1. <https://blogs.oracle.com/java-platform-group/entry/self_signed_certificates_for_a>

### Self-signed certificates for a known community

#### By costlow on [Nov 11, 2013](https://blogs.oracle.com/java-platform-group/entry/self_signed_certificates_for_a)

[Recently announced changes scheduled for Java 7 update 51](https://blogs.oracle.com/java-platform-group/entry/new_security_requirements_for_rias) (January 2014) have established that the default security slider will require code signatures and the Permissions Manifest attribute. [Code signatures are a common practice recommended in the industry](https://casecurity.org/2013/10/16/securing-software-distribution-with-digital-code-signing/) because they help determine that the code your computer will run is the same code that the publisher created.

This post is written to help users that need to use self-signed certificates without involving a public Certificate Authority.

## The role of self-signed certificates within a known community

You may still use self-signed certificates within a known community. The difference between self-signed and purchased-from-CA is that your users must import your self-signed certificate to indicate that it is valid, whereas Certificate Authorities are already trusted by default.

This works for known communities where people will trust that my certificate is mine, but does not scale widely where I cannot actually contact or know the systems that will need to trust my certificate. Public Certificate Authorities are widely trusted already because they abide by [many different requirements and frequent checks](https://www.cabforum.org/documents.html).

An example would be students in a university class sharing their public certificates on a mailing list or web page, employees publishing on the intranet, or a system administrator rolling certificates out to end-users. Managed machines help this because you can automate the rollout, but they are not required -- the major point simply that people will trust and import your certificate.

## How to distribute self-signed certificates for a known community

There are several steps required to distribute a self-signed certificate to users so that they will properly trust it. These steps are:

1. Creating a public/private key pair for signing.
2. Exporting your public certificate for others
3. Importing your certificate onto machines that should trust you
4. Verify work on a different machine

## Creating a public/private key pair for signing

Having a public/private key pair will give you the ability both to sign items yourself and issue a Certificate Signing Request (CSR) to a certificate authority.

Create your public/private key pair by following the [instructions for creating key pairs](http://docs.oracle.com/javase/tutorial/security/toolsign/step3.html).  
Every Certificate Authority that I looked at provided similar instructions, but for the sake of cohesiveness I will include the commands that I used here:

1. Generate the key pair.  
   keytool -genkeypair -alias erikcostlow -keyalg EC -keysize 571 -validity 730 -keystore javakeystore\_keepsecret.jks
   * Provide a good password for this file.
   * The alias "erikcostlow" is my name and therefore easy to remember. Substitute your name of something like "mykey."
   * The sigalg of EC ([Elliptical Curve](http://arstechnica.com/security/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/)) and keysize of 571 will give your key a [good strong lifetime](http://www.keylength.com/en/4/).
   * All keys are set to expire. Two years or 730 days is a reasonable compromise between not-long-enough and too-long. Most public Certificate Authorities will sign something for one to five years.
   * You will be placing your keys in javakeystore\_keepsecret.jks -- this file will contain private keys and therefore should not be shared. If someone else gets these private keys, they can impersonate your signature. Please be cautious about automated cloud backup systems and private key stores.
2. Answer all the questions. It is important to provide good answers because you will stick with them for the "-validity" days that you specified above.  
   What is your first and last name?  
   [Unknown]: First Last  
   What is the name of your organizational unit?  
   [Unknown]: Line of Business  
   What is the name of your organization?  
   [Unknown]: MyCompany  
   What is the name of your City or Locality?  
   [Unknown]: City Name  
   What is the name of your State or Province?  
   [Unknown]: CA  
   What is the two-letter country code for this unit?  
   [Unknown]: US  
   Is CN=First Last, OU=Line of Business, O=MyCompany, L=City, ST=CA, C=US correct?  
   [no]: yes  
   Enter key password for <erikcostlow>  
   (RETURN if same as keystore password):
3. Verify your work:  
   keytool -list -keystore javakeystore\_keepsecret.jks  
   You should see your new key pair.

