**Files Included:**

MeetingRoomServerFinal.java – The final version of the server program for the meeting room reservations.

MeetingRoomClientFinal.java – The final version of the client program for the meeting room reservations.

HandshakeMessage.java – A program that can be used by the client and server for the cryptography package parameters.

PasswordHash.txt – Used to hash the original login credentials passwords into bcrypt hashed password and username combinations.

ClientPublicKey.txt – Contains the client’s public key.

Client ServerKey.txt – Contains the client’s private key.

ServerPublicKey.txt – Contains the server’s public key.

ServerPrivateKey.txt – Contains the server’s private key.

SPublicKey.txt – Contains the server’s public key, client creates this document when receiving the public key sent from the server.

CPublicKey.txt – Contains the client’s public key, server creates this document when receiving the public key sent from the server.

LoginCreds.txt – Contains the usernames and their bcrypt hashed passwords in the format username:bcrypthashedpassword

MeetingTimes.txt – Contains the available and reserved meeting times in the format timeslot:username

**For the certificate exchange(not uploaded due to size of files):**

clientKeyStore.jks – Client’s certificate

serverKeyStore.jks – Server’s certificate

keystoreCA.jks – Certificate Authority to use for validation.

**1.**

The handshake message will be created by the client and use the HandshakeMessage.java program to create the object that is then sent to the server. If the server accepts the parameters outlined by the client then they will be saved into the variables to be used moving forward by the client and server as the cryptography package. The client and server will exchange their public keys to use for the key exchange protocol. The server will print out when the handshake message is received from the client(“Received handshake message from client) and if it is accepted it will proceed forward with sending the public key to the client and when it is sent and the server receives the client’s public key. When this process is complete the server will print out “Handshake Protocol Complete”. Then the client and server will exchange digital certificates and verify them through the following messages to ensure mutual authentication:

Client Digital Certificate Received

Checking Digital Certificate...

Client Digital Certificate Verified

The Key exchange protocol will then be started upon successful completion of the certificate exchange and verification.

Key Exchange Protocol Started...

The key exchange protocol will follow these 4 steps for the secure exchange of the session key. Replay attacks are defended against using the nonces, N1 and N2. As the steps are sent or received, they are printed out in the client and server’s console.

Client:

Key Exchange Protocol Started...

Step 1 Sent

Step 2 Received and N1 Verified

Step 3 Sent

Step 4 Sent

Key Exchange Protocol Complete

Server:

Key Exchange Protocol Started...

Step 1 Received

Step 2 Sent

Step 3 Received and N2 Verified

Step 4 Received, SessionKey Decrypted  
Key Exchange Protocol Complete

As seen above, if the steps are completed and the session key has been established between the two parties, “Key Exchange Protocol Complete” will be printed out to both parties to show that the exchange was successful, a session key has been established, and the parties are ready to move forward with the reservations using secure communication.

All further communication is encrypted using the agreed upon cryptography package and the established session key before sending to the other party to then be decrypted using the session key.

The client will then be prompted to login, if they are unsuccessful they will be prompted to login again. If they fail the login 5 times, the program will exit.

Upon successful login the user will be presented the available time slots, if there are none they will be notified that there are no available times and the program will exit.

When the user picks a timeslot and it is reserved they will be notified and the program will ask if the user would like to make another reservation, if they send “n” the program will exit, if they send “y” the program will keep the connection open and prompt the user for their login.

**2.**

Provided below are a few pictures of the client and server console windows during the reservation process showing the application running properly.

Successful Handshake Protocol, Key Exchange, and Digital Certificate Verification:

Client:  
Graphical user interface, text, application, email

Description automatically generated

Server:  
Graphical user interface, text, application, email

Description automatically generated

Client unsuccessful after 5 login attempts, program exits:

Client:  
Graphical user interface, text, application

Description automatically generated

Server:  
Graphical user interface, text, application, email

Description automatically generated

Successful login and timeslots not available:

Client:  
Graphical user interface, text, email

Description automatically generated

Server:  
Text

Description automatically generated with medium confidence

Client login successful and times displayed with successful reservation and exit:

Client:  
Graphical user interface, text, application, email

Description automatically generated

Server:  
Graphical user interface, text

Description automatically generated with medium confidence

Client login successful and times displayed with successful reservation and wants to make another:

Client:  
Graphical user interface, text

Description automatically generated with medium confidence

Server:  
A picture containing graphical user interface

Description automatically generated

Reflection:

I think that this was a great project to work on, there were many times where I ran into obstacles that required out of the box thinking and trying different ideas to overcome. This was my first time working on having two java programs communicate with each other so I believe that there is some optimization that could take place to clean the code up if there are redundant lines. I really enjoyed learning about and implementing the key exchange protocol to establish the use of the session key, the idea of using nonces to defend against replay attacks I think is especially interesting because it is a slight change that can prevent a very intrusive attack.

**3.**

**Wireshark:**

I am not 100% sure what we should be seeing in Wireshark, but using the loopback traffic capture on Wireshark with the filter tcp.port == 755, which is the port that my java program is using I was able to see some communication between the client and server.

I was able to see the handshake message which was not encrypted:

Graphical user interface, text, application, email

Description automatically generated

There was also a packet that contained the agreed upon cryptographic package parameters:

Graphical user interface, text, application

Description automatically generated

I think that the 4 packets with the length of 390 are the exchange of the public keys, packets 7604, 7606, 7608, 7610. Here is an example of the content of one of the packets.  
Graphical user interface, application, table

Description automatically generated

The rest of the packets after that are smaller which makes sense given that the sent data is smaller, there is no plaintext visible in the packets which is to be expected and confirms that the communication between the client and server are encrypted like they should be using the established session key.

Here is an example screenshot of one of the packets.

Graphical user interface, text, application

Description automatically generated

I think this is an example of something encrypted being sent from the client to the server.