


Task 4: Handshake Certificates

Due 8 Dec 2022 by 17:00 **Points** 1 **Submitting** a website url
Available 23 Nov 2022 at 10:00 - 8 Dec 2022 at 17:00



This assignment was locked 8 Dec 2022 at 17:00.

Certificates are important in secure communication protocols. They can be used to distribute public keys, and they can be used to authenticate the communicating parties to each other.

In this task, you create, sign and verify certificates. The format of certificates is defined by the ITU-T X.509 standard, and its usage on the Internet is standardised by the PKIX working group in [RFC 5280](https://datatracker.ietf.org/doc/html/rfc5280). 
(<https://datatracker.ietf.org/doc/html/rfc5280>)

You will use certificates in the final project, for the handshake phase of the protocol. During the handshake phase, the client and server authenticate each other, and exchange public keys, through the use of certificates.

OpenSSL

 (<https://www.openssl.org>) **OpenSSL**  (<https://www.openssl.org>) is an open source implementation of the SSL/TLS protocol. The OpenSSL software also contains application programs for creating and managing certificates, and much more. OpenSSL is available on Linux, Mac and Windows. It normally comes pre-installed in Linux and Mac OS.

These instructions are for the "openssl" command line tool. There may be other tools for SSL/TLS around as well. However, if you want to use some other tool, you are on your own.

OpenSSL in Windows

For Windows, you may need to download and install OpenSSL yourself. There are several pre-compiled binary versions for Windows. We don't recommended a

particular version. Instead, look into the alternatives and pick one! Please feel free to share your experiences on the project discussion forum.

Creating certificates

All certificate operations are performed using "openssl". You can find much information on the web about how to use openssl to create certificates. These instructions give a brief introduction; for more information, search the web!

CA Certificate

In this assignment, you act both as a CA (Certification Authority), and as a user. Start with creating a certificate for your CA. A CA certificate is *self-signed*, while a certificate for a regular user is signed by a CA.

To create a certificate, give the "req" command to openssl. If you combine it with the "-new" option, openssl will prompt you for the information to include in the certificate. Self-signed certificates are created by giving the "-x509" option for the "req" command. So, a command to create a CA certificate interactively could look something like this:

```
openssl req -new -x509 ...
```

As you know, a certificate contains a public key. A self-signed certificate is signed with the corresponding private key. Create a key pair using the "genrsa" command to openssl (not shown here, look it up!). Store the key pair in a file, and tell openssl to read the keys from the file:

```
openssl req -new -x509 -key <filename> ...
```

If you instead specify the "-newkey" option together with "-x509", openssl will first create a new key pair for you and then create the certificate:

```
openssl req -new -x509 -newkey rsa:2048 -keyout privatekey.pem ...
```

This command creates a self-signed certificate with a 2048-bit RSA key. The key pair is stored in the file "privatekey.pem".

As a CA, you should protect your private key very carefully. So you must not distribute the file with the key pair – keep it to yourself! At the same time, you want to publish your public key so that others can use it to verify certificates you have signed. To publish your public key, you must first extract it from the file with the key pair. Use the "RSA" command for this.

```
openssl rsa -in privatekey.pem -pubout -out publickey.pem
```

Regular Certificate

A certificate for a regular user, which should be signed by a CA, is created in two steps. The first step is to create a *certificate signing request* (CSR). A CSR contains the key, the information about the subject that the user would like to have in the certificate, and directives for the CA. A CSR is created using the "req" command to "openssl", in very much the same way as for self-signed certificates. The difference is that you *should not* use the "-x509" option if you want to create a CSR.

The second step is to ask the CA (which is you...) to sign the certificate. To sign a certificate, use the "x509" command to "openssl" and specify the files with the CSR and the CA certificate as parameters. You may need more parameters as well; you should find this out yourself.

Rules and Guidelines for Creating Certificates


- OpenSSL can use different file formats. You should use PEM (Privacy-Enhanced Mail).
- Use proper validity dates for the certificates, both for the CA and the user certificates. The certificates should be valid at least for the duration of this course.
- Make sure to fill in reasonable values for the subject *Distinguished Name* (DN) in the certificate. In particular, it is mandatory that each certificate has a *Common Name* (CN) in its subject DN. Your name and email address must be included in the CN. Make the CN for the CA and the CN for the user different, so that you can distinguish them.
 - In the grading, the CN field will be checked against your KTH email address and your full name as it appears in Canvas. So do not use a

private email address, or any alternative way of writing your full name.

