wolfSSL Inc.

wolfCrypt

FIPS 140-2 Cryptographic Module Non-Proprietary Security Policy

Version: 2.16

Date: October 8, 2018

Table of Contents

List of Tables and Figures

1 Introduction

This document defines the Security Policy for the wolfSSL Inc. wolfCrypt (Software Versions 3.6.0, 3.6.1, 3.6.6, 3.11.2, 3.12.2, 3.12.4, 3.12.6, and 3.14.2) module, hereafter denoted the Module. The Module is a cryptography software library. The Module meets FIPS 140-2 overall Level 1 requirements.

The Module is intended for use by US Federal agencies and other markets that require FIPS 140-2 validated cryptographic functionality. The Module is a software-only module, multi-chip standalone module embodiment; the cryptographic boundary is the collection of object files from the source code files listed in Table 16 – Source Files. No software components have been excluded from the FIPS 140-2 requirements.

Operational testing was performed for the following Operating Environments:

Table 1 – Tested Operating Environments

	Operating System	Processor	Platform
1	Linux 3.13 (Ubuntu)	Intel® Core™ i7-3720QM CPU @2.60GHz x 8	HP EliteBook
2	iOS 8.1	Apple™ A8	iPhone™ 6
3	Android 4.4	Qualcomm Krait 400	Samsung Galaxy S5
4	FreeRTOS 7.6	ST Micro STM32F	uTrust TS Reader
5	Windows 7 (64-bit)	Intel® Core™ i5	Sony Vaio Pro
6	Linux 3.0 (SLES 11 SP4, 64-bit)	Intel® Xeon® E3-1225	Imprivata OneSign
7	Linux 3.0 (SLES 11 SP4, 64-bit) on Microsoft Hyper-V 2012R2 Core	Intel® Xeon® E5-2640	Dell® PowerEdge™ r630
8	Linux 3.0 (SLES 11 SP4, 64-bit) on VMWare ESXi 5.5.0	Intel® Xeon® E5-2640	Dell® PowerEdge™ r630
9	Windows 7 (64-bit) on VMWare ESXi 5.5.0	Intel® Xeon® E5-2640	Dell® PowerEdge™ r630
10	Android Dalvik 4.2.2	NXP i.MX6	MXT-700-NC 7” touch panel
11	Linux 4.1.15	NXP i.MX5	NX-1200 NetLinx NX Integrated Controller
12	Debian 8.8	Intel Xeon® 1275v3	CA PAM 304L Server
13	Windows Server 2012R2	Intel® Xeon® E5335	CA Technologies PAMHAF995
14	Windows 7 Professional SP1	Intel® Core™ i7-2640M	Dell™ Latitude™ E6520
15	Debian 8.7.0	Intel ® Xeon® E3 Family with SGX support	Intel® x64 Server System R1304SP

16	Windows 10 Pro	Intel ® Core ™ i5 with SGX support	Dell™ Latitude™ 7480
17	NET+OS v7.6	Digi International NS9210	Sigma IV infusion pump
18	Linux 4.4 (SLES 12 SP3, 64-bit) on Microsoft Hyper-V 2016 Core	Intel® Xeon® E5-2650	Dell® PowerEdge™ r720
19	Linux 4.4 (SLES 12 SP3, 64-bit) on VMWare ESXi 6.5.0	Intel® Xeon® E5-2403	Dell® PowerEdge™ r420

The FIPS 140-2 security levels for the Module are as follows:

Table 2 - Security Level of Security Requirements

Security Requirement	Security Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

1.1 Hardware and Physical Cryptographic Boundary

The physical cryptographic boundary is the general purpose computer where the Module is installed. The Module relies on the computer system where it is running for input/output devices.

Table 3 – Ports and Interfaces

Description	Logical Interface Type
API entry point	Control in
API function parameters	Data in
API return value	Status out
API function parameters	Data out

1.2 Software and Logical Cryptographic Boundary

Figure 1 depicts the Module operational environment.

