OpenSSL FIPS Library and Android Guide

Introduction

This document will provide instructions for building the OpenSSL FIPS Object Module and OpenSSL FIPS Capable library for Android devices. The FIPS Object Module provides validated cryptography, and the FIPS Capable Library uses the validated cryptography[1]. As an OpenSSL developer, you will use the library the same as in the past – except you must call `FIPS_mode_set` to enter FIPS mode and engage the validated cryptography.

The FIPS Object Module, `fipscanister.o`, is a sequestered container of object code and data built from source code. The sources, object code and data are strictly controlled by the OpenSSL FIPS 140-2 Security Policy. No changes can be made to the procedure for building the FIPS Object Module, and no changes can be made to the sources. If you need to make changes to the FIPS Object Module, you will need to engage the OpenSSL Foundation for a separate validation.

The FIPS Capable Library is comprised of `libcrypto` and `libssl`. They are the same libraries you have been using for years. The FIPS Capable Library is tolerant of changes to procedures and source code. You are allow to modify them within reason, as long as the changes do not adversely affect the FIPS Object Module.

This guide is intended to be informative and easy to use. In case of discrepancies between this document and the OpenSSL FIPS Security Policy, the Security Policy will prevail. You can download the Security Policy from http://www.openssl.org/docs/fips/.

The instructions that follow depend upon a properly configured Android NDK and SDK. The NDK is used to compile programs and link the OpenSSL library; while SDK tools are used to push programs to a device. Be sure ANDROID_NDK_ROOT and ANDROID_SDK_ROOT are set properly[2], and the SDK's tools and platform-tools are available.

Executive Summary

Use the following commands to build and install the OpenSSL FIPS Object Module and OpenSSL FIPS Capable library. Before running the commands download openssl-1.0.1e.tar.gz, openssl-fips-2.0.5.tar.gz and setenv-android.sh; place the files in the same directory (the 'root' directory mentioned below); ensure ANDROID_NDK_ROOT is set; and verify setenv-android.sh suites your taste. ANDROID_API and ANDROID_TOOLCHAIN will be set by the setenv-android.sh script. The files can be obtained from http://www.openssl.org/source/ and http://openssl.com/fips/2.0/platforms/android/.

---

1 In the FIPS 140-2 world, there are two types of cryptography: validated and not validated. There is no such thing as “FIPS 140-2 certified,” “FIPS 140-2 conformant” or “FIPS 140-2 compliant”. If the marketing department uses a term other than FIPS Validated, then the US government will not recognize your program as FIPS-140-2.

2 *Recommended NDK Directory?*, https://groups.google.com/group/android-ndk/browse_thread/thread/a998e139aca71d77

Prepare the OpenSSL Sources

```
# From the 'root' directory

$ rm -rf openssl-fips-2.0.5/
$ rm -rf openssl-1.0.1e/

$ tar xzf openssl-fips-2.0.5.tar.gz
$ tar xzf openssl-1.0.1e.tar.gz

$ chmod a+x setenv-android.sh
```

Build the FIPS Object Module

```
# From the 'root' directory

$ . ./setenv-android.sh

$ cd openssl-fips-2.0.5/

$ ./config

$ make

$ sudo make install

# Execute after install

$ sudo -E cp $FIPS_SIG /usr/local/ssl/fips-2.0/bin

$ sudo -E mv /usr/local/ssl/fips-2.0/ /usr/local/ssl/$ANDROID_API
```

Build the FIPS Capable Library

```
# From the 'root' directory

$ . ./setenv-android.sh

$ cd openssl-1.0.1e/

$ perl -pi -e 's/install: all install_docs install_sw/install:
install_docs install_sw/g' Makefile.org

$ ./config fips shared -no-ssl2 -no-ssl3 -no-comp -no-hw -no-engine
--openssldir=/usr/local/ssl/$ANDROID_API
--with-fipsdir=/usr/local/ssl/$ANDROID_API
--with-fipslibdir=/usr/local/ssl/$ANDROID_API/lib/

$ make depend

$ make all

$ sudo -E make install
CC=$ANDROID_TOOLCHAIN/arm-linux-androideabi-gcc
RANLIB=$ANDROID_TOOLCHAIN/arm-linux-androideabi-ranlib
```

OpenSSL FIPS Components

While the Executive Summary provided the whirlwind instructions for building and installing the OpenSSL library, this sections provides detailed instructions. There are six steps to building the

FIPS Object Module and FIPS Capable Library for use in various projects, and they are listed below. Projects range from simple NDK based command line programs to Android activities using the JNI bridge.

