Articles

Creating an SSL Keystore Using the Java Keytool

By Jon Svede

Date: Aug 19, 2005

Return to the article

Many Java application servers and Web servers support the use of keystores for SSL configuration. If you're building secure Java programs, learning to build a keystore is, well, key. Jon Svede shows you how to accomplish it using the Java keytool utility with BEA's WebLogic 8.1.

One of the easiest ways to secure an Internet application is to use the Secure Socket Layer (SSL) protocol. SSL allows the data from a client, such as a Web browser, to be encrypted prior to transmission so that someone trying to sniff the data is unable to decipher it.

SSL is used anywhere data needs to be secure. Typically, you see SSL used by banks and financial institutions, as well as on your favorite commercial e-commerce sites, like amazon.com or bestbuy.com. You may also see it at work; your employer may have a Web site for your personal information which is usually protected with SSL.

Many application servers and Web servers support the use of keystores for SSL configuration; WebLogic Server and JBoss both support the use of keystores for SSL. If you want to deploy SSL in your Java applications, building a keystore is one of the first steps. In this article, I show you one method to create a keystore, using the Java `keytool` utility with BEA's WebLogic 8.1.

The Necessities: Private Keys, Public Certificates, Certificate Authorities, and Keystores

Let's start with a very brief overview of the technology.

NOTE

You'll find loads more introductory security information in the Informit.com Security Reference Guide.

SSL works by allowing the client and server to send each other encrypted data that only they can decrypt. That's accomplished when the connection between the client and server is established. In a sequence of events generally known as the *handshake*, the server presents its public certificate for the client to accept or deny. If the certificate is accepted, the client and server agree on a *hash* for the duration of their conversation, which is used to encrypt the data.

The world of SSL has, essentially, three types of certificates: private keys, public keys (also called public certificates or site certificates), and root certificates. The private key contains the identity information of the server, along with a key value. It is critical to keep this information private, because it's used to negotiate the hash during the handshake -- rather like leaving your house key in the door lock. It is possible for someone to use your key to decrypt traffic – not all that *likely*, but a good reason to keep your private keys safe.

The public certificate is the portion that is presented to a client. The public certificate, tightly associated to the private key, is created from the private key using a *Certificate Signing Request* (CSR). After you create a private key, you create a CSR, which is sent to your Certificate Authority (CA). The CA returns a signed certificate, which has information about the server identity and about the CA.

Every CA provides a *root certificate*. A root certificate is one where the Issuer and the Subject are the exact same string. It's actually more involved than that, but that's as much as you need for the purposes of this article.

A Certificate Authority can take several forms. You may be familiar with several of these, such as VeriSign, Thawte, Commodo, GetTrust, and dozens of other companies who will sign certificates for you. Additionally, many companies and institutions act as their own CA, either by building a complete implementation from scratch, or by using an open source option, such as OpenSSL.

NOTE

In this article I use VeriSign. They will sign a temporary certificate valid for 14 days; that makes it easier for you to follow along. Doing so also lets you set up your site, then get your real certificate from Verisign.

The final element that must be mentioned is the *keystore*, a special file type that can hold your keys and certificates and encrypt it all with a password. In simple terms, the keystore is just like a hashtable; it has an alias that identifies a certificate and then the certificate itself. You can access a specific certificate by specifying its alias.

Chaining and Establishing Identities

When a server and client establish an SSL connection, a certificate is presented to the client; the client must determine whether to

Articles

trust this certificate, a process called the *certificate chain*. The client examines the issuer of a certificate, searches its list of trusted root certificates, and compares the issuer on the presented certificate to the subjects of the trusted certificates. Okay, it is more involved than that, but this is as much as you need to create a Java SSL connection.

If a match is found, the connection proceeds. If not, the Web browsers may pop up a dialog box, warning you that it cannot trust the certificate and offering the option to trust the certificate. However, if your code is making a connection, the certificate is rejected. In most cases, it raises an exception; you can choose to handle and continue, or log the exception and exit.

Okay, enough background. Let's put this to use by building a keystore.