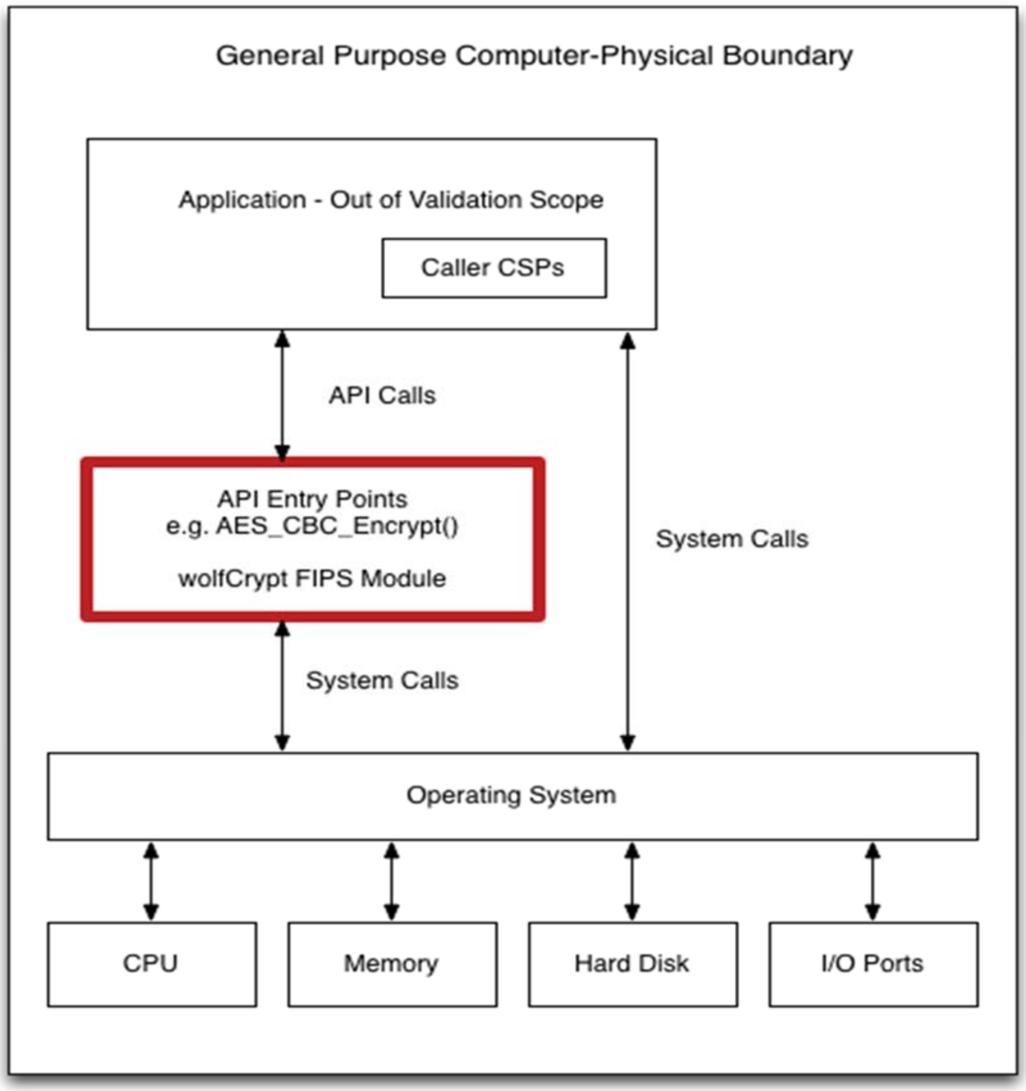

Figure 1 – Module Block Diagram

The above diagram shows the Logical Boundary highlighted in red contained within the Physical Boundary. The Logical Boundary contains all FIPS API entry points. The Logical Boundary is invoked by the Application through the API Calls.

1.3 Modes of Operation

The Module supports a FIPS Approved mode of operation and a non-FIPS Approved mode of operation. FIPS Approved algorithms are listed in Table 4. Non-FIPS Approved but allowed algorithms are listed in Table 5. The module is in the Approved mode of operation when any of the cryptographic functions listed in Table 4 and Table 5 are invoked by the calling application.

The Module is in the non-FIPS Approved mode of operation when any of the non-Approved cryptographic functions are invoked by the calling application (not recommended for applications requiring a FIPS 140-2 validated module). Critical Security Parameters (CSPs) are not shared between the FIPS Approved mode of operation and the non-FIPS Approved mode of operation.

For installation instructions, see Appendix A – Installation Instructions.

The conditions for using the module in an Approved mode of operation are:

1. The module is a cryptographic library and it is intended to be used with a calling application. The calling application is responsible for the usage of the primitives in the correct sequence.
2. The module relies on an entropy source external to the module boundary. The module contains an Approved DRBG which generates random strings whose strengths are modified by available entropy.
3. The keys used by the module for cryptographic purposes are determined by the calling application. The calling application is required to provide keys in accordance with FIPS 140-2 requirements.

2 Cryptographic Functionality

The Module implements the FIPS Approved and Non-Approved but Allowed cryptographic functions listed in the tables below.

Table 4 – Approved and CAVP Validated Cryptographic Functions

Algorithm	Description	Cert #
AES	[FIPS 197, SP 800-38A] Functions: Encryption, Decryption Modes: CBC, CTR Key sizes: 128, 192, 256 bits	3157 3330 3417 3490 3508 4635 4643 4772 5244 5325 5447

Algorithm	Description	Cert #
DRBG	[SP 800-90A]	650
	Functions: Hash DRBG	775
	Security Strengths: 256 bits	821
		863
		875
		1561
		1566
		1651
		2006
		2055
		2132
HMAC	[FIPS 198-1]	1990
	Functions: Generation, Verification	2121
	SHA sizes: SHA-1, SHA-256, SHA-384, and SHA-512	2175
		2228
		2241
		3068
		3075
		3183
		3471
		3523
		3605
RSA	[FIPS 186-4, and PKCS #1 v2.1 (PKCS1.5)]	1602
	Functions: Signature Generation, Signature Verification	1710
	Key sizes: 1024 (verification only), 2048	1749
		1791
		1803
		2530
		2534
		2612
		2804
		2853
		2923

Algorithm	Description	Cert #
SHA	[FIPS 180-4] Functions: Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications SHA sizes: SHA-1, SHA-256, SHA-384, SHA-512	2614 2763 2823 2882 2893 3799 3806 3915 4222 4277 4366
Triple-DES (TDES)	[SP 800-20] Functions: Encryption, Decryption Modes: TCBC Key sizes: 3-key	1800 1901 1928 1966 1972 2465 2470 2535 2652 2687 2737

Table 5 – Non-Approved but Allowed Cryptographic Functions

Algorithm	Description
RSA Primitives and Operations	[IG D.9] Per IG D.9, RSA is an allowed method for supporting key transport in an Approved FIPS mode of operation. RSA may be used by a calling application as part of a key encapsulation scheme. No keys are established into the module using RSA. Key sizes: 2048 bits When used for system level key establishment this service provides 112 bits of security.

