprefix=<prefix>          Prefix for the directory containing the test binaries
--ignore-bogus              Ignore duplicate or bogus files
--ignore-missing            Ignore missing test files
--quiet                     Shhh....
--generate                  Generate algorithm test output
--win32                     Win32 environment (O/S detection is automatic)
```

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

```
dsa
rsa
```

```
rsa-pss0          (disabled by default)
rsa-pss62
sha
hmac
rand-aes
rand-des2         (disabled by default)
aes
aes-cfb1          (disabled by default)
des3
```

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).

Note that as TDES CFB1 and AES CFB1 support has been added to the 0.9.8 branch following 0.9.8k the `fipsalgtest.pl` defaults will change accordingly, so use of recent `fipsalgtest.pl` versions with the original v1.2 code will require that the options `-disable-aes-cfb1` and `--disable-des3-cfb1` be specified.

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:

```
$ ./hmac -v 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
```

The `OPENSSL_FIPS=1` environment variable enables FIPS mode for a `openssl` command generated from a FIPS capable OpenSSL distribution.

Makefile

```
CC = gcc
OPENSSLDIR = /usr/local/ssl
LIBCRYPTO = $(OPENSSLDIR)/lib/libcrypto.a
INCLUDES = -I$(OPENSSLDIR)/include
CMD     = 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[16] = "etaonrishdlcupfm";
    char buf[256];

    /* Generate digest of input stream */
    HMAC_Init(&ctx, key, sizeof(key), EVP_sha1());
    while(fgets(buf,sizeof(buf)-1,fp)) {
        HMAC_Update(&ctx, buf, strlen(buf));
    }
    HMAC_Final(&ctx, hmac_value, &hmac_len);
    HMAC_cleanup(&ctx);

    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,"r")) == NULL) {
                fprintf(stderr,"Unable to open file \"%s\"\n", opt);
                exit(1);
            }
        dofile(fp);
        fclose(fp);
        i++;
        }
    }

    exit(0);
}
```

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

+DES 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.

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

The power-up self-test can take a significant amount of time on slower systems.

HISTORY

FIPS mode support was introduced in version m of OpenSSL.

SEE ALSO

FIPS_mode_set(7)

D.2 FIPS_mode_set(), FIPS_selftest()

NAME

FIPS_mode_set, FIPS_selftest - perform FIPS power-up self-test

SYNOPSIS

```
#include <openssl/fips/fips.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.

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 code of 1 indicates success, 0 failure.

SEE ALSO

FIPS_mode(7)

HISTORY

FIPS support was introduced in version m of OpenSSL.

