Assignment 1

Assigned: 21/10/25
Due: 30/10/25

In this assignment your team is going to implement a secure server-client communication using OpenSSL in C. The purpose of this assignment is to provide you the opportunity to get familiar with the internals and implementations of client-server communication, SSL and OpenSSL.

Basic Theory

What is SSL?

An SSL (Secure Sockets Layer) is the standard security protocol used to establish an encrypted connection between a server and a client. After establishing the connection SSL/TLS ensures that the data transmitted between server and client are secured and intact.

SSL is designed to exchange sensitive data over the network using some secure algorithms and prevent another program that wants to access the private data from the network connection. SSL uses **asymmetric encryption algorithms** to secure the transmission of data. These algorithms use the pair of keys (**public and private**). The **public key** is **freely available** and known for anybody. The **private key** is only known by the **server or the client**.In SSL data encrypted by the **public key can only decrypt by the private key** and the data encrypted by the **private key can only decrypt by the public key**.

In the SSL communication, the client starts the connection from the first hello (SSL) message. This hello message starts the negotiation and performs the handshaking between server and client. After completing the handshaking if everything is fine then generate a secured key for the current connection. The server and client have used this secret key in data exchanging.

<u>SSL handshake flow</u>

The SSL handshake is an authentication process. In which server and client authenticate to each other using a certificate. This certificate is generated by the user himself with the help of **OpenSSL commands.**

Steps of Handshaking between client and server:

1. In the beginning of the communication, SSL/TLS client sends a “client_hello” message to the server. This message contains all the cryptographic information which is supported by the client, like the highest protocol version of SSL/TLS, encryption algorithm lists, data compression method, resume session identifier, and randomly generated data (which will be used in symmetric key generation).
2. The SSL/TLS server responds with a “server_hello” message to provide all the necessary information to establish a connection like protocol version used, data compression algorithms and encryption method selected, assigned session id and random data (which will be used in symmetric key generation).
3. The server sends a certificate to the client and also inserts a request message for the client certificate because the server requires the client certificate for the mutual authentication.
4. The SSL or TLS client verifies the server’s digital certificate.
5. If the SSL or TLS server sent a “client certificate request”, the client sends a random byte string encrypted with the client’s private key, together with the client’s digital certificate, or a “no digital certificate alert”. This alert is only a warning, but with some implementations, the handshake fails if client authentication is mandatory.
6. The SSL or TLS client sends the randomly generated data that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The randomly generated data itself is encrypted with the server’s public key.
7. The SSL or TLS server verifies the client’s certificate.
8. The SSL or TLS client sends the server a “finished” message, which is encrypted with the secret key, indicating that the client part of the handshake is complete.
9. The SSL or TLS server sends the client a “finished” message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.
10. For the duration of the SSL or TLS session, the server and client can now exchange messages that are symmetrically encrypted with the shared secret key.

Secure Server-Client Program using OpenSSL in C (with Mutual TLS)

In this assignment your team will create a secure connection between client and server using the **TLS1.2** protocol and the **OpenSSL** library in C.
Both client and server will authenticate each other using **X.509 certificates (mutual TLS).**
The client will send an XML login request (username + password) to the server.
The server will validate the request and respond with either a valid XML response or an “Invalid Message” reply.
You will then test what happens if a **rogue (unauthorized) client** tries to connect.

Assignment Steps:

Install the OpenSSL library, for the ubuntu.

```
sudo apt update
sudo apt install openssl libssl-dev
```

Create a Certificate Authority (CA) and use it to sign both server and client certificates.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout ca.key -out ca.crt   -subj
"/C=GR/ST=Crete/L=Chania/O=TechnicalUniversityofCrete/OU=ECE
Lab/CN=RootCA"
```

Generate Server Certificate (signed by CA)

```
# 1. Create server key and CSR
openssl req -new -newkey rsa:2048 -nodes -keyout server.key \
  -out server.csr -subj
"/C=GR/ST=Crete/L=Chania/O=TechnicalUniversityofCrete/OU=ECE
Lab/CN=localhost"

# 2. Sign with CA
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key \
  -CAcreateserial -out server.crt -days 365 -sha256
```

Generate Client Certificate (signed by CA)

```
# 1. Create client key and CSR
openssl req -new -newkey rsa:2048 -nodes -keyout client.key \
  -out client.csr -subj
"/C=GR/ST=Crete/L=Chania/O=TechnicalUniversityofCrete/OU=ECE
Lab/CN=client"

# 2. Sign with CA
openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key \
  -CAcreateserial -out client.crt -days 365 -sha256
```

You are given two starter files, with TODOs to fill.

server.c — TLS server skeleton (requires completing missing OpenSSL setup functions and mutual TLS)

client.c — TLS client skeleton (requires certificate and handshake implementation)

Implement one more client, rclient.c (rogue client), which will try to communicate with the server with a rogue certificate (signed by unknown/untrusted CA).
For this step:

1. Copy the existing client.c skeleton
2. Generate Client Certificate (signed by another CA)

Compile the Server: **gcc -Wall -o server server.c -L/usr/lib -lssl -lcrypto**
Compile the Client : **gcc -Wall -o client client.c -L/usr/lib -lssl -lcrypto**
Compile the Rogue Client : **gcc -Wall -o rclient rclient.c -L/usr/lib -lssl -lcrypto**

Or you may use the default Makefile given.

Requirements

1. First run the server. Example: **./server 8082**
    a. What is the number 8082?
    b. Can you run it on number 80, 443, 124? How can you achieve it?

2. Then run the client. Example: **./client 127.0.0.1 8082**
    a. What is 127.0.0.1?
    b. What is 8082?
3. If the client sends a **valid** request then server gives a proper response like then one below:
    a. Client Request:

```
<Body>
        <User>Sousi</UserName>
        <Password>123</Password>
</Body>
```

    b. Server Response:

```
<Body>
        <Name>sousi.com</Name>
        <year>1.5</year>
        <BlogType>Embedede and c c++</BlogType>
        <Author>John Johny</Author>
</Body>
```

4. If the client sends a **invalid** request to the server, then the server responds to an **"Invalid message".**
    a. Client Request:

```
<Body>
        <UserName>Sousi</UserName>
        <Password>12345</Password>
</Body>/
```

    b. Server Response:

```
"Invalid Message"
```

5. If the **rogue** client sends a request to the server, then the server responds that the peer did not return a certificate or a valid one
    a. Server Response: "peer did not return a certificate or returned an invalid one"

Useful Links

OpenSSL Library: https://www.openssl.org/

Important Notes

1. You need to submit the client.c, rclient.c(rogue) and server.c, a README file and a Makefile. The README file should briefly describe your tool and the commands that generated and signed the certificates (knowledge of each parameter). You should place all these files in a folder named <AM>_assign1 and then compress it as a .zip file.

For example, if your login is 20251017 the folder should be named 20251017_assign1 you should commit 20251017_assign1.zip.

2. **Google** your questions first.
3. Use the tab “Συζήτηση” in courses for questions.
4. Do not copy-paste code from online examples, we will know ;)