1. Acquire the required files
2. Adjust the cross-compilation script
3. Prepare the OpenSSL sources
4. Set the Incore utility PATH
5. Build the FIPS Object Module
6. Build the FIPS Capable Library

Acquire the Required Files

First, obtain the base files from http://www.openssl.org/source/:

- openssl-1.0.1e.tar.gz
- openssl-fips-2.0.5.tar.gz

Next, acquire the auxiliary files which can be obtained from http://openssl.com/fips/2.0/platforms/android/. You won't need all the files from the location.

- setenv-android.sh

In addition to the required and auxillary files, there is a test program available. Download it from http://openssl.com/fips/2.0/platforms/android/.

- fips_hmac.c

openssl-fips-2.0.5.tar.gz includes the FIPS Object Module. openssl-1.0.1e.tar.gz includes the FIPS Capable Library. setenv-android.sh is used to set the cross-compilation environment. fips_hmac.c is used to test the static and dynamic libraries on the device.

After collecting the required files, your working directory will look similar to below.

```
android-openssl-fips $ ls -l
-rw-r--r-- 1      2718 Jun 23 17:54 fips_hmac.c
-rw-r--r-- 1   4459777 Jun 15 03:32 openssl-1.0.1e.tar.gz
-rw-r--r-- 1   1442754 Jun 23 16:37 openssl-fips-2.0.5.tar.gz
-rwxr-xr-x 1      6760 Jun 23 01:52 setenv-android.sh
```

Adjust the Cross-Compilation Script

`setenv-android.sh` is used to set the cross-compilation environment. Open the script an ensure the following match your needs. If you are using android-ndk-r8e, android-14, and ANDROID_NDK_ROOT is set, then the script should be ready to use as-is.

- _ANDROID_NDK – the version of the NDK. For example, android-ndk-r8e
- _ANDROID_EABI – the version of the EABI tools. For example, arm-linux-androideabi-4.6

- _ANDROID_API – the API level. For example, android-14

You should also set `ANDROID_SDK_ROOT` and `ANDROID_NDK_ROOT`. The environmental variables are used internally by the Android platform tools and scripts.[3]

Prepare the OpenSSL Sources

Remove stale versions of the OpenSSL FIPS Object Module and FIPS Capable library. Then unpack fresh files. Also ensure the script is executable.

```
$ rm -rf openssl-fips-2.0.5/
$ rm -rf openssl-1.0.1e/

$ tar xzf openssl-fips-2.0.5.tar.gz
$ tar xzf openssl-1.0.1e.tar.gz

$ chmod a+x setenv-android.sh
```

Set the Incore Utility Path

The `incore` utility is a shell script used by `fipsld` to embed the FIPS Object Module's expected fingerprint in the OpenSSL shared object or program. The script is located in `openssl-fips-2.0.5/util`, and you must ensure `incore` can be found by `fipsld`.

Since `openssl-fips-2.0.5/util` is probably not included as one of the directories in the search list specified by the `PATH` environment variable, the OpenSSL library allows you to specify its path via `FIPS_SIG`. Export `FIPS_SIG` as follows by executing `find` from your working directory. Below, the root directory (`PWD`) is `/home/user/android-openssl-fips/`.

```
$ export FIPS_SIG=`find $PWD -name incore`
$ echo $FIPS_SIG
/home/user/android-openssl-fips/openssl-fips-2.0.5/util/incore
```

If `incore` is already installed in a known location, you can use it from there instead:

```
$ export FIPS_SIG=`find /usr/local/ssl -name incore`
$ echo $FIPS_SIG
/usr/local/ssl/android-14/bin/incore
```

Build the FIPS Object Module

The FIPS Object Module provides the validated cryptography for the OpenSSL library. This section of the document will guide you through the creation of the FIPS Object Module. The Module is governed by the FIPS 140-2 program requirements and you cannot deviate from the OpenSSL FIPS 140-2 Security Policy during any stage during handling, from acquisition, through building, to installation.