D.3 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
$
```

The `FIPS_mode_set()` call or other function calls in FIPS mode can return any of the following errors:

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 in this appendix will be removed from upcoming revisions of this document and relocated elsewhere. That new location will be at, or linked from, http://www.openssl.org/docs/fips/.

E.0 Nomenclature for ABIs:

Among other things an ABI is characterized by implicit (or default) integer, long integer, and pointer sizes with the letters I, L, P:

- I stands for sizeof(int)
- L stands for sizeof(long)
- P stands for size(void*), i.e. the size of a pointer

For example, HP-UX for Itanium/IA64 implements two separate ABIs, 32 bit “ILP32” and 64 bit “I32/LP64”. In the first case sizeof(int)=sizeof(long)=sizeof(void*)=4 or 32 bits, while for the latter sizeof(int)=4 or 32 bits and sizeof(long)=sizeof(void*)=8 or 64 bits.

Considerations for Linux:

Due to the diversity of hardware platforms supported by Linux no universal guidance can be given. For Linux on x86_64, 32 bit code generation could be performed by installing 32-bit Linux x86 on an alternative partition or booting it off a "live CD" (or rather "live DVD" as there is insufficient room for a compiler on a CD). For other 64 bit platforms the generation of 32 bit code is more of a challenge, most likely requiring access to old hardware on a corresponding 32-bit Linux can be booted. However, while being perfectly capable of running corresponding legacy 32-bit ***user-land*** code, most 64-bit non-x86_64 hardware can't run corresponding 32-bit kernels (or rather those 32-bit kernelsl not won't know how to handle such CPU). Note that there are some "pure" 64-bit Linux implementation which have no 32-bit legacy counterpart, such as Alpha and Itanium.

Considerations for HP-UX:

HP-UX PA-RISC: It is not known if there are any HP-UX systems that capable of booting either 32- or 64-bit kernels. Presumably old hardware would be needed to compile 32-bit PA-RISC binaries.

HP-UX Itanium: There is only a 64-bit kernel and no way to compile ILP32 Itanium code. Note incidentally a subtle point, there is no such thing as legacy 32-bit Itanium code. The ILP32 Itanium ABI was introduced to facilitate *porting* or re-compilation of non-64-bit safe applications from *other* 32-bit platforms, a backward compatibility of sorts that is really a source code compatibility.

This is apart from the fact that HP-UX Itanium can run legacy 32-bit PA-RISC binaries via emulation.

Considerations for Windows:

Microsoft Windows Win64 only implements “IL32/P64” where sizeof(int)=sizeof(long)=32 bits and sizeof(void*)=64 bits. For Win64 “non 64 bit safe” code is code that is compiled for Win64 but which treats pointers as if they were 32 bit values, and the Win64 kernel can be instructed to execute any particular .exe to a 4Gb address space (the size of a 32 bit pointer).

The FIPS Object Module built on Win64 can be used with either “64 bit safe” or “non-64 bit safe” application code.

Microsoft Visual Studio C++ 2005 comes with a number of different compilers (cl.exe programs). Depending on the install options and target system types several different tool chains may be present. For example, on a Win32/X86 system the following may be present:

1. Win32/x86 cl.exe generating x86 code
2. Win32/x86 cl.exe generating x86_64(x64) code
3. Win32/x86 cl.exe generating IA64 code
4. Win32/x86 cl.exe generating ARM code (for Mobile PC)

A Win64/x86_64 (aka “x64” in Windows-speak) system will have all of the above plus

5. Win64/x64 cl.exe generating x86_64(x64) code

The Platform SDK and DDk for Win64 also adds one more compiler,

6. Win64/IA64 cl.exe generating IA64 code

Only the first four will be present on a Win32/x86 system. Note that there are two different compilers generating x86_64 code. On either Win32 or Win64 at least three of these will be cross-compilers with no option to run code on the same system and hence cannot be used for building the FIPS Object Module. Note it doesn't really matter which compiler is used as long as the resulting binary code can be executed. For example, on Win64 x86_64 systems either the Win32/x86 cl.exe generating x86_86 code (which can be qualified as cross-compiler) or Win64/x64 cl.exe may be used. Likewise for Itanium, either the Win32/x86 cl.exe generating IA64 code (which can be qualified as a cross-compiler) or Win64/IA64 cl.exe may be used. The only difference in this case would be compilation time as the because Win32/x86 compiler would be slower.

To build Win32/x86 compatible code on a Win64 system start