Storing Certificates in Files

There are two formats for storing certificates in files, DER and PEM. They are closely related. DER (Distinguished Encoding Rules) is a binary representation of encryption keys and certificates. PEM (which originally stands for Privacy Enhanced Mail) is a textual representation of the same information. An example of a PEM file with a certificate is shown below.

```
-----BEGIN CERTIFICATE-----
MIID1jsCAr42CQDT/2gogQ2LSjANBgkqhkiG9w04AQsFd3CBqjELMAkGA1UEBhMC
U0UxDjAMBgNVBAgMBUtpc3RhMR1wEAYDVQQHDAlTdG9ja2hvbG0xDDAKBgNVBAoM
...
e6VINwWxq7FE/z03ZmKnG83xCxY45drqjB1GjRZezoEPSU+1aR5TKcgYN/7Dg5cn
xcS3N2kVgQpq475LZFhKLuhITnkwhD2v9uY=
-----END CERTIFICATE-----
```

The first line is a header line that tells that the content is a certificate, and the last line is a footer that marks the end of the certificate. The text between the header and the footer is the DER representation of the certificate, encoded as text using [Base64 encoding](https://en.wikipedia.org/wiki/Base64)  [. \(https://en.wikipedia.org/wiki/Base64\)](https://en.wikipedia.org/wiki/Base64).

OpenSSL by default stores X509 certificates in PEM format.

Certificate Management in Java

To deal with certificates during the handshake, we define a class called `HandshakeCertificate`.

It has two constructors. The first creates a `HandshakeCertificate` from an X509 certificate, which is read from an input stream. Hint: the `X509Certificate` class in Java has a constructor that creates a certificate from a byte stream in PEM or DER format.

```
HandshakeCertificate(InputStream instream)
```

The second creates a `HandshakeCertificate` from an X509 certificate in its encoded form (DER), given as a byte array:

```
HandshakeCertificate(byte[] certBytes)
```

Furthermore, `HandshakeCertificate` has two "getters" to get the X509 certificate in different formats. The first returns the X509 certificate in its encoded form as a byte array:

```
public byte[] getBytes()
```

The second returns the X509 certificate as an `X509Certificate` instance:

```
public X509Certificate getCertificate()
```

Certificate Verification

The `verify` method cryptographically validates a certificate. It takes another `HandshakeCertificate` as parameter, namely the certificate for the CA that signed the certificate. The `verify` method checks the signature, but it does not return anything. Instead, it throws an exception if the verification fails.

```
public void verify(HandshakeCertificate cacert)
```

It is mandatory that in your declaration of the `verify` method, you specify exactly the exceptions that it throws. Using an unspecified exception (such as "`throws Exception`") is in general considered bad programming. In this case it also hides important information about what the possible results could be of calling this method.

You will find that the `X509Certificate` class has a corresponding method that you can use directly, so this method can be done easily as a "wrapper" around the `X509Certificate` method.

Retrieving Information from a Certificate

In addition to the cryptographic validation, certificate verification involves checking the content of the certificate: validity dates, information about the user ("Subject" in X509 terminology), and more. You do not need to do that here, but you will do it in the final assignment. Therefore, to provide support for such operations, implement a few methods to retrieve various pieces of certificate information:

```
public String getCN()
```

```
public String getEmail()
```

The `getCN` method retrieves the subject's Common Name (CN) from the certificate, while `getEmail` retrieves the subject's email address. The `X509Certificate` class has no directly corresponding methods. Instead, you need to do some string processing to get this information from the methods that are available in the `X509Certificate` class.

Assignment

- Create a CA certificate
- Create a user certificate
- Implement the `HandshakeCertificate` class

To help you get started, there are skeleton files with all the required declarations in the Project resources repository on KTH GitHub.

Tip: study JCA carefully, read online tutorials and look at examples. If you do this right, each class will only be a few lines of codes.

Testing

Create certificates with `openssl` and use your implementation of the `HandshakeCertificate` class to check the certificate. You can also ask a friend to check your certificates.

To help you with the testing, the project resources contains a Java file with a few JUnit 5 unit tests.

Submission

Your submission should contain the following:

1. The CA certificate, in a file called "CA.pem"
2. The user certificate, in a file called "user.pem"
3. Your java program in a file named "HandshakeCertificate.java". The file should not contain any package declarations. If your IDE inserts package declarations for you, you need to remove them.

In order to complete your submission, you need to do two things:

- Commit your changes in git and push your repository to your personal repository on KTH GitHub.
- Submit the URL of your personal KTH GitHub repository here, in this assignment, to notify us about the submission.
 - The URL looks something like this: "https://gits-15.sys.kth.se/IK2206HT22/<user>-HandshakeCertificate.git"

Note: We will not automatically check your personal KTH GitHub repository for updates. Whenever you make an update that you want us know about, you have to make a submission here. Submit the URL of your personal KTH GitHub repository, as described above.