## Exporting your public certificate for others

Public Key Infrastructure relies on two simple concepts: the public key may be made public and the private key must be private. By [exporting your public certificate](http://docs.oracle.com/javase/tutorial/security/toolsign/step5.html), you are able to share it with others who can then import the certificate to trust you.

keytool -exportcert -keystore javakeystore\_keepsecret.jks -alias erikcostlow -file erikcostlow.cer

To verify this, you can open the .cer file by double-clicking it on most operating systems. It should show the information that you entered during the creation prompts.

This is the file that you will share with others. They will use this certificate to prove that artifacts signed by this certificate came from you. If you do not manage machines directly, place the certificate file on an area that people within the known community should trust, such as an intranet page.

## Import the certificate onto machines that should trust you

In order to trust the certificate, people within your known network must import your certificate into their keystores. The first step is to verify that the certificate is actually yours, which can be done through any band: email, phone, in-person, etc. Known networks can usually do this

Determine [the right keystore](http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/properties.html):

* For an individual user looking to trust another, the correct file is within that user’s directory.  
  e.g. USER\_HOME\AppData\LocalLow\Sun\Java\Deployment\security\trusted.certs  
  The default password for trusted.certs is an empty string.
* For system-wide installations, Java’s Certificate Authorities are in JAVA\_HOME  
  e.g. C:\Program Files\Java\jre8\lib\security\cacerts  
  The default password for cacerts is "changeit" as described in the [keytool documentation](http://docs.oracle.com/javase/7/docs/technotes/tools/windows/keytool.html).

File paths for Mac and Linux are included in the link above.

Follow the [instructions to import the certificate](http://docs.oracle.com/javase/tutorial/security/toolsign/rstep2.html) into the keystore.

keytool -importcert -keystore THEKEYSTOREFROMABOVE -alias erikcostlow -file erikcostlow.cer

In this case, I am still using my name for the alias because it’s easy for me to remember. You may also use an alias of your company name.

### Scaling distribution of the import

The easiest way to apply your certificate across many machines is to just push the .certs or cacerts file onto them. When doing this, watch out for any changes that people would have made to this file on their machines.

Trusted.certs: When publishing into user directories, your file will overwrite any keys that the user has added since last update.

CACerts: It is best to re-run the import command with each installation rather than just overwriting the file. If you just keep the same cacerts file between upgrades, you will overwrite any CAs that have been added or removed. By re-importing, you stay up to date with changes.

There is a -storepass argument for specifying the password on the command-line. On secure systems, admins typically hide passwords through redirection operators but that's outside the scope of this post.

## Verify work on a different machine

Verification is a way of checking on the client machine to ensure that it properly trusts signed artifacts after you have added your signing certificate. Many people have started using [deployment rule sets](https://blogs.oracle.com/java-platform-group/entry/introducing_deployment_rule_sets). You can validate the deployment rule set by:

1. Create and sign the deployment rule set on the computer that holds the private key.
2. Copy the deployment rule set on to the different machine where you have imported the signing certificate.
3. Verify that the Java Control Panel’s security tab shows your deployment rule set.

### Verifying an individual JAR file or multiple JAR files

You can test a certificate chain by using the jarsigner command.

jarsigner -verify filename.jar

If the output does not say "jar verified" then run the following command to see why:

jarsigner -verify -verbose -certs filename.jar

Check the output for the term “CertPath not validated.”

1. <https://blogs.oracle.com/java-platform-group/entry/new_security_requirements_for_rias>

### New security requirements for RIAs in 7u51 (January 2014)

#### By costlow on [Sep 09, 2013](https://blogs.oracle.com/java-platform-group/entry/new_security_requirements_for_rias)

Java 7 update 51 (January, 2014) intends to include two security changes designed to enhance authentication and authorization for Rich Internet Applications (Applets and Web Start). The default security slider is being updated in a way that will block RIAs that do not adhere to these requirements. **Note: this only applies to RIAs, and not to Java on server or desktop applications run outside of a browser.**

# Summary:

* You are required to sign all RIAs (Applets and Web Start applications).
* You are required to set the "Permissions" attribute within the Manifest.
* Your application will be affected if it uses Java started through a web browser. Your application will not be affected if it runs anywhere outside of a web browser.