Generating a Private Key and a Keystore

To generate a keystore, you need a JDK installed with its `/bin` directory in your path. You can verify this setup by typing `keytool` at the shell prompt. You should see the help text, similar to this (which is an opportunity for you to peek at the options, in case you like to read ahead):

```
C:\>\bea_81sp4\jrockit81_142_05\bin\keytool
keytool usage:

-certreq   [-v] [-alias <alias>] [-sigalg <sigalg>]
       [-file <csr_file>] [-keypass <keypass>]
       [-keystore <keystore>] [-storepass <storepass>]
       [-storetype <storetype>] [-provider <provider_class_name>] ...

-delete   [-v] -alias <alias>
       [-keystore <keystore>] [-storepass <storepass>]
       [-storetype <storetype>] [-provider <provider_class_name>] ...

-export   [-v] [-rfc] [-alias <alias>] [-file <cert_file>]
       [-keystore <keystore>] [-storepass <storepass>]
       [-storetype <storetype>] [-provider <provider_class_name>] ...

-genkey   [-v] [-alias <alias>] [-keyalg <keyalg>]
       [-keysize <keysize>] [-sigalg <sigalg>]
       [-dname <dname>] [-validity <valDays>]
       [-keypass <keypass>] [-keystore <keystore>]
       [-storepass <storepass>] [-storetype <storetype>]
       [-provider <provider_class_name>] ...

-help

-identitydb [-v] [-file <idb_file>] [-keystore <keystore>]
       [-storepass <storepass>] [-storetype <storetype>]
       [-provider <provider_class_name>] ...

-import   [-v] [-noprompt] [-trustcacerts] [-alias <alias>]
       [-file <cert_file>] [-keypass <keypass>]
       [-keystore <keystore>] [-storepass <storepass>]
       [-storetype <storetype>] [-provider <provider_class_name>] ...

-keyclone  [-v] [-alias <alias>] -dest <dest_alias>
       [-keypass <keypass>] [-new <new_keypass>]
       [-keystore <keystore>] [-storepass <storepass>]
       [-storetype <storetype>] [-provider <provider_class_name>] ...

-keypasswd  [-v] [-alias <alias>]
       [-keypass <old_keypass>] [-new <new_keypass>]
       [-keystore <keystore>] [-storepass <storepass>]
       [-storetype <storetype>] [-provider <provider_class_name>] ...

-list    [-v | -rfc] [-alias <alias>]
       [-keystore <keystore>] [-storepass <storepass>]
       [-storetype <storetype>] [-provider <provider_class_name>] ...

-printcert  [-v] [-file <cert_file>]

-selfcert  [-v] [-alias <alias>] [-sigalg <sigalg>]
       [-dname <dname>] [-validity <valDays>]
       [-keypass <keypass>] [-keystore <keystore>]
       [-storepass <storepass>] [-storetype <storetype>]
       [-provider <provider_class_name>] ...

-storepasswd [-v] [-new <new_storepass>]
       [-keystore <keystore>] [-storepass <storepass>]
       [-storetype <storetype>] [-provider <provider_class_name>] ...
```

This output comes from BEA's version of the JVM called JRockit. However, the output would be identical if I had used Sun's version.

We use only a few of these options here, the first of which is `-genkey`, which generates the private key. The `-genkey` option can also

add the private key to an existing keystore; or, if the keystore specified does not exist, it create a new one.

To create a new keystore, you issue the following command at the command prompt:

```
keytool -genkey -alias servercert -keyalg RSA -keysize 1024 -dname "CN=jsvede.bea.com,OU=DRE,O=BEA,L=Der
```

The alias is the unique identifier of this entry. I recommend that you name it something specific to your server or application, but it can be anything you want.

The `-keyalg` is the algorithm that is used to generate the key. You can read about the other values for this argument on Sun's Web site. Most people choose RSA.

The next argument, `-dname`, is the Distinguished Name (DN). It contains the server identity, called the Common Name (CN), as well as other relevant information about your Organizational Unit (OU), Oranization(O), Locality (L), State (S) and Country (C). In WebLogic 8.1, you can use nearly any ASCII characters you want in this string, but in prior versions the SSL implementation was not as flexible.

The `-keypass` is the password for your private key; it is required for obvious reasons. Next, you specify the keystore name and the keystore password. Again, I recommend that you use a password for this and that it be something cryptic. Also, I recommend using a different password than the private key password.

