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.

Signing JAR Files

You use the JAR Signing and Verification Tool to sign JAR files and time stamp the signature. You invoke the JAR Signing and Verification Tool by using the `jarsigner` command, so we'll refer to it as "Jarsigner" for short.

To sign a JAR file, you must first have a private key. Private keys and their associated public-key certificates are stored in password-protected databases called *keystores*. A keystore can hold the keys of many potential signers. Each key in the keystore can be identified by an *alias* which is typically the name of the signer who owns the key. The key belonging to Rita Jones might have the alias "rita", for example.

The basic form of the command for signing a JAR file is

```
jarsigner jar-file alias
```

In this command:

- `jar-file` is the pathname of the JAR file that's to be signed.
- `alias` is the alias identifying the private key that's to be used to sign the JAR file, and the key's associated certificate.

The Jarsigner tool will prompt you for the passwords for the keystore and alias.

This basic form of the command assumes that the keystore to be used is in a file named `.keystore` in your home directory. It will create signature and signature block files with names `x.SF` and `x.DSA` respectively, where `x` is the first eight letters of the alias, all converted to upper case. This basic command will *overwrite* the original JAR file with the signed JAR file.

In practice, you might want to use one or more of the command options that are available. For example, time stamping the signature is encouraged so that any tool used to deploy your application can verify that the certificate used to sign the JAR file was valid at the time that the file was signed. A warning is issued by the Jarsigner tool if a time stamp is not included.

Options precede the `jar-file` pathname. The following table describes the options that are available:

Jarsigner Command Options

Option	Description
`-keystore` *url*	Specifies a keystore to be used if you don't want to use the `.keystore` default database.
`-sigfile` *file*	Specifies the base name for the .SF and .DSA files if you don't want the base name to be taken from your alias. *file* must be composed only of upper case letters (A-Z), numerals (0-9), hyphen (-), and underscore (_).
`-signedjar` *file*	Specifies the name of the signed JAR file to be generated if you don't want the original unsigned file to be overwritten with the signed file.
`-tsa` *url*	Generates a time stamp for the signature using the Time Stamping Authority (TSA) identified by the URL.
`-tsacert` *alias*	Generates a time stamp for the signature using the TSA's public key certificate identified by *alias*.
`-altsigner` *class*	Indicates that an alternative signing mechanism be used to time stamp the signature. The fully-qualified class name identifies the class used.
`-altsignerpath` *classpathlist*	Provides the path to the class identified by the `altsigner` option and any JAR files that the class depends on.

Example

Let's look at a couple of examples of signing a JAR file with the Jarsigner tool. In these examples, we will assume the following:

- Your alias is "johndoe".
- The keystore you want to use is in a file named "mykeys" in the current working directory.
- The TSA that you want to use to time stamp the signature is located at `http://tsa.url.example.com`.

Under these assumptions, you could use this command to sign a JAR file named `app.jar`:

```
jarsigner -keystore mykeys -tsa http://tsa.url.example.com app.jar johndoe
```

You will be prompted to enter the passwords for both the keystore and your alias. Because this command doesn't make use of the `-sigfile` option, the .SF and .DSA files it creates would be named `JOHNDOE.SF` and `JOHNDOE.DSA`. Because the command doesn't use the `-signedjar` option, the resulting signed file will overwrite the original version of `app.jar`.

Let's look at what would happen if you used a different combination of options:

```
jarsigner -keystore mykeys -sigfile SIG -signedjar SignedApp.jar
          -tsacert testalias app.jar johndoe
```

The signature and signature block files would be named `SIG.SF` and `SIG.DSA`, respectively, and the signed JAR file `SignedApp.jar` would be placed in the current directory. The original unsigned JAR file would remain unchanged. Also, the signature would be time stamped with the TSA's public key certificate identified as `testalias`.

Additional Information

Complete reference pages for the JAR Signing and Verification Tool are on-line: Summary of Security Tools

Note: When a certificate is self signed, `UNKNOWN` will be displayed as the publisher of the application. For more information, see Is it safe to run an application from a publisher that is listed as `UNKNOWN`?.

Previous page: Understanding Signing and Verification
Next page: Verifying Signed JAR Files