

Task 5: Handshake Encryption

Due 14 Dec 2022 by 17:00 **Points** 1 **Submitting** a website url
Available 23 Nov 2022 at 11:00 - 14 Dec 2022 at 17:00

This assignment was locked 14 Dec 2022 at 17:00.

In this task, you use the key pair from the previous step to encrypt and decrypt information with RSA. As you can tell by the name, you will use this during the handshake phase in the VPN project.

The HandShakeCrypto class

We define a class called HandshakeCrypto for the cryptography operations in the handshake. It can be used for encryption and decryption, with either a public key or a private key. The class has two constructors. The first takes a HandshakeCertificate as argument:

```
public HandshakeCrypto(HandshakeCertificate handshakeCertificate)
```

When a HandshakeCrypto instance is created in this way, it means that any encryption/decryption operations performed with the instance should be done using the public key from the certificate as key.

The second constructor takes a byte array as argument:

```
public HandshakeCrypto(byte[] keybytes)
```

The byte array contains a private key in PKCS#8/DER format (see below). When a HandshakeCrypto instance is created in this way, it means that any encryption/decryption operations performed with the instance should be done using the private key as key.

The class has two methods:

```
public byte[] encrypt(byte[] plaintext)
```

The encrypt method takes a plaintext as a byte array, and returns the corresponding ciphertext as a byte array.

```
public byte[] decrypt(byte[] ciphertext)
```

The decrypt method takes a ciphertext as a byte array, and returns the corresponding plaintext as a byte array.


Your HandShakeCrypto class should be able to handle certificates and key files with RSA keys of any possible length, that is, all sizes that are defined for RSA.

A note on reading private keys from files

While reading a public key from a certificate file is straight-forward in JCA/JCE, getting the private key is somewhat more involved.

Some background first: there are different ways of representing private keys. OpenSSL supports two common representations, as defined by the standards [PKCS#1](https://en.wikipedia.org/wiki/PKCS_1) and [PKCS#8](https://en.wikipedia.org/wiki/PKCS_8). Moreover, a file that stores a key can also have different formats. The file can contain the key in binary format (in which case the file format is DER, Distinguished Encoding Rules). It can also be text-encoded, in which case the format is PEM (Privacy-Enhanced Mail). So there are four combinations in total. OpenSSL, as described in these instructions, uses the PKCS#1 standard and PEM file format. Java JCA/JCE, on the other hand, uses PKCS#8 internally and therefore cannot directly read keys in the format generated by OpenSSL. There are basically two ways you can deal with this. (If you come up with a better way, please let us know!)

1. Write the Java code yourself to read key data in PKCS#1 format from a PEM file. It is not very complicated. See for example <https://github.com/mendix/SSLTools/blob/master/src/main/java/com/mendix/ssltools/KeyReader.java> and <https://stackoverflow.com/questions/11787571/how-to-read-pem-file-to-get-private-and-public-key>.

2. This is probably easier, so this is what we recommend: Convert the OpenSSL key file from PKCS#1/PEM format to PKCS#8/DER, which has better support in JCA/JCE. You still need to process it in Java, but that can be done in one line of code. See for instance <https://stackoverflow.com/questions/20119874/how-to-load-the-private-key-from-a-der-file-into-java-private-key-object>  (<https://stackoverflow.com/questions/20119874/how-to-load-the-private-key-from-a-der-file-into-java-private-key-object>). To convert a PKCS#1/PEM file to a PKCS#8/DER file, use the "pkcs8" command for openssl:

```
openssl pkcs8 -nocrypt -topk8 -inform PEM -in private-pkcs1.pem -outform DER -out private-pkcs8.der
```

Assignment

Implement the HandshakeCrypto class with the four methods above.

Testing

The primary test is straightforward: take a piece of text and encrypt it into a ciphertext. Decrypt the cipher text, and check that the result matches. In your personal repository, you can find skeleton code and Junit 5 unit tests.

Submission

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>-HandshakeCrypto.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.

