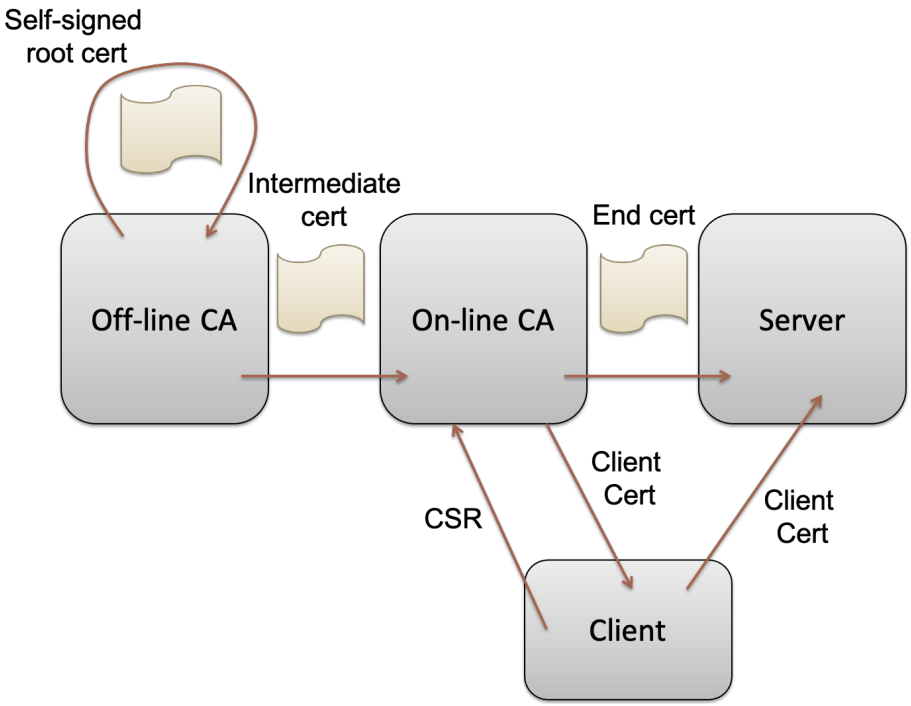
CS 594—Spring 2021
Enterprise and Cloud Security
Assignment Three—Certificate Manager

In this assignment, you will complete a simple certificate manager that you will use in future assignments to develop a public key infrastructure. The certificate manager is a desktop application that only accesses the local file system. Although you could use `keytool` for the same purpose, managing keys and certificates, eventually some of the functionality of the certificate manager will be moved on-line, to a REST service and interface. The purpose of this assignment is partly to give you some experience with developing an enterprise PKI using Java APIs, and partly to develop some of the infrastructure for an online PKI in the next assignment.

You are provided with a desktop application, as a Maven project `CertManager`. When you perform a "`Run As ... | Maven install`", Maven installs a jar file `certmanager.jar` in a subfolder `cs594` of your home directory. You should complete the parts of the assignment that have been deliberately left incomplete (marked "TODO" in the code). Once completed, you should use the certificate manager to generate the certificates for a simple Web service application. In the next assignment, you will deploy a Web service and a desktop Web client, that take over some of the functionality of the certificate manager.

This is the scenario for the application that the certificate manager is intended to support:



There are three keystores associated with this application scenario, that for the offline root CA, that for the online CA, and that for the server. The idea is that the offline CA is never online, because of the danger of a network breach. Instead, it is used to sign the credential for the online CA, that issues client certificates. Its certificate is added as a trust anchor by the server and the clients, so they accept each others' credentials when they communicate.

For this assignment, we will use the certificate manager to generate all certificates (and store them in the appropriate keystores and truststores). These keystores and truststores can then be distributed to the file systems of the applications, so they can recover their credentials to establish their trustworthiness. In a production deployment, a client application would generate its own self-signed certificate, use this to generate a certificate signing request (CSR) for the online CA, the online CA would generate a client certificate from the CSR, and the client would then import that client certificate into its own keystore, so it can authenticate to the server. For this assignment, these four steps are implemented in the certificate manager, since we are not yet deploying an application. If we were deploying an application infrastructure including an online CA, these four parts of the certificate manager would move to the client application and the online CA.

Certificate Manager

The certificate manager manages keystores and truststores in several directories:

1. `certs-offline` stores the offline keystore, `caroot.p12`, that contains the root CA signing key.
2. `certs-online` stores the online keystores `keystore.p12`, that contains the signing key for the online CA, `keystore.jks`, that contains the SSL signing key for the server, and `cacerts.jks`, that contains the certificates that the server uses to authenticate clients.
3. `certs-backup` stores backups of the keystores and truststores when any changes are made, so that no private keys are lost.

Note that we will be using Bouncycastle exclusively as the JCE provider, so although we use PKCS12 format for keystores, this is different from the PKCS12 provided by the JDK provider and by `keytool`.

The certificate manager provides a command line interface (CLI), with several optional arguments at the shell command line:

1. `--basedir`: Specifies the base directory where keystores and truststores are stored. This defaults to the current directory.