Algorithm	Description
Non-SP 800-56A Compliant DH Primitive	[IG D.8] Per IG D.8, Scenario 6 – non-Approved (not compliant with SP 800-56A) primitive only, a partial DH key agreement scheme is allowed in an Approved FIPS mode of operation. No keys are established into the module using DH. Key agreement; key establishment methodology provides 112 bits of encryption strength.
Non-SP 800-56A Compliant ECDH Primitive	[IG D.8] Per IG D.8, Scenario 6 – non-Approved (not compliant with SP 800-56A) primitive only, a partial ECDH key agreement scheme is allowed in an Approved FIPS mode of operation. No keys are established into the module using ECDH. Key agreement; key establishment methodology provides 256 bits of encryption strength.
MD5 for use within TLS	[IG D.2] MD5 is allowed in an Approved mode of operation when used as part of an approved key transport scheme (e.g. SSL v3.1) where no security is provided by the algorithm.

Non-Approved Cryptographic Functions for use in non-FIPS mode only:

- AES GCM (non-compliant)
- RSA Signature Generation with 1024 bit key
- DES
- MD5
- RC4
- RIPEMD-160
- HMAC-MD5

2.1 Critical Security Parameters

All CSPs used by the Module are described in this section. All usage of these CSPs by the Module (including all CSP lifecycle states) is described in the services detailed in Section 4. The CSP names correspond to the API parameter inputs.

Table 6 – Critical Security Parameters (CSPs)

CSP	Description / Usage
Hash_DRBG	Entropy input V (440) and C (440)
HMAC Key	Keyed Hash key
AES EDK	AES (128/192/256) encrypt/decrypt key
TDES EDK	TDES (3-Key) encrypt/decrypt key
RSA KDK	Private component of an RSA key pair (2048bit), used by RSA key establishment
RSA SGK	Private component of an RSA key pair (2048bit), used by RSA signature generation
DH Private	Private Key Agreement Key

2.2 Public Keys

Table 7 – Public Keys

Key	Description / Usage
RSA KEK	Public component of an RSA key pair (2048bit), used by RSA key establishment
RSA VK	Public component for an RSA key pair (2048bit), used by RSA signature verification
DH Public	Public Key Agreement Key

3 Roles, Services, and Authentication

3.1 Assumption of Roles

The Module supports two distinct operator roles, User and Cryptographic Officer (CO). The cryptographic module does not provide an authentication or identification method of its own. The CO and the User roles are implicitly identified by the service requested.

Table 8 lists all operator roles supported by the Module. The Module does not support a maintenance role or bypass capability.

Table 8 – Roles Description

Role ID	Role Description	Authentication Type	Authentication Data
CO	The Cryptographic Officer Role is assigned the Zeroize service.	None	None
User	The User Role is assigned all services except Zeroize.	None	None

3.2 Services

All services implemented by the Module are listed in the tables below with a description of service CSP access. The calling application may use the wolfCrypt_GetStatus_fips() API to determine the current status of the Module. A return code of 0 means the Module is in a state without errors. Any other return code is the specific error state of the module.

Table 9 – Authorized Services available in FIPS mode

Service	Description	Role
Module Reset (Self-test)	Reset the Module by restarting the application calling the Module. Does not access CSPs.	User
Show status	Functions that give module status feedback. Does not access CSPs.	User
Zeroize	Functions that destroy CSPs. FreeRng_fips destroys RNG CSPs. All other services automatically overwrite memory bound CSPs. Cleanup of the stack is the duty of the application. Restarting the general purpose computer clears all CSPs in RAM.	CO
Random number generation	Uses the SP 800-90A DRBG for random number generation. This service is not used by the module to generate keys for the module's use. It merely outputs random numbers per the calling application's request.	User
Symmetric encrypt/decrypt	Used to encrypt and decrypt data using AES EDK and TDES EDK. CSPs passed in by the application	User

Service	Description	Role
Message digest	Used to generate a SHA-1 or SHA-2 message digest. MD5 used only to support TLS 1.1 and lower. Does not access CSPs.	User
Keyed hash	Used to generate or verify data integrity with HMAC. The HMAC Key is passed in by the application.	User
Key transport	Used to encrypt or decrypt a key value on behalf of the application. RSA KDK and RSA KEK are passed in by the calling application. When decrypting a key value, a symmetric key is output to the calling application.	User
Key agreement	Used for DH key agreement on behalf of the application. The DH keys are passed in by the calling application. A symmetric key is output to the calling application.	User
Digital signature	Used to generate or verify RSA digital signatures. RSA SGK and RSA VK are passing in by the calling application.	User

Table 10 – Services available in non-FIPS mode

Service	Description
AES GCM	Used to encrypt and decrypt data using AES GCM
Message digest MD5	MD5 message digest not an approved FIPS cryptographic function.
DES	Single DES symmetric encrypt/decrypt not an approved FIPS cryptographic function.
RC4	RC4 symmetric encrypt/decrypt not an approved FIPS cryptographic function.
HMAC MD5	Keyed hash using MD5 is not an approved FIPS cryptographic function.
Message digest RIPEMD-160	RIPEMD-160 digest not an approved FIPS cryptographic function.
Digital Signature	Used to generate RSA 1024-bit digital signatures. RSA SGK and RSA VK are passed in by the calling application.

See Chapter 10: wolfCrypt Usage Reference in the wolfSSL Manual for additional information on the cryptographic services listed in this section.

