1. Certificate in Java

Objective:

The goal of this exercise is to manipulate certificate in Java. The management of certificate can be done via `java.security.keyStore.`

Exercises:

1.1. Generation of certificate.

For the generation of the certificate, you have to use `keytool`. This tool can be found under `<JavaHome>/bin`. Additional documentation can be found here: http://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html.

With `keytool`, you have to generate a keystore containing a self-signed certificate with the following properties:

- Keystore name : `myKeyStore`
- Keystore password: `security`
- Used Algorithm: `DSA with MD5`
- Alias name: `sessionlab2`
- Alias password: `sessionlab2`

Check if the certificate is in the keystore.

In order to validate this step, you have to make a screenshot of the keystore and certificate generation of the keystore. In addition, you have to make a screenshot of certificate display.

1.2 Keystore management in Java

In this step, you will use the certificate generated with `keytool`. You will have to use `java.security.keyStore`. API can be found here: http://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html

You have to:

- Create an instance of a your keystore.
- Get the certificate associed to your alias
- Get and display the public key
- Get and display the private key

1.3 Sign an object

In this step, you will sign the class `test` with the generated certificate. You will have to use `java.security.SignedObject`. API can be found here: http://docs.oracle.com/javase/7/docs/api/java/security/SignedObject.html

You have to:

- Sign `test` object with your certificate
- Check the signature of your object

1.4 Jar's signature with jarsigner

In this step, you have to sign a jar file with your certificate. You will use the tool `jarsigner`. You can find additional documentation here:
http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jarsigner.html
You have to:

- Generate a jar from the `SqueletonmyKeystore`
- Sign it with `jarsigner`
- Check the signature with `jarsigner`

Make a screenshot of all the commands and results with `jarsigner`.

2. Extensible Authentication Protocol (EAP)

Objective:

The goal of this exercise is to implement the EAP protocol, for an authentication based on MD5 and

Information in EAP:

The EAP protocol is organized around two main phase: (i) **identity exchange**, and (ii) **challenge-response exchange**. Following EAP terminology, messages are exchanged between an authenticator (system in charge of authentication), and a supplicant (the system to be authenticated). The authenticator is the entity in charge of the authentication, and the supplication the entity to be authenticated.
In the **identity exchange**, the authenticator initiates the EAP exchange. It requests the identity of the supplicant. The supplicant sends its identity to the authenticator.
At the **challenge-response** step, the authenticator sends a challenge to the supplicant. The supplicant has to perform a given operation on the random value in order to prove its identity. As depicted in Figure 1, the supplicant can be asked to perform a MD5 on a random value.
The authenticator then verifies if the challenge has been completed successfully by the supplicant. If it is the case, the authenticator sends a success message to the supplicant.

The message flow is depicted in Figure 1, where the authentication is based on a MD5 challenge response.

