The Java™ Tutorials

Trail: Deployment
Lesson: Packaging Programs in JAR Files
Section: Signing and Verifying JAR Files

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

Understanding Signing and Verification

The Java™ platform enables you to digitally sign JAR files. You digitally sign a file for the same reason you might sign a paper document with pen and ink -- to let readers know that you wrote the document, or at least that the document has your approval.

When you sign a letter, for example, everyone who recognizes your signature can confirm that you wrote the letter. Similarly when you digitally sign a file, anyone who "recognizes" your digital signature knows that the file came from you. The process of "recognizing" electronic signatures is called *verification*.

When the JAR file is signed, you also have the option of time stamping the signature. Similar to putting a date on a paper document, time stamping the signature identifies when the JAR file was signed. The time stamp can be used to verify that the certificate used to sign the JAR file was valid at the time of signing.

The ability to sign and verify files is an important part of the Java platform's security architecture. Security is controlled by the security *policy* that's in force at runtime. You can configure the policy to grant security privileges to applets and to applications. For example, you could grant permission to an applet to perform normally forbidden operations such as reading and writing local files or running local executable programs. If you have downloaded some code that's signed by a trusted entity, you can use that fact as a criterion in deciding which security permissions to assign to the code.

Once you (or your browser) have verified that an applet is from a trusted source, you can have the platform relax security restrictions to let the applet perform operations that would ordinarily be forbidden. A trusted applet can have freedoms as specified by the *policy file* in force.

The Java platform enables signing and verification by using special numbers called public and private *keys*. Public keys and private keys come in pairs, and they play complementary roles.

The private key is the electronic "pen" with which you can sign a file. As its name implies, your private key is known only to you so that no one else can "forge" your signature. A file signed with your private key can be verified only by the corresponding public key.

Public and private keys alone, however, aren't enough to truly verify a signature. Even if you've verified that a signed file contains a matching key pair, you still need some way to confirm that the public key actually comes from the signer that it purports to come from.

One more element, therefore, is required to make signing and verification work. That additional element is the *certificate* that the signer includes in a signed JAR file. A certificate is a digitally signed statement from a recognized *certification authority* that indicates who owns a particular public key. Certification authorities are entities (typically firms specializing in digital security) that are trusted throughout the industry to sign and issue certificates for keys and their owners. In the case of signed JAR files, the certificate indicates who owns the public key contained in the JAR file.

When you sign a JAR file your public key is placed inside the archive along with an associated certificate so that it's easily available for use by anyone wanting to verify your signature.

To summarize digital signing:

- The signer signs the JAR file using a private key.
- The corresponding public key is placed in the JAR file, together with its certificate, so that it is available for use by anyone who wants to verify the signature.

Digests and the Signature File

When you sign a JAR file, each file in the archive is given a digest entry in the archive's manifest. Here's an example of what such an entry might look like:

```
Name: test/classes/ClassOne.class
SHA1-Digest: TD1GZt8G11dXY2p4olSZPc5Rj64=
```

The digest values are hashes or encoded representations of the contents of the files as they were at the time of signing. A file's digest will change if and only if the file itself changes.

When a JAR file is signed, a *signature* file is automatically generated and placed in the JAR file's `META-INF` directory, the same directory that contains the archive's manifest. Signature files have filenames with an `.SF` extension. Here is an example of the contents of a signature file:

```
Signature-Version: 1.0
SHA1-Digest-Manifest: h1yS+K9T7DyHtZrtI+LxvgqaMYM=
Created-By: 1.7.0_06 (Oracle Corporation)

Name: test/classes/ClassOne.class
SHA1-Digest: fcav7ShIG6i86xPepmitOVo4vWY=

Name: test/classes/ClassTwo.class
SHA1-Digest: xrQem9snnPhLySDiZyclMlsFdtM=

Name: test/images/ImageOne.gif
SHA1-Digest: kdHbE7kL9ZHLgK7akHttYV4XIa0=

Name: test/images/ImageTwo.gif
SHA1-Digest: mF0D5zpk68R4oaxEqoS9Q7nhm60=
```

As you can see, the signature file contains digest entries for the archive's files that look similar to the digest-value entries in the manifest. However, while the digest values in the manifest are computed from the files themselves, the digest values in the signature file are computed from the corresponding entries in the manifest. Signature files also contain a digest value for the entire manifest (see the `SHA1-Digest-Manifest` header in the above example).

When a signed JAR file is being verified, the digests of each of its files are re-computed and compared with the digests recorded in the manifest to ensure that the contents of the JAR file haven't changed since it was signed. As an additional check, digest values for the manifest file itself are re-computed and compared against the values recorded in the signature file.

You can read additional information about signature files on the Manifest Format page of the JDK™ documentation.

The Signature Block File

In addition to the signature file, a *signature block* file is automatically placed in the `META-INF` directory when a JAR file is signed. Unlike the manifest file or the signature file, signature block files are not human-readable.

The signature block file contains two elements essential for verification:

- The digital signature for the JAR file that was generated with the signer's private key
- The certificate containing the signer's public key, to be used by anyone wanting to verify the signed JAR file

Signature block filenames typically will have a `.DSA` extension indicating that they were created by the default Digital Signature Algorithm. Other filename extensions are possible if keys associated with some other standard algorithm are used for signing.

Related Documentation

For additional information about keys, certificates, and certification authorities, see

- The JDK Security Tools
- X.509 Certificates

For more information about the Java platform's security architecture, see this related documentation:

- Security Features in Java SE
- Java SE Security
- Security Tools

Previous page: Signing and Verifying JAR Files
Next page: Signing JAR Files