When you run this command, it produces no output:

```
D:\ssl-article\examples>keytool -genkey -alias servercert -keyalg RSA -keysize 1024 -dname "CN=jsvede.be
US" -keypass weblogic1234 -keystore server_keystore.jks -storepass weblogic1234

D:\ssl-article\examples>dir
 Volume in drive D is DATA
 Volume Serial Number is 80F6-E25B

 Directory of D:\ssl-article\examples

03/04/2005 07:35p   <DIR>     .
03/04/2005 07:35p   <DIR>     ..
03/04/2005 07:35p         1,353 server_keystore.jks
        1 File(s)     1,353 bytes
        2 Dir(s) 12,589,309,952 bytes free

D:\ssl-article\examples>
```

You can verify the contents of your keystore, which at this time is only the private key, using this command:

```
D:\ssl-article\examples>keytool -list -v -keystore server_keystore.jks -storepass weblogic1234

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: servercert
Creation date: Mar 4, 2005
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=jsvede.bea.com, OU=DRE, O=BEA, L=Denver, ST=Colorado, C=US
Issuer: CN=jsvede.bea.com, OU=DRE, O=BEA, L=Denver, ST=Colorado, C=US
Serial number: 42291b03
Valid from: Fri Mar 04 19:35:47 MST 2005 until: Thu Jun 02 20:35:47 MDT 2005
Certificate fingerprints:
     MD5: D4:55:EA:25:FF:1A:1C:22:F5:3E:76:53:36:96:CF:93
     SHA1: 82:3F:73:37:A5:B9:A0:24:F4:E4:CA:0F:E8:A9:0B:CB:41:2F:F0:29
```

The next step is to generate the Certificate Signing Request (CSR). To do so, issue the following command:

```
D:\ssl-article\examples>keytool -certreq -v -alias servercert -file csr-for-myserver.pem -keypass weblo
server_keystore.jks
Certification request stored in file <csr-for-myserver.pem>
Submit this to your CA
```

As the command output suggests, you should now find a file called `csr-for-myserver.pem` in the current directory. Its contents look something like this:

```
D:\ssl-article\examples>more csr-for-myserver.pem
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBpjCCAQ8CAQAwZjELMAkGA1UEBhMCVVMxETAPBgNVBAgTCENvbG9yYWRvMQ8wDQYDVQQHEwZE
ZW52ZXIxDDAKBgNVBAoTA0JFQTEMMAoGA1UECxMDRFJFMRcwFQYDVQQDEw5qc3ZlZGUuYmVhLmNv
bTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA3hwTyXG2gWGUTMKMTeIkUgYzPJJwboJQdEIc
VbkHuzEnAnymz+402R2lR/mJjSCo1m+Wv3s5JvGZ9d2EfGC9ZpDG3tZGxqWomhAOBwyVBgcCmFOj
hv6rgrRO0NrO5KOyqJQBAHKHlj7M9hYD6Nml4M0wXqg43+9jR2NnTNLdMzsCAwEAAaAAMA0GCSqG
SIb3DQEBBAUAA4GBAIJ67O+gCe8KqyGKiKdcBfqSEGFQYBxJrnM9It9NcOn723Rt4OHvcz0a3Qz0
6ZutPzH0PbVxDIaG1jvy7Bew6khrWVxt6xX2gndxbx1qwDwW1e1VyRrGwP1ZXKZDzaJptlz+7P8B
3jyIgC4HsM5poBfSLtn3xX3Zy1tQkDmmiyL7
-----END NEW CERTIFICATE REQUEST-----
```

Not very enlightening, is it? It isn't supposed to be pretty. This is the encrypted information you specified when creating the private key. Don't worry though; your certificate authority should have no trouble deciphering it.

This file is what you submit to your CA for signing.