Complete information can be found within the Java 7 update 51 release notes ([here once 7u51 is released in January 2014](http://www.oracle.com/technetwork/java/javase/7u-relnotes-515228.html)).

# Developers:

As of 7u51, (January 14, 2014), your RIAs must be updated. The updates required are on the packaging and distribution; no API code changes should be required. The impetus for these changes relates to [potential re-purposing of sandboxed applications](http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/no_redeploy.html), whereby placing permissions within a signed JAR prevents modification of your specified permission level.  
RIAs must contain two things:

1. Code signatures from a trusted authority. All code for Applets and Web Start applications must be signed, regardless of its Permissions attributes.
2. Manifest Attributes
   1. Permissions – Introduced in 7u25, and required as of 7u51. Indicates if the RIA should run within the sandbox or require full-permissions.
   2. Codebase – Introduced in 7u25 and optional/encouraged as of 7u51. Points to the known location of the hosted code (e.g. [intranet.example.com)](http://intranet.example.com).

## Sample META-INF/MANIFEST.MF file:

Manifest-Version: 1.0

Created-By: 1.7.0\_51

Permissions: sandbox

Codebase: www.java.com java.com

This manifest file is created when the JAR is packaged, either through the default [jar command](http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jar.html), your build tool, or your IDE.

## Sample JNLP for launching a web start application:

<?xml version="1.0" encoding="UTF-8"?>

<jnlp href="JavaDetection\_applet.jnlp">

<information>

<title>Java Detection</title>

<vendor>Oracle Inc.</vendor>

</information>

<resources>

<jar href="JavaDetection.jar" />

</resources>

<applet-desc

name="Java Detection Applet"

main-class="JavaDetection"

width="1"

height="1">

</applet-desc>

<update check="background"/>

</jnlp>

Some developers may notice that the <security /> element is missing from the above JNLP. By providing it within the cryptographically signed JAR file, it is no longer necessary within the JNLP for Java 7 update 51 (January 2014).

See the [Development and Deployment Of Rich Internet Applications](http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/index.html) for full details of JNLPs and the deployment toolkit. For instructions on signing code, see the tutorial [Lesson: Signing Code and Granting It Permissions](http://docs.oracle.com/javase/tutorial/security/toolsign/index.html).

# Desktop Administrators

If you are a desktop administrator managing Java installations across a series of computers, consider using [Deployment Rule Sets to whitelist your internal applications](https://blogs.oracle.com/java-platform-group/entry/introducing_deployment_rule_sets). Deployment Rule Sets allow you to certify that an application is known to be trusted and safe, even if you cannot update the application to adhere to these requirements.