```
%SYSTEMROOT%\syswow64\cmd.exe
```

and make sure a `cl.exe` generating x86 code is in `%PATH%`.

Most people associate the term “backward compatibility” with backward *binary* compatibility, as when a 64-bit system is capable of running 32-bit binary code (for the same OS) without recompilation. For example, a x86_64 system can run x86 binary code, a sparcv9 system can run sparcv8 binary code, etc. Win64 can do this, i.e. both Win64 x86_64 and Win64 Itanium can run Windows x86 binaries. But Win64 can do something else as well; the Win64 kernel can provide a run-time environment for *re-compiled* non-64-bit safe applications. Thus backwards compatibility in Win64 is two-fold: binary and "source."

Note that backward "source" compatibility in Win64 (on both x86_64 and Itanium) is nothing like ILP32 in HP-UX Itanium. In HP-UX Itanium ILP32 applications have a distinct binary format, require separate sets of run-time libraries such as libc.so, and use dedicated system call interfaces. The Win64 kernel on the other hand can simply can be instructed to restrict a the address space for a particular application to 2GB. That restriction makes it possible for that application to safely cast pointers to 32-bit integer and back, a characteristic of non-64-bit safe programs. But this is done with no dedicated ILP32 ABI, no special ntdll.dll, nothing but a single bit in .exe header, which can be set in the link stage with /LARGEADDRESSAWARE:NO (and even manipulated subsequently with editbin). This capability means that the same .dll can be used in both 64-bit safe and non-64-bit safe applications.

E.1 Compiler placement of read-only data

The in-core hashing mechanism requires that read-only data be placed in a read-only data segment. The `FIPS_set_mode()` function is designed to detect situations where this requirement is not met. One example of this problem is on 32 bit[34] HP-UX PA-RISC when using `gcc` to generate position independent code (-FPIC). At least some versions of `gcc` do not discriminate read-only data and put it into a writable data segment. This problem has been observed with gcc 3.2.3 and 3.4.2. This placement effectively voids the embedded digest value and the verification procedure is bound to fail.

A simple test program will demonstrate the problem. On a 32 bit HP-UX PA-RISC system using `gcc` the commands

```
echo "const int i=0;" > a.c
gcc -c a.c
size a.o
gcc -c -fPIC a.c
size a.o
gcc -v
```

[34]Note this problem occurs for 32 bit code whether executed on a 32 or 64 bit processor. 64 bit PA-RISC code has not been observed to exhibit this problem.

generate the output

```
8 + 0 + 0 = 8
0 + 8 + 0 = 8
Reading specs from /usr/local/lib/gcc-lib/hppa2.0n-hp-hpux11.00/3.2.3/specs
Configured with: ../gcc-3.2.3/configure --enable-languages=c,c++ --with-gnu-as
Thread model: single
gcc version 3.2.3
```

On the same 32 bit HP-UX PA-RISC system using the HP C compiler the commands:

```
echo "const int i=0;" > a.c
cc -Ae -c a.c
cc -Ae +Z -c a.c
cc -V -Ae -c a.c
```

give the result

```
4 + 0 + 0 = 4
4 + 0 + 0 = 4
cpp.ansi: HP92453-01 B.11.11.32871.GP HP C Preprocessor (ANSI)
ccom: HP92453-01 B.11.X.34412-34415.GP HP C Compiler
```

The workaround for this bug is to advise users to use the HP C compiler for their 32-bit HP-UX PA-RISC applications until/if `gcc` is fixed.

E.2 Bugs in Microsoft TLS Implementation

In FIPS mode the number of available ciphersuites is restricted to those using 3DES, AES, SHA-1 and SHA-2, resulting in negotiation of AES and 3DES in CBC mode with a TLS client. At least some versions of the Microsoft SSL/TLS implementation of CBC are unable to handle the empty fragments inserted in CBC mode by OpenSSL as a countermeasure for a minor security issue.

A discussion of this security issue can be found at http://www.openssl.org/~bodo/tls-cbc.txt. The general consensus of the OpenSSL developers who implemented the FIPS mode is that this vulnerability is relatively minor.

This use of empty fragments can be disabled with the function call

```
SSL_CTX_set_options(ctx,SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS)
```

Note the rudimentary https server provided by the (FIPS compatible) *openssl* command can be used to test for this bug:

```
OPENSSL_FIPS=1 openssl s_server -www [-debug]
```

Note that TLS is disabled by default for the Microsoft Internet Explorer (go to “Tools” menu, select “Internet Options...”, select “Advanced”, scroll top the bottom and enable “Use TLS 1.0”).

E.3 Solaris and gcc Problems

There is a known problem with some versions of gcc[35] on x86 Solaris where a program linked with the OpenSSL libcrypto will segfault in "`_init`".

Every object module consists of segment fractions which are merged by the link-editer. For example, `.text` fractions from all `*.o` files are concatenated to form the program `.text` segment. Every such fraction is allowed to specify alignment in the resulting image and the link-editer is expected to align it accordingly. When different fractions specify different alignments the link-editer pads the previous fragment until the alignment requirement for the currently processed fragment is met. The value used for padding varies by implementation. Solaris `ld` uses a value of zero, which means that for executable segments machine code is padded with zeros. That means that if the processor attempts to interpret that zero padding (opcode 0), it would execute a series of '`add %al,(%eax)`' instructions. Depending on the value in register `%eax` that instruction might work or it might incur a segmentation violation. But, even if the instruction does not segfault it can cause unpredicted behavior later on. A better choice of padding value would be one which maps into an instruction which has no effect on the processor state. In x86 context 0x90 (the NOP machine instruction) would be an appropriate value.

Why is only the `.init` segment is affected and not `.text`? Because `.text` fragments contain either complete functions and padding zeros are preceded by flow transfer instructions, such as return or branch), or code fragments that never complete (e.g. calling "exit this process" system call). In other words padding is never executed in `.text` segments. The `.init` fragments on the other hand contain pure code and are concatenated to form a linear stream of machine code. The padding values are therefore executed as machine instructions and non-NOP padding is bound to have undesired side effects such as segmentation violation.

Andy Polyakov has prepared a patch of sorts to address this issue (http://www.openssl.org/~appro/values.c). This “patch”, a single file which is both a shell script and a C source file, modifies the Solaris gcc development environment only, not the OpenSSL code. On Solaris linking with `libc` requires linking with an object module which instantiates the `_lib_version` constant. This object module is commonly provided by Sun and is linked first prior to `crtbegin.o`. The patch procedure strategically places a replacement module[36] on the `gcc` library search path. But in addition to `_lib_version` the replacement module contains an `.init`

[35] And with the Sun C compiler also, but OpenSSL currently does not use assembler optimizations with that compiler.
[36] The `values.c` patch actually installs several files with names of the form `values-X?.o`. Which of these files is referenced depended on the compiler options. For example, “`cc -Xc ...`” references `values-Xc.o`, “, “`cc -Xt ...`” references `values-Xt.o`, and `gcc` references `values-Xc.o` if `-ansi` is specified and `values-Xa.o` otherwise.

snippet, which simply checks if the value following the snippet code is zero (i.e., the troublesome padding) or not and conditionally skips over or executes it.

The alignment of this replacement module .init fragment and the OpenSSL .init fragments is chosen carefully to work with both older and newer versions of `gcc`. If OpenSSL .init fragments were not aligned the way they are, then padding *following* the last one would cause the problem with older versions of `gcc`. Lack of replacement module padding *preceding* the first OpenSSL .init fragment would cause the problem with newer versions of gcc. The patch addresses both older and newer gcc versions.

Note the end-user does not need to patch anything to run the linked application. Replacement modules are linked statically (just like the original ones) into either application or shared object code.

The gcc installation modifications can be reverted by removing the files "`values-X*.o`" from the directory containing `libgcc.a`. The location of that directory can be obtained by executing "`gcc -print-libgcc-file-name`".

E.5 HP-UX Vendor Support

Pre-compiled, pre-packaged supported versions of OpenSSL, the OpenSSL FIPS Object Module, and the `prngd` random number generator for HP-UX are available at http://h20293.www2.hp.com/portal/swdepot/displayProductInfo.do?productNumber=OPENSSL11I.

HP was a sponsor of the original OpenSSL FIPS Object Module validation.

E.6 Apple OS X Support

The build environment can be specified by the auxiliary files:

```
setenv-reset.sh
setenv-darwin-i386.sh              (for 32 bit)
setenv-darwin-x86_64.sh            (for 64 bit)
```

These files can be obtained from:

http://openssl.com/fips/1.2/platforms/osx/

Note the instructions are very similar for the two platforms (32 and 64 bit), the only essential difference being the use of the appropriate environment settings.

First we set the environment variables to define the target platform in the appropriate workarea containing the source files:

```
$ cd  openssl-fips-1.2.4
$ .  ../setenv-reset.sh            (note leading dot ".")
$ .  ../setenv-darwin-i386.sh      (for 32 bit)
$ .  ../setenv-darwin-x86_64.sh    (for 64 bit)
```

At this point we are ready to commence the standard FIPS canister build for the target platform.

```
$ ./config  fipscanisterbuild
should see several screens of output
$ make
should see lots of output
$ cd  ..
```

The architecture type can be confirmed with the `lipo` command:

```
$ lipo  -info  openssl-fips-1.2.4/test/fips_algvs
Non-fat file: openssl-fips-1.2.4/test/fips_algvs is
architecture: i386                 (for 32 bit)
Non-fat file: openssl-fips-1.2.4/test/fips_algvs is
architecture: x86_64               (for 64 bit)
$
```

Note the `make` command automatically generates the `fips_algvs` utility, no separate `make build_algvs` command is necessary.

The `fips_algvs` utility can now be invoked:

```
$ openssl-fips-1.2.4/test/fips_algvs  fips_test_suite
$
     FIPS-mode test application

1. Non-Approved cryptographic operation test...
     a. Included algorithm (D-H)...successful
2. Automatic power-up self test...successful
3. AES encryption/decryption...successful
     .
     .
     .
     char buffer key after overwriting:
          ccfe6b7c02eb009d3157a67a4c2fe4e5
```

```
        successful as expected

    All tests completed with 0 errors
    $
```

E.7 Apple iOS Support

The build environment requires the auxiliary files:

```
    setenv-reset.sh
    setenv-darwin-i386.sh
    setenv-ios-10.sh                    (for use with OS X 10)
    setenv-ios-11.sh                    (for use with OS X 11)
    ios-incore-1.2.4.tar.gz
```

These files can be obtained from:

http://openssl.com/fips/1.2/platforms/ios/

The `ios-incore.tar.gz` file contains an OS X and iOS specific "incore" utility to determine the object code digest.

The `ios-project.tar.gz` file contains OS X and iOS specific files that define Xcode "project" information needed to create the test suite application for the iOS platform.

Setup

On the OS X build system:

```
    $ gunzip -c openssl-fips-1.2.4.tar.gz | tar xf -
```

As the integrity check requires calculation of a fingerprint and the Darwin host cannot directly execute the target images, we use the cross-compilation approach of calculating the incore image on the build host using a separate "incore" utility.

To compile the "incore" utility.

```
    $ . setenv-reset.sh                 (note the leading dot ".")
    $ . setenv-darwin-i386.sh           (note the leading dot ".")
    $ cd openssl-fips-1.2.4
    $ gunzip -c ../ios-incore-1.2.4.tar.gz | tar xf -
    $ ./config fipscanisterbuild
    should see several screens of output
    $ make
    should see lots of output
```

```
$ cd  iOS
should see several lines of output
$ make
```

The output of the build process is a utility *incore_macho* that should be installed in a location in the PATH (this is done automatically for this build process). Note this utility is a native binary that executes on the Darwin host.

Confirm the utility works:

```
$ ./incore_macho
usage:
        ./incore_macho [--debug] [-exe|-dso] executable
$
```

Then cleanup the build which was just done in order to construct the incore utility – it is no longer needed once incore_macho has been built.