2. `--passwordfile`: Specifies a properties file where keystore and truststore passwords are stored[1]. Defaults to "`passwords.properties`" in the base directory.
3. `--namesfile`: Specifies a properties file that provides distinguished names for the managed certificates. Defaults to "`names.properties`" in the base directory.
4. `--scriptfile`: Specifies the name of a script file of commands. If this is not provided, then the certificate manager launches its own shell to accept commands interactively.

The manager understands the following commands. For the off-line root certificate authority, it provides these commands:

1. `gencaroot`: Initialize the off-line keystore with the self-signed certificate for the root CA.
2. `exportcaroot --cert cert-file`: Export an X509 certificate for the root CA, as a PEM file.

For managing online keystores, the manager provides these commands:

1. `genservercert --dns server-domain-name`: Generate an SSL certificate for the server, signed by the root CA, that clients can use to authenticate the server, and stored in the application server keystore.
2. `genonlinecacert`: Generate a certificate for the online CA, signed by the offline CA, and stored in the online keystore.
3. `exportonlinecacert --cert cert-file`: Export an X509 certificate for the online CA, as a PEM file.

As explained above, for now we manage client certificates using the certificate manager, with these commands:

1. `genclientroot --clientdn client-dist-name --duration cert-duration --keystore client-keystore --storepass keystore-password --keypass key-password`: Initialize the client keystore with a self-signed v1 certificate.
2. `genclientcsr --keystore client-keystore --storepass keystore-password --keypass key-password --csr csr-file [--dns client-domain-name]`: Generate a certificate signing request with the client keystore, outputting the result as a PEM file. The CSR is signed by the self-signed root certificate for the client, stored in the specified client keystore. Optionally the CSR may include a DNS domain name for the client.
3. `genclientcert --csr csr-file --cert cert-file`: Generate a client certificate from the certificate signing request (CSR). The client certificate is signed by the online CA. Both the input CSR and the output certificate are PEM files.
4. `importclientcert --keystore client-keystore --storepass keystore-password --keypass key-password --cert cert-file`: Import the client

[1] You will need to make sure that the passwords for the application server keystore and truststore match the master password that you specified when you set up your domain in the application server (The default master password for Payara is "changeit").

certificate into the specified client keystore. The certificate is provided as a PEM file.

Finally, the certificate manager has a `showcerts` command, that shows the certificates in the online and offline keystores. If you provide the appropriate arguments, it can also show you the contents of a client keystore:

1. `showcerts [--keystore client-keystore --storepass keystore-password --keypass key-password]`: Show the certificates, including the client certificate if specified.

The `res/main/resources` folder of the Maven project for the certificate manager specifies several properties files:

- `names.properties` specifies the distinguished names that are used in the certificates: the name for the root CA, the name for the online CA, and the name for the server itself. The latter name should include a DNS name, you can use something like `www.example.org`.
- `names.passwords` specifies the passwords that are used to protect the truststores and keystores. Those for the application server should be consistent with the master password for the application server.

These are defaults and can be overridden with other properties files on the command line of the certificate manager.

Deployment [Not necessary until Assignment 4]

In order to deploy this app, you will need to have administrative access to any machines on which you wish to deploy, because of the use of Bouncycastle, and because you should be using reasonably sized cryptographic keys~~. Even though the Maven POM file for the certificate manager has the proper Maven dependencies for the Bouncycastle libraries, you will need to manually install the jar files on any machine on which this app runs. First, download the following jar files from the Bouncycastle Web site (https://www.bouncycastle.org/latest_releases.html):~~

- ~~bcprov-jdk15on-168.jar~~
- ~~bcpkix-jdk15on-168.jar~~

~~Copy these jar files to $JAVA_HOME/jre/lib/ext, the folder in the JDK installation directory for jar files that extend the standard JDK library. On Unix, type:~~

```
$ sudo cp bcpkix-jdk15on-168.jar bcprov-jdk15on-168.jar \
          $JAVA_HOME/jre/lib/ext
$ cd $JAVA_HOME/jre/lib/ext
$ sudo chmod go+r *.jar
```

Edit the file ~~`$JAVA_HOME/jre/lib/security/java.security`~~ `$JAVA_HOME/conf/security/java.security` file, adding Bouncycastle as the second provider in order of priority (after the default Sun provider), and renumbering all providers below it. See the lecture slides for an example of how to do this.

Assuming you are using JDK 11 or later, you do not need to install jurisdiction policy files for unlimited strength encryption (This was necessary through JDK 8). Make sure that your project is not exported outside the U.S., or you may be in violation of US export laws. If deploying to the cloud, (not required for this assignment) make sure that you deploy in a US data center.

To manage your certificates, you should first copy the application server keystore and truststore, `keystore.jks` and `cacerts.jks` respectively, from the `domains/domain1/config` subdirectory of the Payara installation directory, to the `certs-online` subfolder of your base directory:

```
docker cp \
   cid:/opt/payara/appserver/glassfish/domains/domain1/config/keystore.jks \
   certs-online
docker cp \
   cid:/opt/payara/appserver/glassfish/domains/domain1/config/cacerts.jks \
   certs-online
```

Do this before you create the server certificate or the online CA certificate. Then create the root CA certificate, the server SSL certificate and the online CA certificate.