1. <http://11idba.blogspot.ro/2013/12/use-java-code-signing-certificate-in.html>
2. **Create and Import Java Code Signing Certificate to use in Oracle E-Business Suite**
3. Ref: Oracle Note
4. Enhanced Jar Signing for Oracle E-Business Suite (Doc ID 1591073.1)
5. Oracle note is important document and covered most of the errors/resolutions. But there are short comings in the document .I did not use ADJAVA command and substituted with KEYTOOL command.
6. 1**) Generate KeyPair**
7. keytool -genkey -keyalg RSA -keysize 4096 -alias EbsJarCert -keypass EbsJarkp1 -storepass EbsJarsp1 -keystore /ebs/home/test/keystore.dat –v
8. What is your first and last name?
9. [Unknown]: EbsJarServ.com
10. What is the name of your organizational unit?
11. [Unknown]: Ebs
12. What is the name of your organization?
13. [Unknown]: Ebs Company
14. What is the name of your City or Locality?
15. [Unknown]: Washington
16. What is the name of your State or Province?
17. [Unknown]: District of Columbia
18. What is the two-letter country code for this unit?
19. [Unknown]: US
20. Is CN=EbsJarServ.com, OU=Ebs, O=Ebs Company, L=Washington, ST=District of Columbia, C=US correct?
21. [no]: yes
22. Generating 4,096 bit RSA key pair and self-signed certificate (SHA1withRSA) with a validity of 90 days
23. for: CN=EbsJarServ.com, OU=Ebs, O=Ebs Company, L=Washington, ST=District of Columbia, C=US
24. [Storing /ebs/app/keystore.dat]
25. **2) Verify KeyPair**
26. $ keytool -list –v -storepass EbsJarsp1-keystore /ebs/home/test/keystore.dat -alias EbsJarCert
27. ( You can also execute the same command with adjkey
28. adjkey -list -v -storepass EbsJarsp1 -keystore /ebs/home/test/keystore.dat -alias EbsJarCert)
29. *Alias name: EbsJarCert*
30. *Creation date: Dec 17, 2013*
31. *Entry type: PrivateKeyEntry*
32. *Certificate chain length: 1*
33. *Certificate[1]:*
34. *Owner: CN=EbsJarServ.com, OU=Ebs, O=Ebs Company, L=Washington, ST=District of Columbia, C=US*
35. *Issuer: CN=EbsJarServ.com, OU=Ebs, O=Ebs Company, L=Washington, ST=District of Columbia, C=US*
36. *Serial number: 52b06a4a*
37. *Valid from: Tue Dec 17 09:05:46 EST 2013 until: Mon Mar 17 10:05:46 EDT 2014*
38. *Certificate fingerprints:*
39. *MD5: 10:37:62:7D:60:EA:89:AF:15:77:19:62:59:49:28:DD*
40. *SHA1: 02:93:CC:96:BD:EF:CE:28:B9:38:51:8E:4B:F8:94:05:ED:C8:68:0A*
41. *Signature algorithm name: SHA1withRSA*
42. *Version: 3*
43. **3) Request Certificate (CSR)**
44. keytool -certreq -alias EbsJarCert \
45. -storepass EbsJarsp1 -keypass EbsJarkp1 \
46. -file /ebs/home/test/EbsJarCert.csr \
47. -keystore /ebs/home/test/keystore.dat
48. **4) Submit your CSR to Signing Authority for certificate (Verisign)**
49. **Note**: Be sure to ***request a Java Code Signing Certificate***. This certificate can be used to sign your jar content across one or mutliple Oracle E-Business Suite environmments.After you get the certificate, export in X509 format and ftp in ascii to web server.
50. **5) Verify the Certificate (CRT) after you receive from Verisign**
51. keytool -printcert -v -file /ebs/home/test/EbsJarCert.crt
52. **6) Import Intermediate and Root Certificate to the cacerts Keystore File**
53. Create myca.crt ( by exporting Intermediate and Root Certificate in X509 format)
54. Go to = $OA\_JRE\_TOP/jre/lib/security/
55. chmod +w cacerts
56. cat inter.crt root.crt> myca.crt
57. keytool -import -alias myca -file myca.crt -trustcacerts -v -keystore cacerts
58. chmod a-w cacerts
59. Note: The default Java for the cacerts keystore certificates file is usually ***changeit***.
60. **7) Import the Java Code Signing Certificate into the Keystore**
61. *= $APPL\_TOP/admin*
62. Create EbsJarCert.crt ( by exporting .crt in X509 format)
63. cp EbsJarCert.crt $APPL\_TOP/admin/adkeystore.crt
64. cp /ebs/home/test/keystore.dat $APPL\_TOP/admin/adkeystore.dat
65. *Import the certificate*
66. keytool -import \
67. -file /ebs/app/test/ appl/admin/adkeystore.crt \
68. -trustcacerts -storepass EbsJarsp1
