wolfCrypt JCE Provider and JNI Wrapper Manual

2024-03-13

Contents

1   Introduction

The JCE (Java Cryptography Extension) framework supports the installation of custom Cryptographic Service Providers which can in turn implement a subset of the underlying cryptographic functionality used by the Java Security API.

This document describes the details and usage of the wolfCrypt JCE provider. The wolfCrypt JCE provider (wolfJCE) wraps the native wolfCrypt cryptography library for compatibility with the Java Security API. See the Github repository here.

The wolfcrypt-jni package contains both the wolfCrypt JNI wrapper in addition to the JCE provider. The JNI wrapper can be used independently if desired.

2   Requirements

2.1   Java / JDK

wolfJCE requires Java to be installed on the host system. There are several JDK variants available to users and developers - including the Oracle JDK and OpenJDK. wolfJCE has currently been tested with OpenJDK, Oracle JDK, and Android. OpenJDK and Android do not require JCE providers to be code signed, whereas the Oracle JDK does. For details on code signing, please see Chapter 7

For reference, the specific version of OpenJDK which wolfJCE has been tested with is:

```
$ java -version
Openjdk version "1.8.0"_91
OpenJDK Runtime Environment (build 1.8.0_91-8u91-b14-3ubuntu1~15.10.1~b14)
OpenJDK 64-Bit Server VM (build 25.91-b14, mixed mode)
```

It has also been tested with Oracle JDK 1.8.0_121, and Android 24.

2.2   JUnit

In order to run the unit tests, JUnit is required to be installed on the development system. JUnit can be downloaded from the project website at www.junit.org.

To install JUnit on a Unix/Linux/OSX system: 1. Download "**junit-4.13.jar**" and "**hamcrest-all-1.3.jar**"from junit.org/junit4/. At the time of writing, the mentioned .jar files could be downloaded from the following links:

LINK: junit-4.13.jar LINK: hamcrest-all-1.3.jar

2. Place these JAR files on your system and set JUNIT_HOME to point to that location. Ex:

```
$ export JUNIT_HOME=/path/to/jar/files
```

2.3   make and ant

Both "make" and "ant" are used to compile native C code and Java code, respectively.

Please ensure that these are installed on your development machine.

2.4   wolfSSL / wolfCrypt Library

As a wrapper around the native wolfCrypt library, wolfSSL must be installed on the host platform and placed on the include and library search paths. wolfJCE can be compiled against either the FIPS or non-FIPS version of the wolfSSL/wolfCrypt native library.

2.4.1   Compiling wolfSSL/wolfCrypt

To compile and install wolfSSL in a Unix/Linux environment for use with wolfJCE, please follow build instructions in the wolfSSL Manual. The most common way to compile wolfSSL is with the Autoconf system.

You can install a wolfSSL (wolfssl-x.x.x), wolfSSL FIPS release (wolfssl-x.x.x-commercial-fips), or wolfSSL FIPS Ready release.In any of these cases, you will need the --enable-keygen ./configure option.

wolfSSL Standard Build:

```
$ cd wolfssl-x.x.x
$ ./configure --enable-keygen
$ make check
$ sudo make install
```

wolfSSL FIPSv1 Build:

```
$ cd wolfssl-x.x.x-commercial-fips
$ ./configure --enable-fips --enable-keygen
$ make check
$ sudo make install
```

wolfSSL FIPSv2 Build:

```
$ cd wolfssl-x.x.x-commercial-fips
$ ./configure --enable-fips=v2 --enable-keygen
$ make check
$ sudo make install
```

wolfSSL FIPS Ready Build:

```
$ cd wolfssl-x.x.x-commercial-fips
$ ./configure --enable-fips=ready --enable-keygen
$ make check
$ sudo make install
```

This will install the wolfSSL library to your system default installation location. Onmany platforms this is:

```
/usr/local/lib
/usr/local/include
```

3   Compilation

Before following steps in this section, please ensure that the dependencies in Chapter 2 above are installed.