Table 11 – CSP Access Rights within Services, defines the relationship between access to CSPs and the different module services. The modes of access shown in the table are defined as:

- R = Read: The module reads the CSP. The read access is typically performed before the module uses the CSP.
- E = Execute: The module executes using the CSP.
- Z = Zeroize: The module zeroizes the CSP.

Table 11 – CSP Access Rights within Services

Service	CSPs						
	Hash_DRBG	HMAC Key	AES EDK	TDES EDK	RSA KDK	RSA SGK	DH Private
Module Reset (Self-test)	-	-	-	-	-	-	-
Show Status	-	-	-	-	-	-	-
Zeroize	Z	Z	Z	Z	Z	Z	Z
Random number generation	R,E	-	-	-	-	-	-
Symmetric encrypt/decrypt	-	-	R,E,Z	R,E,Z	-	-	-
Message digest	-	-	-	-	-	-	-
Keyed hash	-	R,E,Z	-	-	-	-	-
Key transport	-	-	-	-	R,E,Z	-	-
Key agreement	-	-	-	-	-	-	R,E,Z
Digital signature	-	-	-	-	-	R,E,Z	-

4 Self-tests

Each time the Module is powered up it tests that the cryptographic algorithms still operate correctly and that sensitive data have not been damaged. The Module provides a default entry point to automatically run the power on self-tests compliant with IG 9.10. Power on self–tests are available on demand by reloading the Module.

On power-on or reset, the Module performs the self-tests described in Table 12. All KATs must complete successfully prior to any other use of cryptography by the Module. If one of the KATs fails, the Module enters the self-test failure error state. To recover from an error state, reload the Module into memory.

During the FIPS 140-2 validation testing process, InfoGard Laboratories verified that the HASH DRBG implements the required Health Testing described in SP 800-90A Section 11.3. InfoGard Laboratories is accredited by the National Voluntary Laboratory Accreditation Program (NVLAP) to perform cryptographic testing under Lab Code 100432-0.

Table 12 – Power-on Self-tests

Test Target	Description
Software Integrity	HMAC-SHA-256
AES	KATs: Encryption, Decryption Modes: CBC Key sizes: 128 bits
DRBG	KATs: HASH DRBG Security Strengths: 256 bits
HMAC	KATs SHA sizes: SHA-1, SHA-512
RSA	KATs: Signature Generation, Signature Verification Key sizes: 2048 bits
TDES	KATs: Encryption, Decryption Modes: TCBC, Key sizes: 3-key

Table 13 – Conditional Self-tests

Test Target	Description
DRBG	DRBG Continuous Test performed when a random value is requested from the DRBG.

5 Physical Security

The FIPS 140-2 Area 5 Physical Security requirements do not apply because the Module is a software module.

6 Operational Environment

The tested environments place user processes into segregated spaces. A process is logically removed from all other processes by the hardware and Operating System. Since the Module exists inside the process space of the application this environment implicitly satisfies requirement for a single user mode.

7 Mitigation of Other Attacks Policy

The Module is not intended to mitigate against attacks that are outside the scope of FIPS 140-2.

8 Security Rules and Guidance

The Module design corresponds to the Module security rules. This section documents the security rules enforced by the cryptographic module to implement the security requirements of this FIPS 140-2 Level 1 module.

1. The Module provides two distinct operator roles: User and Cryptographic Officer.
2. Power-on self-tests do not require any operator action.
3. Data output is inhibited during self-tests, zeroization, and error states.
4. Status information does not contain CSPs or sensitive data that if misused could lead to a compromise of the Module.
5. There are no restrictions on which keys or CSPs are zeroized by the zeroization service.
6. The calling application is the single operator of the Module.
7. The Module does not support manual key entry.
8. The Module does not have any external input/output devices used for entry/output of data.
9. The module does not support key generation.

9 References and Definitions

The following standards are referred to in this Security Policy.

Table 14 – References

Abbreviation	Full Specification Name
[FIPS140-2]	*Security Requirements for Cryptographic Modules*, May 25, 2001
[SP800-131A]	*Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*, January 2011

Table 15 – Acronyms and Definitions

Acronym	Definition
AES	Advanced Encryption Standard
API	Application Programming Interface
CO	Cryptographic Officer
CSP	Critical Security Parameter
DES	Data Encryption Standard
DH	Diffie-Hellman
DRBG	Deterministic Random Bit Generator
ECDH	Elliptic Curve Diffie-Hellman
FIPS	Federal Information Processing Standard
HMAC	Keyed-Hash Message Authentication Code
RSA	Rivest, Shamir, and Adleman Algorithm
SSL	Secure Sockets Layer
TDES	Triple-DES
TLS	Transport Layer Security
SHA	Secure Hash Algorithm

The source code files in Table 16 create the object files of the wolfCrypt module on each supported operating environment.

Table 16 – Source Files

Source File Name	Description
aes.c	AES algorithm
des3.c	TDES algorithm
fips.c	FIPS entry point and API wrappers
fips_test.c	Power on Self Tests
hmac.c	HMAC algorithm
random.c	DRBG algorithm
rsa.c	RSA algorithm
sha.c	SHA algorithm
sha256.c	SHA-256 algorithm
sha512.c	SHA-512 algorithm
wolfcrypt_first.c	First FIPS function and Read Only address
wolfcrypt_last.c	Last FIPS function and Read Only address

10 Appendix A – Installation Instructions

This Appendix describes using wolfCrypt in FIPS 140-2 mode as a software component. The intended audience is Users and Crypto Officers using/needing FIPS software.

10.1 Linux INSTALLATION

wolfCrypt in FIPS mode requires the wolfCrypt FIPS library version 3.6.0 or later. The wolfCrypt FIPS releases can be obtained with a link provided by wolfSSL through direct email.