The FIPS Object Module build procedures use the cross-compilation tools supplied in the NDK. It does not use Android.mk and friends. A shell script is used to set the environment for the cross-compilation, and you might need to adjust the script to suit your taste.

---

3 *Recommended NDK Directory?*, https://groups.google.com/group/android-ndk/browse_thread/thread/a998e139aca71d77

To compile the FIPS Object Module for the embedded platform, perform the following steps. You cannot specify any arguments to `config` or `make`. *Note the leading '.' when running the setenv-android.sh script.* If you have any errors from the script, then you should fix them before proceeding.

```
$ . ./setenv-android.sh
$ cd openssl-fips-2.0.5/
$ ./config
$ make
```

After make completes, verify `fipscanister.o` was built for the embedded architecture.

```
$ find . -name fipscanister.o
./fips/fipscanister.o

$ readelf -h ./fips/fipscanister.o | grep -i 'class\|machine'
  Class:                             ELF32
  Machine:                           ARM
```

Finally, install the library. You must run `make install` without arguments.

```
$ sudo make install
```

After installing the FIPS Object Module, the four files of interest can be found in the `lib/` directory.

```
 $ ls /usr/local/ssl/fips-2.0/lib/
fipscanister.o  fipscanister.o.sha1  fips_premain.c
fips_premain.c.sha1
```

Once installed, you are outside the scope of FIPS 140-2 and allowed to move the library as long as it remains protected.

```
$ sudo mv /usr/local/ssl/fips-2.0/ /usr/local/ssl/android-14
$ ls /usr/local/ssl/
android-14
```

The `android-14` is the API level you are building for, and is needed if you have multiple OpenSSL libraries installed (for example, you might have `macosx`, `iphoneos`, and `android-14` in `/usr/local/ssl`). You can retrieve the API level from `ANDROID_API`, which was set in the `setenv-android.sh` script:

```
$ echo $ANDROID_API
android-14
```

Finally, copy `incore` into the installation directory. This will allow you to delete the temporary folders in your working area (`openssl-fips-2.0.5` and `openssl-1.0.1e`) once the libraries are installed.

```
$ find . -name incore
./util/incore
$ sudo cp ./util/incore /usr/local/ssl/android-14/bin/
$ ls /usr/local/ssl/android-14/bin/
fipsld  fips_standalone_sha1  incore
```

Build the FIPS Capable Library

The FIPS Capable Library is a standard OpenSSL distribution that can use the validated cryptography provided by the FIPS Object Module. This section of the document will guide you through the creation of the the FIPS Capable Library. You are allowed to modify the FIPS Capable Library within reason, as long as it does not adversely affect the FIPS Object Module.

The FIPS Capable version of the library can operate with or without FIPS validated cryptography. It handles all the details of operation while in FIPS mode after you successfully call `FIPS_mode_set`. If you don't call `FIPS_mode_set`, the library will still operate as expected; but it will not be using validated cryptography.

Its recommended that you build the shared object since Android will load and link it out of the box. If you build a static library, then you will have to build a wrapper shared object around the static archive.

The FIPS Capable `Makefile` (and `Makefile.org`) needs its `install` rule modified. The `install` rule includes the `all` target, which causes items to be built during install. A bug in the process when running as root results in an empty signature for the shared object (the signature is a string of zeros).

To build the FIPS Capable library, you must issue `config fips`, but other options are up to you. Some suggested options for configure include: `shared`, `-no-ssl2`, `-no-ssl3`, `-no-comp`, `-no-hw`, and `-no-engine`. `shared` will build and install both the shared object and static archive. You should specify `--openssldir`, `--with-fipsdir` and `--with-fipslibdir` to ensure the FIPS Capable build system finds components from the FIPS Object Module.

