Important info:

Keystore - TestKeyStore.jks

Keystore alias – testks

Certificate – testks.cer

Password for keystore (TestKeyStore.jks) – manage

Alias (testks) Password– manage

csr file for CA verification - testks.csr

CA key-pair - cakey.pem. Top secret file!!

CA CSR - careq.pem

Password for CA key-pair (cakey.pem) – manage

Test CA Root certificate - caroot.cer : This is generated using CA CSR file i.e. careq.pem and signed (self) by CA private key stored in cakey.pem

signedTestCA.cer - Your Own Certificate Signed by CA

**Summary:**

* Generate a Key-Pair (testks, for server) using Java keytool
* Export a certificate (containing the public key) testks.cer from keystore
* Generate a Certificate Signing Request (CSR) testks.csr outof testks
  + This will be used by CA for verification before signing.
* setup a single tier test CA for using Openssl commands
  + For this create the key-pair(cakey.pem ) and create a certificate signing request(careq.pem ) for the root CA's public key
  + Generate a self-signed certificate (caroot.cer) out of Test CA's CSR.
* installed CA root certificate (caroot.cer) in browser’s Trusted Root Certificate Authorities, so that your local browser will trust Test CA's root certificate
* Now generate a certificate (signedTestCA.cer : server certificate signed by CA) signed by Test CA(caroot.cer) on testks.csr using openssl.

Now whoever trust Test CA(for ex our browser), will be able to trust the public key written in signedTestCA.cer and be able to send encrypted message to testks. And testks can decrypt the message using the private key stored in TestKeyStore.jks.

**Create Your Own Certificate and CA:**

**Generate a Key Pair**: Java stores keys and certificate in Key Store. A Key Store is password protected file often with JKS (Java Key Store) extension.

Type the following command to generate a key pair, i.e. a private key and a public key.

C:\TEST\SSLws>keytool -genkey -alias testks -keystore TestKeyStore.jks -keyalg RSA -sigalg SHA1withRSA

Command options:

* genkey: Generate a key pair and add entry to the key store.
* alias: short name of the key pair in the key store

The "Keytool -genkey ..." command also prompts for alias's distinguish name (DN). A DN carries identity information of an entity in ASN.1 format. It consists of

* Certificate owner's common name
* Organization
* Organizational unit
* Locality or city
* State or province
* Country or region

When you run this command you will be asked to enter two password.

* Key Store Password: You can consider that keytool will append this password to the content of the key store and then generate a hash/digest and store it into the key store.
  + If someone modifies the key store without this password, he won't be able to update the digest message.
  + The next time you run keytool on this keystore, it will note the mismatch and warn you not to use this key store anymore.
* Alias Password or Private Key Password: You need to provide an entry password to protect the entry for the alias, here testks. You can consider that keytool will use this password to encrypt testks's private key.

To verify that the key pairs are added to the key store run the command.

C:\TEST\SSLws>keytool -list -v -keystore TestKeyStore.jks

Now the key store TestKeyStore.jks has a private and public key.

**Your Self-Signed Certificate:**

Keystore is a store to keep certificates and private keys. Now we need to export a certificate from keystore containing the public key.

Use the following command to export the public key as a certificate.

**C:\TEST\SSLws**>keytool -export -alias testks -**file testks.cer** -keystore TestKeyStore.jks

Now testks.cer is the certificate containing your public key.

testks.cer is a self signed certificate. Its Owner and Issuer have the same DN. This certificate is signed by the private key of testks.

Open the certificate by double clicking on it. You will notice the red cross sign with the certificate which means the certificate is not trusted. To be trusted you need your certificate to be signed by a well known CA

Note: You cannot export the private key using keytool. To extract the private key you need to use Java Cryptography API.

**Generate a Certificate Signing Request (CSR):**

To be trusted you need your certificate to be signed by a well known CA. To do that first you need to generate a Certificate Signing Request (CSR) and send it to CA. Keytool uses following command to generate a CSR:

**C:\TEST\SSLws**>keytool -certreq -alias testks -keystore TestKeyStore.jks -file **testks.csr**

The above command extracts required information such as public key, DN and put it in a standard CSR format in file testks.csr. A commercial CA should verify all information before they can issue a certificate with their signature.

In the next section we are going to create our own test CA and register it as trusted to the browser and use it to sign our public key.

**Set Up a Certificate Authority:**

Here we are going to create a single tier CA, where root CA and issuing CA are the same.

setup a Test CA folder.

c:\>cd CA

c:\CA>set RANDFILE=rand

Openssl commands need to save a random seed information to a file ("random file"). You need to tell it the path to that file. Here, just tell it to use a file named "rand" in the current folder.

To start Test CA, we need a private key. This is the top secret of the CA. If this is compromised then the CA is doomed!!! All certificates issued by this CA will be revoked. This is why the root private key is so important and often kept off-line necessitating a multi-tier hierarchy.

For our test CA we need to create the **key-pair** and create a **certificate signing request** for the root CA's public key.

Please note this CSR is for the CA itself. These two steps can be done in a single command using SSL as follows:

C:\TEST\SSLws\CA> openssl req -new -keyout **cakey.pem** -out **careq.pem** -config "C:\Program Files\OpenSSL-Win64\bin\openssl.cfg"

cakey.pem - CA key-pair. Top secret file!!

careq.pem - CA CSR

When you run the above command it will ask our Test CA's DN and a password to encrypt the private key while writing in cakey.pem file.