```
$ cd   ..
$ make   clean          (remove host build except for iOS directory)
$ rm   -f   *.dylib     (remove left over dynamic libraries)
```

This instructions from this point assume the build environment has been prepared, including the creation of the "incode_macho" utility, as documented in the previous section.

Cross-compilation

This step can be performed either via a SSH command line session or in a terminal window in a graphical session (e.g. VNC).

First we set the environment variables to define the target platform, and confirm the pathnames are correct in the first window. Note that while these commands look similar to those recently executed for the generation of the incore utility, there are some subtle differences. This time around we are cross-compiling binaries for the iOS target device:

```
$ cd   openssl-fips-1.2.4
$ .   ../setenv-reset.sh          (note the leading dot ".")
$ .   ../setenv-ios-NN.sh         (note the leading dot ".")
$ llvm-gcc    -v                  (confirm we have the PATH right)
Target: i686-apple-darwin10
Using built-in specs.
Target: i686-apple-darwin11
Configured with: /private/var/tmp/llvmgcc42/llvmgcc42-
2336.1~1/src/configure --disable-checking --enable-werror
```

```
--prefix=/Developer/usr/llvm-gcc-4.2 --mandir=/share/man
--enable-languages=c,objc,c++,obj-c++ --program-prefix=llvm-
--program-transform-name=/^[cg][^.-]*$/s/$/-4.2/ --with-
slibdir=/usr/lib --build=i686-apple-darwin11 --enable-
llvm=/private/var/tmp/llvmgcc42/llvmgcc42-2336.1~1/dst-
llvmCore/Developer/usr/local --program-prefix=i686-apple-
darwin11- --host=x86_64-apple-darwin11 --target=i686-apple-
darwin11 --with-gxx-include-dir=/usr/include/c++/4.2.1
Thread model: posix
gcc version 4.2.1 (Based on Apple Inc. build 5658) (LLVM
build 2336.1.00)
$
```

At this point we are ready to commence the standard FIPS canister build for the target platform.

```
$ ./config   fipscanisterbuild
should see several screens of output
$ make
should see lots of output
$ make   install
should see lots of output
```

The output will be installed into the `/usr/local/ssl/Release-iphoneos` area (the installation location can be adjusted in the `setenv-ios-`*`NN`*`.sh` script).

Note we can confirm that these binaries are for the iOS target device:

```
$ lipo  -info  /usr/local/ssl/Release-iphoneos/lib/fipscanister.o
Non-fat file: /usr/local/ssl/Release-
iphoneos/lib/fipscanister.o is architecture: armv7
$
```

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: Veridical Systems, Inc.
POINT OF CONTACT: Steve Marquess
PHONE and/or FAX: 301-831-8447
MANUFACTURER: N/A
PRODUCT NAME/MODEL #: OpenSSL
ECCN: 5D002

NOTIFICATION: http://cvs.openssl.org/dir

Employee(s) of Veridical Systems, Inc. are participating in the
development of the freely available open source OpenSSL product by
providing feedback on new releases, by requesting new features, and by
correspondence either to the developer and user mailing lists or
directly with the core developers.   This correspondence may include
suggested source code fragments or patches.  All versions of any such
contributions incorporated in any of the OpenSSL software will be
publicly accessible at http://cvs.openssl.org/dir.
```

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/140sp1051.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.

Appendix A

The entry "x84-64" in the last row of the table should be "x86-64". The sentence "Verify that the SHA-1 HMAC digest ..." in Installation Instructions should read "Verify the SHA-1 HMAC digest...". Reported by Thomas Hruska of Shining Light Productions.

Appendix B

The link in Appendix B on page 16 of the original published OpenSSL FIPS 140-2 Security Policy Version 1.2 (dated August 8, 2008) shows as http://www.openssl.org/source/openssl-fips-1.2.tar.gz but incorrectly links to http://www.openssl.org/source/OpenSSL-fips-1.0.tar.gz. Reported by Henry P. Unger of Hitech Systems, Inc. This error has been corrected in the subsequent Security Policy dated November 3, 2009.