Begin building the FIPS Capable library by setting the cross-compilation environment. *Note the leading '.' when running the setenv-android.sh script.* If you have any errors from the script, then you should fix them before proceeding.

```
$ . ./setenv-android.sh
$ cd openssl-1.0.1e/
```

Next, fix the makefile and run configure.

```
$ perl -pi -e 's/install: all install_docs install_sw/install:
install_docs install_sw/g' Makefile.org
```

```
$ ./config fips shared -no-ssl2 -no-ssl3 -no-comp -no-hw -no-engine
--openssldir=/usr/local/ssl/android-14/
--with-fipsdir=/usr/local/ssl/android-14/
--with-fipslibdir=/usr/local/ssl/android-14/lib/
```

Then run `make depend` and `make all`:

```
$ make depend
$ make all
```

After `make` completes, verify `libcrypto.a` and `libssl.a` were built for the embedded architecture.

```
$ find . -name libcrypto.a
./libcrypto.a
```

```
$ readelf -h ./libcrypto.a | grep -i 'class\|machine' | head -2
  Class:                             ELF32
  Machine:                           ARM
```

Finally, install the library. The makefile's `install` rule uses both `CC` and `RANLIB`, so you will need to fully specify the command variables on the command line (during install, `sudo` drops the user's path). You must also use sudo's -E option; otherwise `ANDROID_TOOLCHAIN` will be empty and tools such as `arm-linux-androideabi-gcc` and `arm-linux-androideabi-ranlib` will not be found.

```
$ sudo -E make install
CC=$ANDROID_TOOLCHAIN/arm-linux-androideabi-gcc
RANLIB=$ANDROID_TOOLCHAIN/arm-linux-androideabi-ranlib
```

Testing the OpenSSL Libraries

Testing the installation consists of building a sample program, installing it with `adb`, and then running the program using a remote shell. Both the static and dynamic version of the OpenSSL library can be tested using fips_hmac, which is a test program to calculate a hmac over the files given as arguments.

The test program is built in a cross-compilation environment, just like the FIPS Object Module and FIPS Capable library. To begin, set the Android environment and verify `ANDROID_SYSROOT`. *Note the leading '.' when running the setenv-android.sh script.* If you have any errors from the script, then you should fix them before proceeding.

```
$ . ./setenv-android.sh
```

```
$ echo $ANDROID_SYSROOT
/opt/android-ndk-r8e/platforms/android-14/arch-arm
```

Linking with the Shared Object

Linking with the shared object is easiest. That's because the FIPS Capable library build process takes care of a number of items for you, including running `fipsld` on the shared object. The downside to dynamic linking is you have to push the program and shared object to the device and modify the loader path before executing.

The command below compiles `fips_hmac.c` using `ANDROID_SYSROOT` and the shared object (`libcrypto.so`). `ANDROID_SYSROOT` specifies the location of Android's headers and libraries, and is set using `setenv-android.sh`.

```
$ arm-linux-androideabi-gcc --sysroot="$ANDROID_SYSROOT"
-I/usr/local/ssl/android-14/include fips_hmac.c -o fips_hmac.exe
/usr/local/ssl/android-14/lib/libcrypto.so
```

There's no need to run `fipsld` on a program which dynamically links to the OpenSSL library.

Once the program is built, push it to the device and execute it.

```
# Copy the program and shared library to the Android device
$ adb push fips_hmac.exe /data/local/tmp/
303 KB/s (18548 bytes in 0.059s)
```

```
$ adb push libcrypto.so.1.0.0 /data/local/tmp
1106 KB/s (2154620 bytes in 1.900s)

# Execute the program on the Android device
$ adb shell

shell@android: $ cd /data/local/tmp
shell@android: $ LD_LIBRARY_PATH=./; ./fips_hmac.exe -v
fips_hmac.exe

FIPS mode enabled
f178788e8a439dbaaa760ef774e8d92e01d2440e
```

The hashes produced by the test program will vary with the files being digested.

