CA (Certificate Authority).

I used openssl to create the CA and do the signing.

This is the command I used to create the CA (self signed) certificate and its key.

**openssl req -newkey rsa:2048 -nodes -keyout keyCA.em -x509 -days 365 -out certificateCA.pem**

This are the commands I used to generate the key and the request for my CSR.

**openssl genrsa -out x509.key 2048**

**openssl req -new -key x509.key -out x509.csr**

This is how I signed the CSRs.

**openssl x509 -req -in x509.csr -CA certificateCA.pem -CAkey keyCA.pem -CAcreateserial -out newx509.crt -days 1825 -sha256**

This is one of the commands I used to the newly created certificates.

**openssl verify -verbose -CAfile certificateCA.pem newx509.crt**

I also used would try to visually verify the contents of the CSRs and CRTs I made. At first I would just use cat, but after a bit of research, I found out about:

**openssl req -text -noout -verify -in x509.csr**

**openssl x509 -in x509.crt -text -noout**

Even keys have their own command, though it is not much different from cat.

**openssl rsa -in x509.key -check**

I also tested my certificate by signing a file (via openssl). I could not do it for other certificates I made, because I did not have their keys.

**openssl dgst -sha256 -sign x509.key -out sign.txt.sha256 sign.txt**

**openssl dgst -sha256 -verify <(openssl x509 -in x509.crt -pubkey -noout) -signature sign.txt.sha256 sign.txt**

As you see, the signature is done with the private key (x509.key), and then I assume the digital signature of the file and the public key from the certificate are compared.

I wrote a small bash script to automate my CA.

#!/usr/bin/env bash

list=$(ls \*.csr)

array=( $list );

size=${#array[@]};

x=0;

name="";

temp="";

suffix=".csr";

while [ $x -lt $size ]

do

name=$(echo ${array[$x]} | sed -e s/$suffix$//)

echo $name

openssl x509 -req -in "$name".csr -CA certificateCA.pem -CAkey keyCA.pem -CAcreateserial -out "$name".crt -days 1825 -sha256

mv "$name".crt output

mv "$name".csr ./done

let "x=x+1";

done

Most of the code is straight forward, except for the sed part, which I discovered quite recently. I tried looking for some reg exp solution (something like grep) but did not find any, and while implementing the solution in code would not be that hard, I got too used to high end languages, and basic default features like split string by a delimiter.

Anyway, let me explain the syntax – s/string1/string2 means that every occurrence of string 1 is replaced by string 2. In my case (s/$suffix$//), I replace the suffix (which is “.csr”) with an empty string.

The -e flag is used when you point to an expression, but there are other flags, like -f that point to a script file. Honestly, I do not quite get it, but the documentation on sed was quite big, and I feel that this command is just the tip of the iceberg.

Importing into keystore.

I had a lot of problems importing the keys. The main issue is that I did not use the keytool to create my CSR, which created some issues.

I could import my CA and X509 certificates, but I could not import the private key of my X509. From my previous experiment I did in openssl (to sign a file), I knew that we needed a private key, and I was right, because the jarsigner would give errors when I would try using it. Unfortunately, it is impossible to just import a key with keytool.

The only solution is to combine the X509 certificate and the private key together into the PKCS12 format. Note that the alias that will appear in the keytool is the name argument that I set here.

**openssl pkcs12 -export -in x509.crt -inkey x509.key -name x509cert -out x509.p12**

Then I can import it into the keystore with this keytool command

**keytool -importkeystore -deststorepass [password] -destkeystore keystore.dir -srckeystore x509.p12 -srcstorepass [password] -srcstoretype PKCS12**

I imported the CA with this command

**keytool -import -trustcacerts -alias ca\_root -file certificateCA.pem -keystore keystore.dir**

This is the command I used to check my keystore

**keytool -list -v -keystore keystore.dir**

* *it can also be run without -v flag, and with an -alias if you are looking for a specific certificate.*

That almost made my x509 certificate work. I created a dummy jar file and tried signing it with the jarsigner. The commands are as follows

**jarsigner -signedjar signed.jar -keystore keystore.dir -storepass [password] file.jar x509cert**

**jarsigner -verify signed.jar**

* *verify can also be run with -verbose and -certs.*
* *signer can also be run with -verbose.*

It gave proper responses this time, like “jar file signed” and “jar file verified”, but it would also show errors. One of the them a tsa timestamp warning, and the other one was this, which I felt was more important.

*sun.security.validator.ValidatorException: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target*

According to the sources on internet it is because my certificate was signed by local CA, and it is not in the cacerts, a personal java keystore that has all the certificates it trusts.

At first I used this command to find the cacerts, as well as find out what version of java I am using. I also created a JAVA\_HOME, thinking that maybe the machine cannot find the cacert file without that variable (I had the java bin path pasted directly into path).

**where java**

**dir /s /b “cacerts”** (I run that from the very root directory – C:\)

Then I would import my CA into that keystore, with this command

**keytool -import -trustcacerts -alias ca\_root -file certificateCA.pem -keystore cacerts.dir**

At first it would not run, saying that I did not have permission, until I realized that I have to run my cmd in the administrator mode.

After that it run without any issues.

I also tried the keytool commands that were in the lecture (the signing I still did in openssl)

**keytool -genkeypair -alias x509cert -validity 365 -keystore keystore.dir -storepass [password]**

**keytool -certreq -alias x509cert -file x509.csr -keystore keystore.dir -storepass [password]**

**keytool -import -alias x509cert -file x509.crt -keystore keystore.dir -storepass [password]**

There are also openssl equivalent commands for reading certificates and certificate requests.

**keytool -printcertreq -file x509.csr**

**keytool -printcert -file x509.crt**

Anyway, I tested them with a jarsigner, and they worked too. I even tried using the default keystore to see if that makes a difference, and tried changing settings in java security, but I guess the only problem was the cacert file.