Si vuole realizzare una applicazione client-server che sfrutti le funzionalità offerte dal protocollo OpenSSL per garantire una comunicazione sicura e l'identità delle parti coinvolte.

In particolare, si vuole implementare l'autenticazione lato server, ovvero:

- creare una mini *Certification Authority*
- creare un *server certificato*, sfruttando i comandi di `openssl`
- creare un *client* che, prima di iniziare il trasferimento dei dati, controlli l'identità del server verificando che il certificato personale inviato dal server sia firmato da una *CA* (*Certification Authority*) che riconosce

Il *server* verrà realizzato mediante i comandi `openssl`, mentre il *client* sarà realizzato in *Java* utilizzando le librerie `javax.net.ssl`.

Procederemo in 4 passi:

A) Creazione del certificato per la *Certification Authority*

B) Creazione del certificato per il s*erver*

C) Implemenatzione del *client Java*

D) Avvio del *server* e del *client*

A) Creazione del certificato per la *Certification Authority*

A meno che non abbiate un contratto con una Certification Authority occorre generare una chiave e un certificato 'self-signed' anche per la CA.

A.1) Creazione di una chiave RSA per la CA

```
$ openssl genrsa -aes256 -out ca.key 2048
```

A.2) Creazione di un CSR (*Certificate Signing Request*), cioè un file di testo criptato che viene utilizzato per la creazione e l'assegnazione di un certificato SSL. In questo file sono contenute tutte le informazioni che la CA utilizza per creare il certificato SSL vero e proprio, per tanto è obbligatorio allegare un CSR ad ogni richiesta di certificato SSL

```
$ openssl req -new -key ca.key > ca.csr
```

A.3) Creazione di un certificato CA X.509 privato valido per 100 giorni che assieme alla propria chiave potrà essere utilizzato per rilasciare certificati server

```
$ openssl x509 -req -days 100 -trustout -signkey ca.key < ca.csr > ca.cert
```

B) Creazione del certificato per il server

B.1) Creazione di una chiave RSA per il server

```
$ openssl genrsa -aes256 -out server.key 2048
```

B.2) Creazione di un CSR (*Certificate Signing Request*)

```
$ openssl req -new -key server.key -out server.csr
```

B.3) Richiesta di certificato per il server valido per 100 giorni

```
$ openssl x509 -req -days 100 -CA ca.cert -CAkey ca.key -set_serial 1 < server.csr > server.cert
```

C) Implemenatzione del client Java

Un client Java SSL che si connette ad un SSL server riceve un certificato di autenticazione dal server e deve validarlo rispetto ad un insieme di certificati presenti nel suo *truststore*. Il *truststore* di default è *<java-home>/lib/security/cacerts*. In questo caso è possibile aggiungere il certificato del server in un *nuovo truststore* prima che si stabilisca una connessione tra le parti.

C.1) Il seguente comando permette di creare un keystore tramite il comando **keytool**

```
$ keytool -import -alias alias_name -file ca.cert -keystore /path_to_the/my_key_store
```

C.2)

```
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.*;

public class Client {
    public static void  main(String[] arstring) {
     try {
      SSLSocketFactory sf = (SSLSocketFactory) SSLSocketFactory.getDefault();
      SSLSocket sslsocket = (SSLSocket) sf.createSocket("localhost", 4433);
      InputStream inputstream = System.in;
      InputStreamReader isreader = new InputStreamReader(inputstream);
      BufferedReader bufferedreader = new BufferedReader(isreader);
      OutputStream outputstream = sslsocket.getOutputStream();
      OutputStreamWriter oswriter = new OutputStreamWriter(outputstream);
      BufferedWriter bufferedwriter = new BufferedWriter(oswriter);
      String string = null;
      while ((string = bufferedreader.readLine()) != null) {
            bufferedwriter.write(string + '\n');
            bufferedwriter.flush();
            }
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}
```

D) Avvio del *server* e del *client*

D.1)

```
$ openssl s_server -key server.key -cert server.cert
```

D.2)

```
$ java -Djavax.net.ssl.trustStore=./path_to/my_key_store
-Djavax.net.ssl.trustStorePassword=key_store_password
-Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol
-Djavax.net.debug=ssl Client
```

Altri comandi utili:

```
$ keytool -list -keystore nome_key_store
```

usato per elencare il contenuto del `keystore`, quindi i certificati installati (richiede la conoscenza della password settata durante la generazione).

```
$ keytool -printcert -file ca.cert
```

per visualizzare il certificato stampato in un file