Linking with the Static Archive

Linking against the static library is more challenging. That's because you have to compile your program to produce an object file, and then run `fipsld` to embed the FIPS fingerprint and produce the final executable. The upside to static linking is its easy to push and run the binary on the device.

Before compiling the test program, you need to set two variables – `CC` and `FIPSLD_CC`. They must be exported because `fipsld` uses them. Behind the scenes, `fipsld` will compile and link `fips_premain.c`, modify the `libcrypto` archive, assemble the final program, and embed the fingerprint.

```
$ export CC=`find /usr/local/ssl/$ANDROID_API -name fipsld`
$ echo $CC
/usr/local/ssl/android-14/bin/fipsld

$ export FIPSLD_CC="$ANDROID_TOOLCHAIN/arm-linux-androideabi-gcc"
$ echo $FIPSLD_CC
/opt/android-ndk-r8e/toolchains/arm-linux-androideabi-4.6/prebuilt/
linux-x86/bin/arm-linux-androideabi-gcc
```

Finally, compile and link as normal using `CC`. The command below compiles `fips_hmac.c` using `ANDROID_SYSROOT` and the static archive (`libcrypto.a`). `ANDROID_SYSROOT` specifies the location of Android's headers and libraries, and is set using `setenv-android.sh`.

```
$CC --sysroot="$ANDROID_SYSROOT"
-I/usr/local/ssl/android-14/include fips-test.c -o fips-test.exe
/usr/local/ssl/android-14/lib/libcrypto.a
```

Once the program is built, push it to the device and execute it.

```
# Copy the program to the Android device
$ adb push fips_hmac.exe /data/local/tmp
1099 KB/s (1462276 bytes in 1.298s)

# Execute the program on the Android device
$ adb shell
shell@android: $ cd /data/local/tmp

shell@android: $ fips_hmac.exe -v fips_hmac.exe
FIPS mode enabled
```

```
a7518364fbba32cca0df974ee646e8a13833eb3d
```

The hashes produced by the test program will vary with the files being digested.

Miscellaneous

Once the FIPS Object Module and FIPS Capable Library are installed, you can safely delete the source directories. The headers, libraries and programs (such as `fipsld` and `incore`) are located in subdirectories of `/usr/local/ssl/<platform>`.

The NDK supplies headers for each major platform - for example, API 14, API 9, API 8, and API 5. If you are building for Android 4.2 (API 17), Android 4.1 (API 16) and Android 4.0 (API 14), then you would use the NDK's API 14 (android-14 platform).[4]

Specify the full library name when calling Java's `System.load`. That is, call `System.load("libcrypto.so.1.0.0")`. Also note that some Android routines expect the prefix of "lib" and suffix of "so", so you might have to rename the library.

Some versions of the Android Java system loader will load the system's version of the OpenSSL library, even though you built and included a copy with your application. In this case, you might need to write a wrapper shared object and link to the static version of the OpenSSL library.

For Android, you never compile `fips_premain.c` and you never link against `fipscanister.o`. `fipsld` will compile and link `fips_premain.c`, and `libcrypto.a` will include `fipscanister.o`.

Internally, `fipsld` will call on `incore`. If you try to use `incore` directly (rather than through `fipsld`), you will encounter a number of problems. The problems are outside the scope of this guide.

If you compile with `-fPIE` and `-pie`, then you will core dump unless using Android 4.1 and above. Logcat shows the linker (`/system/bin/linker`) is the problem.

```
shell@android: $ ./fips_hmac.exe -v fips_hmac.exe
[2] + Stopped (signal)     ./fips_hmac.exe -v fips_hmac.exe
[1] - Segmentation fault   ./fips_hmac.exe -v fips_hmac.exe
```

4 *Relationship between APIs, EABIs, and Toolchains*, https://groups.google.com/forum/#!topic/android-ndk/KBfiNaGv7uk