Creating the root CA certificate will save the credential in `caroot.p12`, a PKCS12 keystore in the `certs-offline` subdirectory.

Creating the server SSL certificate will save the credential in `keystore.jks` (in the certs-online subdirectory) and will save the root CA certificate in `cacerts.jks`. Clients would import the root CA certificate into their truststore, so that they will trust SSL certificates provided by the server and issued by the root CA. Inserting the root CA certificate in `cacerts.jks` ensures that the administration console for Payara trusts the server it is administering. Any clients wishing to access the server safely would include the root CA certificate in their truststore, so that they can authenticate the server.

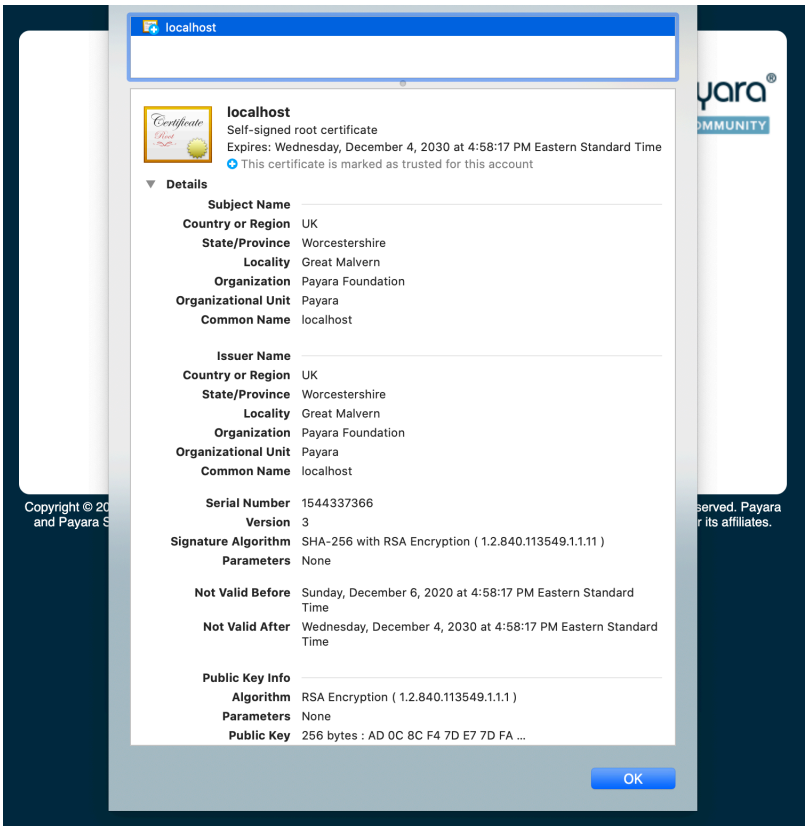
Creating the online CA certificate will save the credential in the online `keystore.p12`, that will be used by the application rather than the application server. The certificate will be saved in `cacerts.jks`, so that the application server will accept client certificates issued by the online CA. We may use this in the next assignment, when we deploy a RESTful application using client certificates for authentication.

To deploy the certificates to your service, you should copy the app keystore, app server keystore and app server truststore (`keystore.p12`, `keystore.jks` and `cacerts.jks`) to the `domains/domain1/config` subfolder of your Payara installation:[2]

[2] Make sure to save a copy of the app server keystore and truststore before you do this.

```
docker cp certs-online/* \
   cid:/opt/payara/appserver/glassfish/domains/domain1/config
```

If you install the root CA certificate as a trust anchor in your browser, then you should be able to use the Web application from the previous assignment without being warned about an untrusted host. Show accessing the application server with a Web browser without getting a warning, and show the certificate that the Web server provides:



Submission

You should submit your completed version of the `CertManager` class, as well as certificates (keystores, truststores and PEM files) generated by your completed tool. ~~Include in these a client truststore and keystore that incorporate client certificates issued by your application. You can use the `keytool` command to import the root CA certificate into the client truststore, which you can make a JKS truststore (`truststore.jks`) for now.~~

Your solution should be developed for Java 11 or later, and Bouncycastle 1.68. In addition, record short mpeg videos of a demonstration of your assignment working. In these videos, you should run the commands, and at each step use the `showcerts` command to show your certificates. Make sure that your name appears at the beginning of the video. *Do not provide private information such as your email or cwid in the video*.

Your solution should be uploaded via the Canvas classroom. Your solution should consist of a zip archive with one folder, identified by your name. Within that folder, you should have the Eclipse projects: `CertManager`. You should also have a folder called test with the results of your testing. This should include the `certs-offline` and `certs-online` folders, as well as a client truststore and keystore, and any properties files required for your application (`passwords.properties` and `names.properties`). You should also provide videos demonstrating the working of your assignment. You should also provide a completed rubric for the assignment. Finally the root folder should contain the jar file (`CertManager.jar`) that you used to test your application.