# Signing JAR Files with IAIK-CMS

SIC/IAIK, Graz University of Technology
July 2011

## Introduction

In this article, we show how to sign a JAR file using the IAIK-CMS library. The example program provides its own classes for generating the manifest file (MANIFEST.MF), signature info file (*.SF) and the signature file (*.RSA,*.DSA or *.ECDSA) inside the archive. It can sign any JAR file. This is useful for applications that need to sign JAR files programmatically and cannot access external tools like SUN's jarsigner tool ([3]).

We will start explaining how signing JAR files works in principle. Following, we provide pieces of source code that show the core signing functionality that uses IAIK-CMS. The c complete source code of the IAIK JAR signing tool is included in the customer version of the IAIK-CMS toolkit.

## The Process of JAR Signing

A JAR file is actually a ZIP file with some additional meta-files in a dedicated directory inside the archive. Usually, this META-INF directory contains at least a manifest file called MANIFEST.MF. This manifest is a text file that contains information about the contents of the archive. Often, it only contains information about the version of the manifest specification that this file follows and about the tool that created the archive. You can try opening any JAR file with your favorite ZIP tool, e.g. with WinZIP. Then, display the contents of the manifest file. If the jar-tool of JDK 6.0 created the file, you would see something like this:

```
Manifest-Version: 1.0
Created-By: 1.6.0 (Sun Microsystems Inc.)
```

In addition, a manifest can hold a list of all files in the archive. Each such manifest entry consists of a list of attribute-value pairs, with one such assertion per line. Blank lines separate the individual entries.

A manifest of an archive that contains two files may look like this.

```
Manifest-Version: 1.0
Created-By: 1.1 (IAIK Jar Signer)

Name: iaik/apps/jarsigner/JarSigner.class
SHA1-Digest: uEHvlBbWNCr9vpkMImMgZ5g7EbM=

Name: iaik/apps/jarsigner/Manifest.class
SHA1-Digest: WSsnWP8K6OTCYH1fFtfj68fiKNk=
```

We can see that each entry has a `Name` attribute and a `SHA1-Digest` attribute. The name refers to the name of the file in the archive and the digest attribute contains the SHA-1 hash of this file as a base-64 encoded value. The signature can only protect files of the archive that have such an entry.

The input to the signing process is not the manifest file. The input is a signature file, which looks very similar to the manifest. This file resides in the META-INF directory and has the file extension SF. A key alias usually serves as the base name of this file, but in general, it is arbitrary. Based on the previous manifest example, the contents of a signature file would look like the following:

```
Signature-Version: 1.0
SHA1-Digest-Manifest: 8Uy3LAro+t/2Cg9Udi96EGRihhw=
Created-By: 1.0 (IAIK Jar Signer)

Name: iaik/apps/jarsigner/JarSigner.class
SHA1-Digest: PZgxay+VQHGI3RO6S9ooZhhL9gg=

Name: iaik/apps/jarsigner/Manifest.class
SHA1-Digest: kEJh1iJQjLaViU0Y4bdEI0JOols=
```

The name attribute is the same as before. However, you might wonder why the hash values of the entries are different to those of the manifest, even though the referenced classes are the same. Instead of using the hash of the class files, the signing tool calculates the hash over the corresponding entries of the manifest file. For example, to get the hash value `PZgxay+VQHGI3RO6S9ooZhhL9gg=` for the `JarSigner` class, the tool hashes these lines of the manifest (including the trailing blank line)

```
Name: iaik/apps/jarsigner/JarSigner.class
SHA1-Digest: uEHvlBbWNCr9vpkMImMgZ5g7EbM=
```

The complete SF file is the input of the signing process, which produces a digital signature in PKCS#7 format. Please note that CMS is the successor standard of PKCS#7,

and both are compatible in our context. This signature does not contain the signed data, which is sometimes called external, explicit or detached signature. The signer tool stores this digital signature in the META-INF directory and uses the same base name as for the signature file. The extension is RSA, DSA or ECDSA, depending on the signature algorithm.

The signed JAR file can be verified using other tools like SUN's JAR Signing and Verification tool ([3]).

For further details about the JAR file format, please refer to [1].

## Creating a JAR Signature with IAIK-CMS

Once we have the signature file (SF), we need the signature key and the associated certificate chain. The following code fragment shows the required steps to create a digital signature, which is the content of the file with the RSA extension. This code uses the IAIK-CMS library ([2]).

```
X509Certificate[] certificateChain = ...
iaik.x509.X509Certificate[] iaikCertificateChain =
  Util.convertCertificateChain(certificateChain);
IssuerAndSerialNumber issuerSerial =
  new IssuerAndSerialNumber(iaikCertificateChain[0]);
AlgorithmID signatureAlg = AlgorithmID.rsaEncryption;
AlgorithmID digestAlg = AlgorithmID. sha1;
SignerInfo signerInfo =
  new SignerInfo(issuerSerial, digestAlg, signatureAlg, signatureKey);
byte[] signatureFileContent = ... // the content of the *.SF file
SignedData signedData =
  new SignedData(signatureFileContent, SignedData.EXPLICIT);
signedData.setCertificates(iaikCertificateChain);
signedData.addSignerInfo(signerInfo);
byte[] digitalSignature = new ContentInfo(signedData).getEncoded();
```

## Signing a JAR file

For signing a jar file you must create a `JarSignStream` with the jar file to be signed, the signer key and the signer certificate chain:

```
// add IAIK provider
IAIK.addAsProvider();

// create jar file to be signed
```

```
JarFile jarFileToBeSigned = new JarFile(jarFileName_);
// signer key
PrivateKey signerKey = ...;
// signer certificate chain
X509Certificate[] signerChain = ...;
// name ("alias") of the signer key
String signerKeyName = ...;
```

Subsequently write the JarSignStream to the file you want to store the signed jar file:

```
// create a JarSignStream
JarSignStream jarSignStream =
  new JarSignStream(jarFileToBeSigned, signerKey, signerChain,
                    signerKeyName);
// write to file
String signedJarFile = ...;
FileOutputStream output = new FileOutputStream(signedJarFile);
jarSignStream.writeTo(output);
output.flush();
output.close();
```

See the JarSigner source for a demo example. It also can be used to sign jar files with a key contained in a HSM or smardcard (using the IAIK PKCS#11 provider for accessing the hardware token).

## Summary

This document describes the process of signing a JAR and explains the most important contents of the manifest file, the signature file and the digital signature. Moreover, it gives details about their construction. Finally, it shows how to calculate the digital signature by using the IAIK-CMS library. With the code from the example program for this article, an application can sign JAR files programmatically.

## References


1. JAR File Specification
   http://download.oracle.com/javase/1.5.0/docs/guide/jar/jar.html
2. IAIK-CMS Toolkit (containing the IAIK JarSigner tool)
   http://jce.iaik.tugraz.at/products/communication_messaging_security/cms_s_mime
3. JAR Signing and Verification Tool
   http://download.oracle.com/javase/1.5.0/docs/tooldocs/windows/jarsigner.html