In this case, I use VeriSign's Trial Certificate Authority. You fill out various fields about yourself and your organization. This is all for business purposes; the actual information about your certificates are already encoded into the CSR. With VeriSign, after you've completed the online stuff, you receive an e-mail message with an entry like this:

```
-----BEGIN CERTIFICATE-----
MIIDJjCCAtCgAwIBAgIQVd1yrfv2854JsDjU08ZA+DANBgkqhkiG9w0BAQUFADCB
qTEWMBQGA1UEChMNVmVyaVNpZ24sIEluYzFHMEUGA1UECxM+d3d3LnZlcmlzaWdu
LmNvbS9yZXBvc2l0b3J5L1Rlc3RDUFMgSW5jb3JwLiBCeSBSZWYuIExpYWIuIExU
RC4xRjBEBgNVBAsTPUZvciBWZXJpU2lnbiBhdXRob3JpemVkIHRlc3Rpbmcgb25s
eS4gTm8gYXNzdXJhbmNlcyAoQylwUzE5OTcwHhcNMDUwMzA1MDAwMDAwWhcNMDUw
MzE5MjM1OTU5WjBmMQswCQYDVQQGEwJVUzERMA8GA1UECBMIQ29sb3JhZG8xDzAN
BgNVBAcUBkRlbnZlcjEMMAoGA1UEChQDQkVBMQwwCgYDVQQLFANEUkUxFzAVBgNV
BAMUDmpzdmVkZS5iZWEuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDe
HBPJcbaBYZRMwoxN4iRSBjM8knBuglB0QhxVuQe7MScCfKbP7jTZHaVH+YmNIKjW
b5a/ezkm8Zn13YR8YL1mkMbe1kbGpaiaEA4HDJUGBwKYU6OG/quCtE7Q2s7ko7Ko
lAEAcoeWPsz2FgPo2aXgzTBeqDjf72NHY2dM0t0zOwIDAQABo4HRMIHOMAkGA1Ud
EwQCMAAwCwYDVR0PBAQDAgWgMEIGA1UdHwQ7MDkwN6A1oDOGMWh0dHA6Ly9jcmwu
dmVyaXNpZ24uY29tL1NlY3VyZVNlcnZlclRlc3RpbmdDQS5jcmwwUQYDVR0gBEow
SDBGBgpghkgBhvhFAQcVMDgwNgYIKwYBBQUHAgEWKmh0dHA6Ly93d3cudmVyaXNp
Z24uY29tL3JlcG9zaXRvcnkvVGVzdENQUzAdBgNVHSUEFjAUBggrBgEFBQcDAQYI
KwYBBQUHAwIwDQYJKoZIhvcNAQEFBQADQQBVDkyW9fc1o28p7+lZW7GY/VRL/am5
7dxNQPLXxQjP1gG1wfw2OGh7YAPZCS2QaX/hQnXRWGhtVKBmdB80FDFJ
-----END CERTIFICATE-----
```

Copy this information into a file. I usually name it `servername-signed-cert.pem` to distinguish it from my other certificates.

We can peek at the contents using keytool:

```
D:\ssl-article\examples>keytool -printcert -v -file jsvede-signed-cert.pem
Owner: CN=jsvede.bea.com, OU=DRE, O=BEA, L=Denver, ST=Colorado, C=US
Issuer: OU=For VeriSign authorized testing only. No assurances (C)VS1997, OU=http://www.verisign.com/rep
eriSign, Inc"
Serial number: 55dd72adfbf6f39e09b038d4d3c640f8
Valid from: Fri Mar 04 17:00:00 MST 2005 until: Sat Mar 19 16:59:59 MST 2005
Certificate fingerprints:
     MD5: 29:F6:04:14:90:20:FC:80:FF:BB:A0:EE:A7:4A:81:F7
     SHA1: C3:EC:12:C2:CC:CE:B1:F4:C4:6A:56:80:74:93:EC:A7:48:43:3C:7F
```

This tells us that this certificate is for a server named `jsvede.bea.com` and it was signed by VeriSign but for testing purposes only. Additionally, we see that this certificate is only good for 14 days.