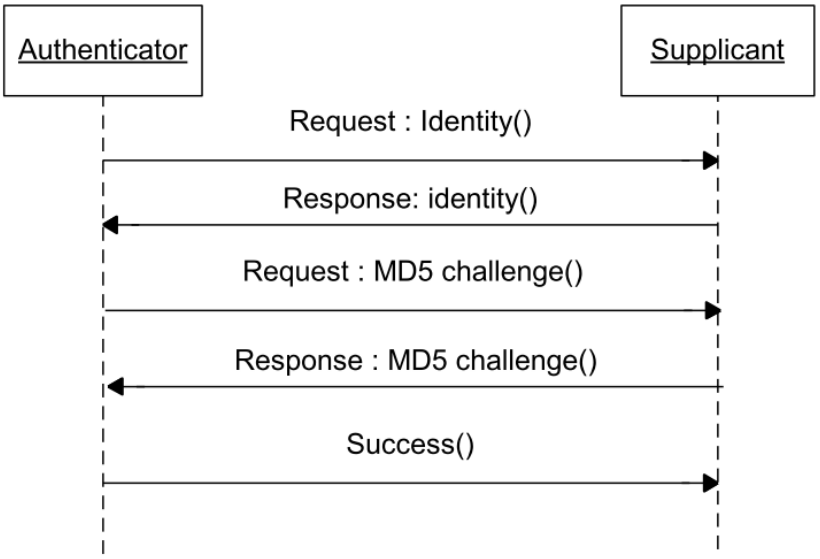

Figure 1. EAP Message flow

Identity exchange

In this step, the authenticator and supplicant exchange their identity. In figure 1., the authenticator send its identity with Request:Identity(). The supplicant replies with a Response:Identity().

Challenge-response exchange

Once supplicant and authenticator have exchanged their identity, the authenticator sends a challenge to the supplicant. In Figure 1., the authenticator sends a random String (Request :MD5-Challenge). The supplication hashes the challenge with MD5 and sends it back to the authenticator (Response:MD5-Challenge). The authenticator then computes the MD5 hash, and checks if it equals the MD5-hash received from the supplicant.

In case of successful authentication, the authenticator sends an EAP-Success, otherwise a EAP-Failure.

EAP Packet structure

The structure of EAP packet is depicted in Figure 2.

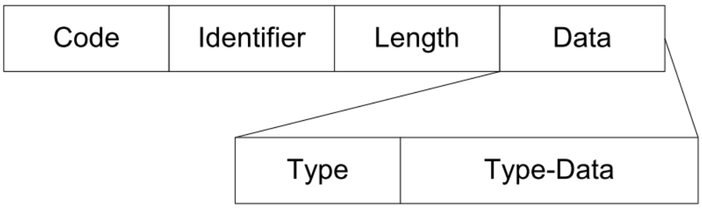

Figure 2. EAP Packet Structure

Code field

Code field describes the type of packet

- EAP-request
- EAP-response
- EAP-failure
- EAP-success

Identifier field

Identifier field provides a packet identifier, which is used for the binding between a request and its response. The two packets will have the same **identifier field.**

Length field

Length field is the size of the **data field.**

Data field

Data field is composed of two fields: **type** and **type-data**.
Type characterizes the type of data. The EAP's RFC defines a set of data types:

- **Identity**
- **Notification**
- **NAK**
- **EAP-MD5, EAP-OTP, EAP-GTC ,EAP-PAP ,EAP-TLS, LEAP, EAP-TTLS, PEAP**

Type-data contains the data of type **Type**.

EAP Packets with MD5 challenge response

Message	Code	Type	Type-Data
Request:Identity()	EAP-Request	Identity	Name
Response:Identity	EAP-Response	Identity	Name
Request:MD5Challenge()	EAP-Request	EAP-MD5	Challenge
Response:MD5Challenge()	EAP-Response	EAP-MD5	MD5 hash
Success		EAP-Success	
Failure		EAP-Failure	

Exercises:

The project EAP contains the following java class:

- `Frame`: EAP Packet
- `Data`: EAP Data field
- `Authenticator`
- `Supplicant`

2.1. Implement the EAP-MD5 protocol

In this step, you have to modify `Supplicant.authenticate()` and `Authenticator.handleFrame(Frame)` methods. You have to implement the identity and challenge-response steps as described above.

2.2. Implement the EAP-TLS protocol

The EAP-TLS protocol is based on the supplicant's certificate.
At identity step, the supplicant sends its certificate to the authenticator.
At challenge-response step, the supplicant has to sign a random value provided by the authenticator.
The latter verifies the signature of the supplicant based on the exchange certificate.

You can find below the request and response code for EAP-TLS.

Message	Code	Type	Type-Data
Request:Identity()	EAP-Request	Identity	Name
Response:Identity	EAP-Response	Identity	Name
Request:TLSChallenge()	EAP-Request	EAP-TLS	Challenge
Response:TLSChallenge()	EAP-Response	EAP-TLS	Signature
Success		EAP-Success	
Failure		EAP-Failure	