Now we need to generate a certificate out of Test CA's CSR. Obviously this would be self signed. Command:

C:\TEST\SSLws\CA>openssl x509 -signkey cakey.pem -req -days 3650 -in careq.pem -**out caroot.cer** -extensions v3\_ca

Command Explanation:

* x509: Work on an X.509 certificate
* signkey: self sign the certificate using private in as stored in file cakey.pem
* req: tells that input is a CSR
* days: specify days of validity of the generated certificate
* in: the input, i.e. CSR careq.pem
* out: write the output certificate on caroot.cer
* extensions: apply x.509 v3 extensions

Now you have self signed root certificate of our Test CA. This certificate along with the private key will be used to sign others certificates.

This root public certificate should be publicly available and must be trusted by browsers and programs.

**Trusting CA's Root Certificate:**

How browser will trust Test CA's root certificate?

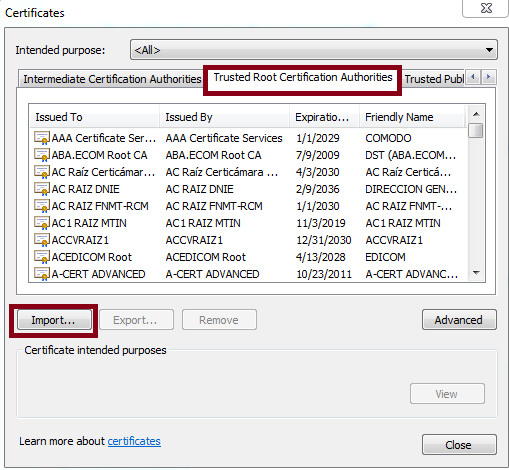
Every browser keeps well known CA's root certificates in their trust store. Browsers manage (add/delete) these certificates during security patches time to time.

If you put/installed CA root certificate (caroot.cer) in browser’s Trusted Root Certificate Authorities, browser will start trusting CA root certificate.

Open IE and click on Internet Options->Content->Certificates->Trusted Root Certification Authorities

Chrome : Settings->Advanced->Privacy and Security-> Manage certificates-> Trusted Root Certification Authorities

Add our Test CA's root certificate here. Click on import and follow instructions and select caroot.cer file from C:\TEST\SSLws\CA folder.



Java Environment also keeps root CA certificates in C:\Program Files (x86)\Java\jre1.6.0\_22\lib\security\cacerts file. It is a keystore and you can open it using the following command.

C:\Program Files (x86)\Java\jre1.6.0\_22\lib\security>keytool -list -protected -keystore cacerts

You can add Test CA's root certificate to your JRE using keytool command. The initial password of the "cacerts" keystore file is "changeit".

You can also add the Test CA's root certificate in TestKeyStore.jks or your own key store and point to that key store at runtime when you need to use any certificate signed by this Test CA.

If you cannot convince JRE that Test CA's root is trusted all certificates signed by Test CA will not work at runtime!

**Get Your Own Certificate Signed by CA:**

Test CA is ready, we have added it to the browser's truststore.

Now it the time to get your own certificate signed by the Test CA.

Note: when a CA issues a new certificate, it will put a unique serial number into that certificate.

So you need to tell OpenSSL what is the next serial number to use.

To do that drop a serial.txt file containing a serial number in the CA directory.

C:\TEST\SSLws\CA>echo 1234>serial.txt

This way OpenSSL will use 1234 as the next serial number. Then it will set it to 1235 automatically.

To sign testks's public key and DN we now need the CSR which is testks.csr and use the following OpenSSL command:

C:\CA>openssl x509 -CA caroot.cer -CAkey cakey.pem -CAserial serial.txt -req -in ../keys/testks.csr -out ../keys/signedTestCA.cer -days 700

Command Explanation:

* x509: again working with X.509 certificates.
* CA: sign the certificate using caroot.cer.
* CAkey: take the private key from cakey.pem file
* CAserial: point to serial number file
* req: says that input is a CSR and not a certificate itself
* in: provides the input CSR as ../Keys/testks.csr
* out: write the output certificate in ../Keys/signedTestCA.cer file
* days: validity of the certificate

Now you have a certificate signed by Test CA, i.e. signedTestCA.cer.

The signedTestCA is trusted as it is signed by Test CA which is trusted to the Windows/IE.

Now whoever trust Test CA, will be able to trust the public key writtin in signedTestCA.cer and be able to send encrypted message to testks. And testks can decrypt the message using the private key stored in TestKeyStore.jks.

**Keep Your Certificates in Key Store:**

You can separately keep the signedTestCA.cer and send it to clients or

import signedTestCA.cer to the key store.

But before that we need to import the signer certificate, i.e. root certificate of Test CA otherwise keytool will not import signedTestCA.cer. Because it would not be able to trust the signer.

While importing Test CA's root certificate keytool will ask for a confirmation that we really trust Test CA. If you say YES keytool will take all certificates signed by the root certificate.

C:\Keys>keytool -import -alias TestCA -file ../CA/caroot.cer -keystore TestKeyStore.jks

Reference:

<https://sites.google.com/site/ddmwsst/create-your-own-certificate-and-ca>