First, copy the correct "makefile" for your system, depending if you are on Linux or OS X. For example, if you were on Linux:

```
$ cd wolfcrypt-jni
$ cp makefile.linux makefile
If you are instead on Mac OSX:
$ cd wolfcrypt-jni
$ cp makefile.macosx makefile
```

Then proceed to compile the native (C source) code with "make":

```
$ cd wolfcrypt-jni
$ make
```

To compile the Java sources, "ant" is used. There are several ant targets to compile either the JNI or JCE (includes JNI) packages, in either debug or release mode. Running regular "ant" will give usage options:

```
$ ant
...
build:
[echo] wolfCrypt JNI and JCE
[echo]
   ----------------------------------------------------------------------------

[echo] USAGE:
[echo] Run one of the following targets with ant:
[echo] build-jni-debug | builds debug JAR with only wolfCrypt JNI classes
[echo] build-jni-release | builds release JAR with only wolfCrypt JNI classes
[echo] build-jce-debug | builds debug JAR with JNI and JCE classes
[echo] build-jce-release | builds release JAR with JNI and JCE classes
[echo]
   ----------------------------------------------------------------------------
```

Use the build target that matches your need. For example, if you want to build the wolfJCE provider in release mode, run:

```
$ ant build-jce-release
```

And, to run the JUnit tests, run the following command. This will compile only the tests that match the build that was done (JNI vs. JCE) and run those tests as well.

```
$ ant test
```

To clean the both Java JAR and native library:

```
$ ant clean
$ make clean
```

3.1   API Javadocs

Running ant will generate a set of Javadocs under the `wolfcrypt-jni/docs` directory. To view the root document, open the following file in a web browser:

`wolfcrypt-jni/docs/index.html`

4   Installation

There are two ways that wolfJCE can be installed and used:

4.1   Installation at Runtime

To install and use wolfJCE at runtime, first make sure that "**libwolfcryptjni.so**" is on your system's library search path. On Linux, you can modify this path with:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/add
```

Next, place the wolfCrypt JNI / wolfJCE JAR file (**wolfcrypt-jni.jar**) on your Java classpath. You can do this by adjusting your system classpath settings, or at compile time and runtime like so:

```
$ javac -classpath <path/to/jar> ...
$ java -classpath <path/to/jar> ...
```

Finally, in your Java application, add the provider at runtime by importing the provider class and calling Security.addProvider():

```
import com.wolfssl.provider.jce.WolfCryptProvider;
public class TestClass {
    public static void main(String args[]) {
        ...
        Security. **addProvider** (new **WolfCryptProvider**());
        ...
    }
}
```

To print a list of all installed providers for verification, you can do:

```
Provider[] providers = Security. **getProviders** ()
for (Provider prov:providers) {
    System.out.println(prov);
}
```

4.2   Installation at OS / System Level

To install the wolfJCE provider at the system level, copy the JAR into the correct Java installation directory for your OS and verify the shared library is on your library search path.

Add the wolfJCE JAR file (**wolfcrypt-jni.jar**) and shared library (**libwolfcryptjni.so**) to the following directory:

```
$JAVA_HOME/jre/lib/ext directory
```

For example, on Ubuntu with OpenJDK this may be similar to:

```
/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/ext
```

Next, add an entry to the java.security file that looks similar to the following:

```
security.provider.N=com.wolfssl.provider.jce.WolfCryptProvider
```

The java.security file will be located at:

```
$JAVA_HOME/jre/lib/security/java.security
```

Replacing "N" with the order of precedence you would like the WolfCryptProvider to have in comparison to other providers in the file.

5   Package Design

wolfJCE is bundled together with the "wolfcrypt-jni" JNI wrapper library. Since wolfJCE depends on the underlying JNI bindings for wolfCrypt, it is compiled into the same native library file and JAR file as wolfcrypt-jni.

For users wanting to use only the JNI wrapper, it is possible to compile a version of "wolfcrypt-jni.jar" that does not include the JCE provider classes.

wolfJCE / wolfCrypt JNI package structure:

```
wolfcrypt-jni /
AUTHORS
build.xml                           ant build script
COPYING
docs /                              Javadocs
jni /                               native C JNI binding source files
lib /                               output directory for compiled library
LICENSING
Makefile generic                    Makefile
Makefile.linux                      Linux-specific Makefile
Makefile.osx                        OSX-specific Makefile
README_JCE.md
README.md
src /
    main/java/                      Java source files
    test/java/                      Test source files
```

The wolfJCE provider source code is located in the "src/main/java/com/wolfssl/provider/jce" directory, and is part of the "**com.wolfssl.provider.jce**" Java package.

The wolfCrypt JNI wrapper is located in the "src/main/java/com/wolfssl/wolfcrypt" directory and is part of the "**com.wolfssl.wolfcrypt**" Java package. Users of JCE will not need to use this package directly, as it will be consumed by the wolfJCE classes.

Once wolfCrypt-JNI and wolfJCE have been compiled, the output JAR and native shared library are located in the "./lib" directory. Note, these contain BOTH the wolfCrypt JNI wrapper as well as the wolfJCE provider when a JCE build is compiled.

```
lib/
    libwolfcryptjni.so
    wolfcrypt-jni.jar
```

6   Supported Algorithms and Classes

wolfJCE currently supports the following algorithms and classes:

```
MessageDigest Class
    MD5
    SHA-1
    SHA-256
    SHA-384
    SHA-512

SecureRandom Class
    HashDRBG

Cipher Class
    AES/CBC/NoPadding
    DESede/CBC/NoPadding
    RSA/ECB/PKCS1Padding

Mac Class
    HmacMD5
    HmacSHA1
    HmacSHA256
    HmacSHA384
    HmacSHA512

Signature Class
    MD5withRSA
    SHA1withRSA
    SHA256withRSA
    SHA384withRSA
    SHA512withRSA
    SHA1withECDSA
    SHA256withECDSA
    SHA384withECDSA
    SHA512withECDSA

KeyAgreement Class
    DiffieHellman
    DH
    ECDH

KeyPairGenerator Class
    EC
    DH
```

7   JAR Code Signing

The Oracle JDK/JVM require that JCE providers be signed by a code signing certificate that has been issued by Oracle. The wolfcrypt-jni package's ant build script supports code signing the generated "wolfcrypt-jni.jar" file by placing a custom properties file in the root of the package directory before compilation.

To enable automatic code signing, create a file called "codeSigning.properties" and place it in the root of the "wolfcrypt-jni" package. This is a properties file which should include the following:

```
sign.alias=<signing alias in keystore>
sign.keystore=<path to signing keystore>
sign.storepass=<keystore password>
sign.tsaurl=<timestamp server url>
```

When this file is present when "ant" or "ant test" is run, it will sign "wolfcrypt-jni.jar" using the keystore and alias provided.

7.1   Using a Pre-Signed JAR File

wolfSSL (company) has it's own set of code signing certificates from Oracle that allow wolfJCE to be authenticated in the Oracle JDK. With each release of wolfJCE, wolfSSL ships a couple pre-signed versions of the 'wolfcrypt-jni.jar", located at:

wolfcrypt-jni-X.X.X/lib/signed/debug/wolfcrypt-jni.jar wolfcrypt-jni-X.X.X/lib/signed/release/wolfcrypt-jni.jar

This pre-signed JAR can be used with the JUnit tests, without having to re-compile the Java source files. To run the JUnit tests against this JAR file:

$ cd wolfcrypt-jni-X.X.X $ cp ./lib/signed/release/wolfcrypt-jni.jar ./lib $ ant test

8   Usage

For usage, please follow the Oracle/OpenJDK Javadocs for the classes specified in Chapter 6 above. Note that you will need to explicitly request the "wolfJCE" provider if it has been set lower in precedence than other providers that offer the same algorithm in the java.security file. For example, to use the wolfJCE provider with the MessageDigest class for SHA-1 you would create a MessageDigest object like so:

```
MessageDigest md = MessageDigest.getInstance"(SHA"-1, ""wolfJCE);
```

Please email support@wolfssl.com with any questions or feedback.