To verify the fingerprint of the package, calculate the SHA-256 sum using a FIPS 140-2 validated cryptographic module. The following command serves as an example:

```
shasum -a 256 wolfssl-3.6.0-commercial-fips-linux.7z
746341ac6d88b0d6de02277af5b86275361ed106c9ec07559aa57508e218b3f5
```

And compare the sum to the sum provided with the package. If for some reason the sums do not match stop using the module and contact wolfSSL.

To unpack the bundle:

```
7za x wolfssl-3.6.0-commercial-fips-linux.7z
```

When prompted, enter the password. The password is provided in the distribution email.

To build and install wolfCrypt with FIPS:

```
./configure --enable-fips

make check

sudo make install
```

If for some reason you have not received the library with FIPS support the ./configure step will fail. Please contact wolfSSL.

'make check' will verify the build and that the library is operating correctly. If 'make check' fails this probably means the In Core Integrity check has failed. To verify this do:

```
./wolfcrypt/test/testwolfcrypt

MD5 test passed!

in my Fips callback, ok = 0, err = -203 message = In Core
Integrity check FIPS error

hash =

622B4F8714276FF8845DD49DD3AA27FF68A8226C50D5651D320D914A5660B3F5

In core integrity hash check failure, copy above hashinto
```

```
verifyCore[] in fips_test.c and rebuild
```

Copy the value given for "hash" in the output, and replace the value of "verifyCore[]" in ./ctaocrypt/src/fips_test.c with this new value. After updating verifyCore, re-compile the wolfSSL library by running 'make check' again. The In Core Integrity checksum will vary with compiler versions and runtime library versions.

10.2 Linux SGX INSTALLATION

wolfCrypt in FIPS mode with SGX Enclave support requires the wolfCrypt FIPS library version 3.12.4 or later. The wolfCrypt FIPS releases can be obtained with a link provided by wolfSSL through direct email.

To verify the fingerprint of the package, calculate the SHA-256 sum using a FIPS 140-2 validated cryptographic module. The following command serves as an example:

```
shasum -a 256 wolfssl-3.13.0-commercial-fips-SGX.7z
746341ac6d88b0d6de02277af5b86275361ed106c9ec07559aa57508e218b3f5
```

And compare the sum to the sum provided with the package. If for some reason the sums do not match stop using the module and contact wolfSSL.

To unpack the bundle:

```
7za x wolfssl-3.13.0-commercial-fips-SGX.7z
```

When prompted, enter the password. The password is provided in the distribution email.

To build and install wolfCrypt with FIPS:

```
cd wolfssl-3.13.0-commercial-fips-SGX/fips/SGX/Linux-SGX-Harness/static-lib-dir

./build.sh
```

This will build a static library for linking with Enclaves. If for some reason you have not received the library with FIPS SGX support this will fail. Please contact wolfSSL.

```
cd wolfssl-3.13.0-commercial-fips-SGX/fips/SGX/Linux-SGX-Harness

./build.sh

./App -t
```

The test application will verify the build and that the library is operating correctly. If './App -t fails this probably means the In Core Integrity check has failed. To verify this do:

```
./App -t

Crypt Test:
```

```
Starting Power On Self Test

Power On Self Test FAILURE

error    test passed!

base64   test passed!

base64   test passed!

MD5      test passed!

MD4      test passed!

in my Fips callback, ok = 0, err = -203

message = In Core Integrity check FIPS error

hash =
E4E2899B697F1BC3B8E73F625C13E7899388DD08BCA7107C805660DDF0BEF64F

In core integrity hash check failure, copy above hash

into verifyCore[] in fips_test.c and rebuild

SHA      test failed!

 error = -1700

Crypt Test: Return code -1
```

Copy the value given for "hash" in the output, and replace the value of "verifyCore[]" in ./ctaocrypt/src/fips_test.c with this new value. After updating verifyCore, re-compile the static library and the application again. The In Core Integrity checksum will vary with compiler versions and runtime library versions.

10.3 iOS INSTALLATION

wolfCrypt in FIPS mode requires the wolfCrypt FIPS library version 3.6.0 or later. The wolfCrypt FIPS releases can be obtained with a link provided by wolfSSL through direct email.

To verify the fingerprint of the package, calculate the SHA-256 sum using a FIPS 140-2 validated cryptographic module. The following command serves as an example:

```
shasum –a 256 wolfssl-3.6.0-commercial-fips-ios.7z
32f7bfcb4ce250da3c43a3d944ab443e1be1c4508e86e0ef664a52ba3f4ea603
```

And compare the sum to the sum provided with the package. If for some reason the sums do not match stop using the module and contact wolfSSL.

To unpack the bundle:

```
7za x wolfssl-3.6.0-commercial-fips-ios.7z
```

When prompted, enter the password. The password is provided in the distribution email.

wolfCrypt with FIPS for iOS is used as a static library. One has to:

1. Build the library
2. Link it against their application
3. Get the In Core Integrity check value from the target platform
4. Copy the value given for "hash" in the output, and replace the value of "verifyCore[]" in ctaocrypt/src/fips_test.c with this new value
5. Rebuild the library
6. Relink it into the application

To build and install wolfCrypt with FIPS:

1. In Xcode open the project IDE/iOS/wolfssl-FIPS.xcodeproj
2. Select the build type and target
3. Archive the code to make a release library
4. If using a release library, click on the libwolfssl.a item in the file list, on the right pane click the copy button on the Full Path, open that path in the Finder, but delete everything after "Products" in the path, then pick the end product that was built, copy the header directory and the libwolfssl.a file into your project
5. In your application project, add the following preprocessor macros:
   - IPHONE
   - HAVE_FIPS
   - HAVE_HASHDRBG
   - HAVE_AESGCM
   - WOLFSSL_SHA512
   - WOLFSSL_SHA384
   - NO_MD4
   - NO_HC128
   - NO_RABBIT
   - NO_DSA
   - NO_PWDBASED
6. Build the project
7. Run the code on your target hardware with the standard cable connected, the default FIPS check failure should be output in the output window in Xcode

The first run should indicate the In Core Integrity check has failed:

```
in my Fips callback, ok = 0, err = -203 message = In Core
Integrity check     FIPS error

hash =
622B4F8714276FF8845DD49DD3AA27FF68A8226C50D5651D320D914A5660B3F5

In core integrity hash check failure, copy above hash  into
verifyCore[] in fips_test.c and rebuild
```

The In Core Integrity checksum will vary with compiler versions, runtime library versions, target hardware, and build type.

10.4 Android INSTALLATION

wolfCrypt in FIPS mode for Android requires the wolfCrypt Android FIPS library version 3.6.0 or later. The wolfCrypt FIPS releases can be obtained with a link provided by wolfSSL through direct email.

To verify the fingerprint of the package calculate the SHA-256 sum using a FIPS 140-2 validated cryptographic module. The following command serves as an example:

```
shasum -a 256 wolfssl-3.6.0-commercial-fips-android.7z
99c01cbf9c75d787ff34470e8c810af66c1443148ae8caf568a7c96e10419900
```

And compare the sum to the sum provided with the package. If for some reason the sums do not match stop using the module and contact wolfSSL.

To unpack the bundle:

```
7za x wolfssl-3.6.0-commercial-fips-android.7z
```

When prompted, enter the password. The password is provided in the distribution email.

The wolfCrypt FIPS for Android bundle contains the wolfSSL library, the wolfCrypt FIPS library (used to create the crypto boundary), the wolfCrypt JNI wrapper, and a sample Android NDK application (demonstrating how to correctly include wolfSSL, wolfCrypt FIPS, and wolfCrypt-JNI in an Android.mk file).

In order to build the wolfCrypt JNI wrapper and the wolfCrypt Android NDK sample application, Java, the Android SDK, and the Android NDK need to be installed on the development machine in use.

wolfSSL and wolfCrypt FIPS for Android are compiled as part of an Android NDK application's build process. Each Android NDK application has an Android.mk build file that controls the compilation of native shared libraries. This Android.mk file should be modified to compile shared libraries.

Both wolfCrypt FIPS and wolfCrypt JNI can be compiled by Android.mk, by following the example shown in the "Android NDK Sample App" (wolfcrypt-android-ndk). The Android.mk file for this project is located at:

```
./IDE/Android/wolfcrypt-android-ndk/jni/Android.mk
```

This sample demonstrates the correct use of source files, order of source files, and preprocessor defines to use.

The native shared libraries need to be loaded by the main Activity in a static block, in the correct order. Applications will need to call System.loadLibrary() in a static code block for both the wolfCrypt FIPS and wolfCrypt JNI shared libraries.

The FIPS library contains a self-check verify hash. Since the library is compiled as a shared library and is position independent, the library looks the same to every application that builds against it, and the code can be verified.

The library provides the function `wolfCrypt_GetCoreHash_fips()` that returns a string with the check value calculated with the existing code. The `verifyCore` in `fips_test.c` will need to be updated with this value, the library rebuilt then relinked into your application.

10.5 FreeRTOS INSTALLATION

wolfCrypt in FIPS mode for FreeRTOS requires the wolfCrypt FIPS library version 3.6.1 or later. The wolfCrypt FIPS releases can be obtained with a link provided by wolfSSL through direct email.

To verify the fingerprint of the package, calculate the SHA-256 sum using a FIPS 140-2 validated cryptographic module. The following command serves as an example:

```
shasum -a 256 wolfssl-3.6.1-commercial-fips-freertos.7z
```

And compare the sum to the sum provided with the package. If for some reason the sums do not match stop using the module and contact wolfSSL.

To unpack the bundle:

```
7za x wolfssl-3.6.1-commercial-fips-freertos.7z
```

When prompted, enter the password. The password is provided in the distribution email.

The wolfCrypt FIPS for FreeRTOS bundle contains the wolfSSL library and the wolfCrypt FIPS library. To build wolfCrypt with FIPS for FreeRTOS:

1. Build and link the library against the application or pull the source code and header files into the project with these preprocessor definitions:
   - FREERTOS
   - HAVE_FIPS
   - NO_DSA
   - NO_PSK
   - NO_MD4
   - NO_HC128
   - NO_PWDBASED
   - HAVE_HASHDRBG
   - WOLFSSL_SHA384
   - WOLFSSL_SHA512
   - NO_RC4
   - NO_RABBIT
2. Get the In Core Integrity check value from the target platform by running the application on the target platform and obtaining the “hash” value that is given in the output. The first run should indicate the In Core Integrity check has failed:

```
in my Fips callback, ok = 0, err = -203 message = In Core
Integrity check     FIPS error

hash =
622B4F8714276FF8845DD49DD3AA27FF68A8226C50D5651D320D914A566
0B3F5

In core integrity hash check failure, copy above hash  into
verifyCore[] in fips_test.c and rebuild
```

The In Core Integrity checksum will vary with compiler versions, runtime library versions, target hardware, and build type.