Before we move on, we need one more thing: the root certificate for the VeriSign trial certificate authority. It's available on VeriSign's site, but here are its contents:

```
-----BEGIN CERTIFICATE-----
MIICTTCCAfcCEFKp9CTaZ0ydr09TeFKr724wDQYJKoZIhvcNAQEEBQAwgakxFjAU
BgNVBAoTDVZlcmlTaWduLCBJbmMxRzBFBgNVBAsTPnd3dy52ZXJpc2lnbi5jb20v
cmVwb3NpdG9yeS9UZXN0Q1BTIEluY29ycC4gQnkgUmVmLiBMaWFiLiBMVEQuMUYw
RAYDVQQLEz1Gb3IgVmVyaVNpZ24gYXV0aG9yaXplZCB0ZXN0aW5nIG9ubHkuIE5v
IGFzc3VyYW5jZXMgKEMpVlMxOTk3MB4XDTk4MDYwNzAwMDAwMFoXDTA2MDYwNjIz
NTk1OVowgakxFjAUBgNVBAoTDVZlcmlTaWduLCBJbmMxRzBFBgNVBAsTPnd3dy52
ZXJpc2lnbi5jb20vcmVwb3NpdG9yeS9UZXN0Q1BTIEluY29ycC4gQnkgUmVmLiBM
aWFiLiBMVEQuMUYwRAYDVQQLEz1Gb3IgVmVyaVNpZ24gYXV0aG9yaXplZCB0ZXN0
aW5nIG9ubHkuIE5vIGFzc3VyYW5jZXMgKEMpVlMxOTk3MFwwDQYJKoZIhvcNAQEB
BQADSwAwSAJBAMak6xImJx44jMKcbkACy5/CyMA2fqXK4PlzTtCxRq5tFkDzne7s
cI8oFK/J+gFZNE3bjidDxf07O3JOYG9RGx8CAwEAATANBgkqhkiG9w0BAQQFAANB
AKWnR/KPNxCglpTP5nzbo+QCIkmsCPjTCMnvm7KcwDJguaEwkoi1gBSY9biJp9oK
+cv1Yn3KuVM+YptcWXlfxxI=
-----END CERTIFICATE-----
```

Save this as `verisign-demo-cert-root.pem`.

NOTE

One caveat about Verisign's root certificates: for their production certificates, they usually have a root and an intermediate certificate. If you are using Verisign you should be able to call their customer support organization and ask for the location of both the intermediate and root certificates.

Building the Keystore

Now — finally — I am ready to build the keystore.

The first step is to import the root and where applicable, the intermediate, certificate just mentioned, using the following command:

```
D:\ssl-article\examples>keytool -import -v -noprompt -trustcacerts -alias verisigndemocert -file verisi
store.jks -storepass weblogic1234
Certificate was added to keystore
[Saving server_keystore.jks]
```

The message indicates that the certificate was imported successfully. Remember to give all your certificates a unique alias. I usually use `CA-root-cert` or `CA-int-cert`, where CA is the name of the Certificate Authority.

Two arguments are noteworthy at this point: `-trustcacerts` and `-noprompt`.

The `-trustcacerts` argument tells keytool that you want to import this as a trusted certificate. There are implications of this in other SSL configurations; for now, just be aware that you only want to import certificates as trusted certificates in very specific situations, such as when you are building an identity keystore.

The `-noprompt` argument turns off a prompt that asks you if you are sure that you want to import this certificate as a trusted certificate. Unless you are very comfortable with SSL configuration, I'd recommend not using this option, but you should be aware of it.

Next, verify the contents of the keystore again:

```
D:\ssl-article\examples>keytool -list -v -keystore server_keystore.jks -storepass weblogic1234

Keystore type: jks
Keystore provider: SUN

Your keystore contains 2 entries

Alias name: servercert
Creation date: Mar 4, 2005
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=jsvede.bea.com, OU=DRE, O=BEA, L=Denver, ST=Colorado, C=US
Issuer: CN=jsvede.bea.com, OU=DRE, O=BEA, L=Denver, ST=Colorado, C=US
Serial number: 42291b03
Valid from: Fri Mar 04 19:35:47 MST 2005 until: Thu Jun 02 20:35:47 MDT 2005
Certificate fingerprints:
     MD5: D4:55:EA:25:FF:1A:1C:22:F5:3E:76:53:36:96:CF:93
     SHA1: 82:3F:73:37:A5:B9:A0:24:F4:E4:CA:0F:E8:A9:0B:CB:41:2F:F0:29

Alias name: verisigndemocert
Creation date: Mar 4, 2005
Entry type: trustedCertEntry

Owner: OU=For VeriSign authorized testing only. No assurances (C)VS1997, OU=http://www.verisign.com/repo
riSign, Inc"
Issuer: OU=For VeriSign authorized testing only. No assurances (C)VS1997, OU=http://www.verisign.com/rep
eriSign, Inc"
Serial number: 52a9f424da674c9daf4f537852abef6e
Valid from: Sat Jun 06 18:00:00 MDT 1998 until: Tue Jun 06 17:59:59 MDT 2006
Certificate fingerprints:
     MD5: 40:06:53:11:FD:B3:3E:88:0A:6F:7D:D1:4E:22:91:87
     SHA1: 93:71:C9:EE:57:09:92:5D:0A:8E:FA:02:0B:E2:F5:E6:98:6C:60:DE
```

Now the keystore has the private key and the root certificate. We need to import the server certificate.