69. -alias EbsJarCert \
70. -keystore /ebs/app/test / appl/admin/adkeystore.dat
72. **8) Verify and Update jripasswords in the database**
73. Login as APPS
74. SQL>
75. set serveroutput on
76. declare
77. spass varchar2(30);
78. kpass varchar2(30);
79. begin
80. ad\_jar.get\_jripasswords(spass, kpass);
81. dbms\_output.put\_line('--------');
82. dbms\_output.put\_line('STOREPASS = '||spass);
83. dbms\_output.put\_line('KEYPASS = '||kpass);
84. dbms\_output.put\_line('--------');
85. end;
86. /
87. If STOREPASS and KEYPASS does not match to adkeystore.dat then update with correct passwords
88. SQL> exec ad\_jar.DEL\_JRIPASSWORDS;
89. SQL> exec ad\_jar.PUT\_JRIPASSWORDS('EbsJarsp1',' EbsJarkp1');
90. **9) Generate JAR files with ADADMIN**
91. vi $APPL\_TOP/admin/adsign.txt
92. **EbsJarCert 1 CUST**
93. ***adadmin***
94. Choose Generate Applications Files menu
95. From this menu choose Generate product JAR files
96. Enter yes when prompted with: Do you wish to force regeneration of all jar files? [No] ? ***yes***
97. **10) Verify the Digital Signature of a Signed Jar File**
98. jarsigner -verify -verbose -certs /ebs/app/test /comn/java/oracle/apps/fnd/jar/fndlist.jar
99. X.509, CN=EbsJarServ.com, OU=Digital ID Class 3 - Java Object Signing, OU=Ebs, O=Ebs Company, L=Washington, ST=District of Columbia, C=US
100. [certificate is valid from 12/17/13 7:00 PM to 12/16/15 6:59 PM]
101. jar verified.
102. **11) Test the Applications**
103. Java Control Panel -> Security -> Security Level: Very High
104. Java Control Panel -> Advanced -> Show Console (Check)
105. Java Control Panel -> Advanced -> Enable logging (Check)
106. Java Control Panel -> Advanced -> Enable Trace (Check)
107. Login to Applications and test
108. Verify output In Java Console for any issues/errors/warning.
109. Some of them are
110. *Missing Permissions manifest attribute for*:
111. https://11iurl/OA\_JAVA/oracle/apps/fnd/jar/fndutil.jar
112. *Missing Codebase manifest attribute for*:
113. https:// https://11iurl /OA\_JAVA/oracle/apps/fnd/jar/fndutil.jar
114. Fix
115. All the JAR files are not signed properly , run the following commands
116. adjava -mx512m -nojit oracle.apps.ad.jri.adjcopy -masterArchive $JAVA\_TOP -sync -reportfile javatopfiles.lst
117. adjava -mx512m -nojit oracle.apps.ad.jri.adjcopy -masterArchive $JAVA\_TOP -sync -mode APPLY
118. Run ADADMIN again and force generate JAR files
119. For error in java console  
     network: Connecting http://ocsp.verisign.com/ with proxy=DIRECT  
     network: Connecting http://ocsp.verisign.com:80/ with proxy=DIRECT  
     security: Failing over to CRLs: java.net.SocketTimeoutException: connect timed out  
     network: Cache entry not found [url: http://crl.verisign.com/pca3-g5.crl, version: null]  
     network: Connecting http://crl.verisign.com/pca3-g5.crl with proxy=DIRECT  
     network: Connecting http://crl.verisign.com:80/ with proxy=DIRECT
120. Fix
121. Control Panel->Java->Advanced' Tab->'Perform certificate revocation checks on' section - Select “DO NOT CHECK”

4. <https://blogs.oracle.com/stevenChan/entry/sign_e_business_suite_jar>

### Sign E-Business Suite JAR Files Now

#### By Steven Chan (Oracle Development) on [Oct 18, 2013](https://blogs.oracle.com/stevenChan/entry/sign_e_business_suite_jar)

Oracle E-Business Suite uses Java, notably for running Forms-based content via the Java Runtime Environment (JRE) browser plug-in.   
  
The default security settings for the JRE plug-in are expected to become more stringent over time. To prepare for upcoming changes to Java security, **all** EBS 11i, 12.0, 12.1, and 12.2 system administrators **must** follow the procedures documented here:

* [Enhanced Signing of Oracle E-Business Suite JAR Files](https://support.oracle.com/rs?type=doc&id=1591073.1) (Note 1591073.1 )

More information about Java security is available here:

* [Security of the Java Platform](http://www.oracle.com/technetwork/java/javase/overview/security-2043272.html)