3. Copy the value given for "hash" in the output, and replace the value of "verifyCore[]" in ctaocrypt/src/fips_test.c with this new value.
4. Rebuild the library.
5. Relink it into the application.

10.6 Windows 7 INSTALLATION

wolfCrypt in FIPS mode for Windows 7 requires the wolfCrypt FIPS library version 3.6.6 or later. The wolfCrypt FIPS releases can be obtained with a link provided by wolfSSL through direct email.

To verify the fingerprint of the package, calculate the SHA-256 sum using a FIPS 140-2 validated cryptographic module. The following command serves as an example:

```
shasum –a 256 wolfssl-3.6.6-commercial-fips-windows.7z
02da35d0a4d6b8e777236fe30da7a6ff93834fb16939ea16da663773f1b34cf0
```

And compare the sum to the sum provided with the package. If for some reason the sums do not match stop using the module and contact wolfSSL.

To unpack the bundle:

```
7za x wolfssl-3.6.6-commercial-fips-windows.7z
```

When prompted, enter the password. The password is provided in the distribution email.

wolfCrypt with FIPS for Windows is used as a static library. One has to:

1. Build the library
2. Link it against their application
3. Get the In Core Integrity check value from the target platform
4. Copy the value given for "hash" in the output, and replace the value of "verifyCore[]" in fips_test.c with this new value
5. Rebuild the library
6. Relink it into the application

To build and install wolfCrypt with FIPS:

1. In Visual Studio open IDE\WIN\wolfssl-fips.sln
2. Select the build type and target (Release x64)
3. Build the solution
4. The library should be in the directory IDE\WIN\Release\x64 as wolfssl-fips.lib, it can be added to your project

5. In your application project, add the following preprocessor macros:
   - HAVE_FIPS
   - HAVE_HASHDRBG
   - HAVE_AESGCM
   - WOLFSSL_SHA512
   - WOLFSSL_SHA384
   - NO_MD4
   - NO_HC128
   - NO_RABBIT
   - NO_DSA
   - NO_PWDBASED
6. Build the solution
7. Run the code from the Release\x64 directory, the default FIPS check failure should be output in the shell

The first run should indicate the In Core Integrity check has failed:

```
in my Fips callback, ok = 0, err = -203 message = In Core
Integrity check FIPS error

hash =
622B4F8714276FF8845DD49DD3AA27FF68A8226C50D5651D320D914A5660B3F5

In core integrity hash check failure, copy above hash into
verifyCore[] in fips_test.c and rebuild
```

The In Core Integrity checksum will vary with compiler versions, runtime library versions, target hardware, and build type.

Note: if using 32-bit builds, one must disable Randomize Base Address. For Operating Environments 13 and 14 of “Table 1 – Tested Operating Environments”, enable the option RSA_LOW_MEM (add the define to <wolfssl-root>/IDE/WIN/user_settings.h)

10.7 Windows SGX Installation

wolfCrypt in FIPS mode with SGX Support on Windows requires the wolfCrypt FIPS library version 3.12.4 or later. The wolfCrypt FIPS releases can be obtained with a link provided by wolfSSL through direct email.

To verify the fingerprint of the package, calculate the SHA-256 sum using a FIPS 140-2 validated cryptographic module. The following command serves as an example:

```
shasum –a 256 wolfssl-3.13.0-commercial-fips-SGX.7z
02da35d0a4d6b8e777236fe30da7a6ff93834fb16939ea16da663773f1b34cf0
```

And compare the sum to the sum provided with the package. If for some reason the sums do not match stop using the module and contact wolfSSL.

To unpack the bundle:

```
7za x wolfssl-3.13.0-commercial-fips-SGX.7z
```

When prompted, enter the password. The password is provided in the distribution email.

wolfCrypt with FIPS and SGX Support for Windows is used as a static library. One has to:

7. Build the library
8. Link it against their application
9. Get the In Core Integrity check value from the target platform
10. Copy the value given for "hash" in the output, and replace the value of "verifyCore[]" in fips_test.c with this new value
11. Rebuild the library
12. Relink it into the application

To build and install wolfCrypt with FIPS and SGX Support:

8. In Visual Studio open fips\SGX\Windows-SGX-Harness-and-optest\static-lib-dir\wolfSSL_SGX.sln
9. Select the build type and target (Debug Win32)
10. Build the solution
11. The library should be in the directory …static-lib-dir\Debug\Win32 as wolfssl.lib, it can be added to your project.
12. An example application "Benchmarks" has been provided in fips\SGX\Windows-SGX-Harness-and-optest\Benchmarks.sln
13. Settings are included in fips\SGX\Windows-SGX-Harness-and-optest\user-settings directory along with the pre-processor macros that are set in the Enclave and Benchmark Apps built by the Benchmarks.sln
14. Build the provided example solution
15. Run the code from the Debug\ directory, the default FIPS check failure should be output in the shell

The first run should indicate the In Core Integrity check has failed:

```
benchmark.exe -t

Crypt Test:

Starting Power On Self Test

Power On Self Test FAILURE

error    test passed!

base64   test passed!

base64   test passed!

MD5      test passed!

MD4      test passed!

in my Fips callback, ok = 0, err = -203

message = In Core Integrity check FIPS error

hash =
E4E2899B697F1BC3B8E73F625C13E7899388DD08BCA7107C805660DDF0BEF64F
```

```
In core integrity hash check failure, copy above hash

into verifyCore[] in fips_test.c and rebuild

SHA      test failed!

 error = -1700

Crypt Test: Return code -1
```

The In Core Integrity checksum will vary with compiler versions, runtime library versions, target hardware, and build type.