To do this, we use the import command again, but we drop the `-trustcacerts` and `-noprompt`, and we specify that the alias is the same as the private key alias. This is a little counter-intuitive, but when we do this, keytool understands that we are trying to establish a relationship between this certificate and the private key.

I run this command only *after* I have imported the root certificate and any intermediate certificates. Having these files available allows the keytool to properly chain your signed certificate to the proper root certificates. If you do not do this in this order, you may have problems when you try to use your SSL port.

This activity of importing the public certificate is often referred to as *creating the chain*. This is accomplished using the following command:

```
D:\ssl-article\examples>keytool -import -v -alias servercert -file jsvede-signed-cert.pem -keystore ser
orepass weblogic1234
Certificate reply was installed in keystore
[Saving server_keystore.jks]
```

Again, keytool provides useful messaging, letting you know that it understood this certificate to be the certificate for key that is identified by the alias servercert.

Finally, if you do another listing of the keystore, it should look like this:

```
D:\ssl-article\examples>keytool -list -v -keystore server_keystore.jks -storepass weblogic1234

Keystore type: jks
Keystore provider: SUN

Your keystore contains 2 entries

Alias name: servercert
Creation date: Mar 4, 2005
Entry type: keyEntry
Certificate chain length: 2
Certificate[1]:
Owner: CN=jsvede.bea.com, OU=DRE, O=BEA, L=Denver, ST=Colorado, C=US
Issuer: OU=For VeriSign authorized testing only. No assurances (C)VS1997, OU=http://www.verisign.com/rep
eriSign, Inc"
Serial number: 55dd72adfbf6f39e09b038d4d3c640f8
Valid from: Fri Mar 04 17:00:00 MST 2005 until: Sat Mar 19 16:59:59 MST 2005
Certificate fingerprints:
     MD5: 29:F6:04:14:90:20:FC:80:FF:BB:A0:EE:A7:4A:81:F7
     SHA1: C3:EC:12:C2:CC:CE:B1:F4:C4:6A:56:80:74:93:EC:A7:48:43:3C:7F
Certificate[2]:
Owner: OU=For VeriSign authorized testing only. No assurances (C)VS1997, OU=http://www.verisign.com/repo
riSign, Inc"
Issuer: OU=For VeriSign authorized testing only. No assurances (C)VS1997, OU=http://www.verisign.com/rep
eriSign, Inc"
Serial number: 52a9f424da674c9daf4f537852abef6e
Valid from: Sat Jun 06 18:00:00 MDT 1998 until: Tue Jun 06 17:59:59 MDT 2006
Certificate fingerprints:
     MD5: 40:06:53:11:FD:B3:3E:88:0A:6F:7D:D1:4E:22:91:87
     SHA1: 93:71:C9:EE:57:09:92:5D:0A:8E:FA:02:0B:E2:F5:E6:98:6C:60:DE
Alias name: verisigndemocert
Creation date: Mar 4, 2005
Entry type: trustedCertEntry

Owner: OU=For VeriSign authorized testing only. No assurances (C)VS1997, OU=http://www.verisign.com/repo
riSign, Inc"
Issuer: OU=For VeriSign authorized testing only. No assurances (C)VS1997, OU=http://www.verisign.com/rep
eriSign, Inc"
Serial number: 52a9f424da674c9daf4f537852abef6e
Valid from: Sat Jun 06 18:00:00 MDT 1998 until: Tue Jun 06 17:59:59 MDT 2006
Certificate fingerprints:
     MD5: 40:06:53:11:FD:B3:3E:88:0A:6F:7D:D1:4E:22:91:87
     SHA1: 93:71:C9:EE:57:09:92:5D:0A:8E:FA:02:0B:E2:F5:E6:98:6C:60:DE
```

That has a lot more information now! The most important thing you want to see is that, under the private key alias, additional information is being displayed. You're looking for this:

```
Certificate chain length: 2
```

This tells you that keystore was successful in establishing the certificate chain, and your keystore is ready for use. (Again, if I were using real production Verisign certificates, this value would be 3 instead of 2.)

SSL is one of the first things you can configure to secure your Web applications. Using Java's keytool, you can quickly build a usable keystore.