10.8 NET + OS v7.6 Installation

Browse to where the Digi NET + OS build environment is located. The sub-directory path will be: "…\DIGI-BAXTER\Digi\Digi NET+OS 7.6\GNU Tools\"

Find and double-click "Digi NET + OS 7.6 Build Environment"

Execute this command: "cd <path-to>/netos_sources/src/wolfssl/

Execute the "make" command which will build and output libwolfssl.a

Next change directory to the application. By default examples are provided, see wolfssl/examples/client for reference. In each application directory there will be a "32b/" directory which contains the makefile for the application and a dependency on libwolfssl.a. There is also a appconf.h header file in the root directory of each example which can be used to configure the application stack and other variable parameters for the application. From the 32b/ directory of the application execute the "make" command to build the app and link libwolfssl.a. Once complete make sure that the image and debug binaries "image.elf" and "image.bin" were successfully generated.

10.8.1 Connecting the hardware / Debugging

To test and debug an application connect a Digi JTAG LINK debugger to the device and host computer. Connect a Serial modem cable to P3 on the device and to a host computer running a Terminal service such as "Tera Term" for windows or "CoolTerm" for macOS to view device outpu.

Use the <path-to>\SEGGER\JLinkARM_V408l\ JLinkGDBServer.exe to connect to the ARM9 core. Once all three lower indicators are green you may now execute the command "gdbtk -se image.elf" from the "Digi NET + OS 7.6 Build Environment" (run from the 32b/ directory of the app to be debugged/tested).

Click "Yes" if prompted while the binary is being downloaded. Once the download is complete a debug window will open, select "Continue" (Little icon with two curly braces and a right-facing red arrow).

The first time the application is executed and makes a call into the FIPS module you will see the NETOS FIPS callback return a message similar to this:

```
in my Fips callback, ok = 0, err = -203

message = In Core Integrity check FIPS error

hash =
E4E2899B697F1BC3B8E73F625C13E7899388DD08BCA7107C805660DDF0BEF64F
```

```
In core integrity hash check failure, copy above hash

into verifyCore[] in fips_test.c and rebuild

SHA      test failed!

 error = -1700

Crypt Test: Return code -1
```

Once you see this message copy the hash and modify the source file <path-to>/netos_sources/src/wolfssl/ctaocrypt/src/fips_test.c. Search for the variable “verifyCore” and paste the new hash over the old. Return to the wolfSSL root directory i.e. <path-to>/netos_sources/src/wolfssl/ and re-build libwolfssl.a by running “make clean && make localclean && make”. Then return to the application directory (for example) <path-to>/netos_sources/src/wolfssl/examples/client/32b and run “make clean && make”. This will recompile the application with the updated libwolfssl.a which now has an updated hash. Debug the application a second time. Now the call into the wolfCrypt module should succeed.

10.8.2 Configure Network

Using the Terminal Interface press a button within the first five seconds of the launch to configure the board to connect to a wireless network. Once configured ensure the device is assigned an IP address. This IP will be used to permanently flash image.bin to the device once app is debugged and working as expected.

10.8.3 Permanent installation

Once the app has been debugged, the in-core hash is updated and the device has received an IP address. Use the “Digi NET + OS 7.6 Build Environment” to launch a file transfer protocol connection to the device using the IP address assigned to it. Example:

ftp 192.168.1.119

The default user name is “root” the default password is “password”. Once the ftp connection is established switch to binary mode and “put” the image.bin that was compiled for the application.

10.8.3.1 “Digi NET + OS 7.6 Build Environment” exchange

```
ftp 192.168.1.119

Connected to 192.168.1.119.

220 NET+OS 7.6.1.8 FTP server ready.

Name (192.168.1.119:nick): root

331 User root OK, send password.

Password:

230 Password OK.

Remote system type is NET+ARM.

ftp> binary

200 Type set to I.
```

```
ftp> put image.bin
200 PORT command Ok.
150 About to open data connection.
226 Transfer complete
2520448 bytes sent in 7.01 seconds (359345 bytes/s)
```

10.8.3.2 Serial Port Terminal Application exchange

At the same time the above is occurring in the Serial Port Terminal Application you should see the following messages printed out:

```
Checksum passed, writing to flash...
Firmware updated, quit the session to restart.
```

Once you see the firmware was successfully updated return to the ftp connection and type “quit”. Upon the ftp service disconnecting the board will automatically reset itself and launch the newly installed application.

If the FIPS module ever enters an error state the only solution to recover from that error state is to power off the device and power it back on again. Power cycling will return the device to a working state.

10.9 wolfCrypt FIPS API

wolfCrypt adds the string _fips to all FIPS mode APIs. For example, ShaUpdate() becomes ShaUpdate_fips(). The FIPS mode functions can be called directly, but they can also be used through macros.

HAVE_FIPS is defined when using wolfCrypt in FIPS mode and that creates a macro for each function with FIPS support. For the above example, a user with an application calling ShaUpdate() can recompile with the FIPS module and automatically get ShaUpdate_fips() support without changing their source code. Of course, recompilation is necessary with the correct macros defined.

A new error return code:

FIPS_NOT_ALLOWED_E

may be returned from any of these functions used directly or even indirectly.

The error is returned when the Power-On Self-Tests (POST) are not yet complete or they have failed. POST is done automatically as a default entry point when using the library, no user interaction is required to start the tests. To see the current status including any error code at any time call wolfCrypt_GetStatus_fips(). For example, if the AES Known Answer Test failed during POS GetStatus may return

AES_KAT_FIPS_E