

Project  
Gunnar Yonker

**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
```

## Project

Gunnar Yonker

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.

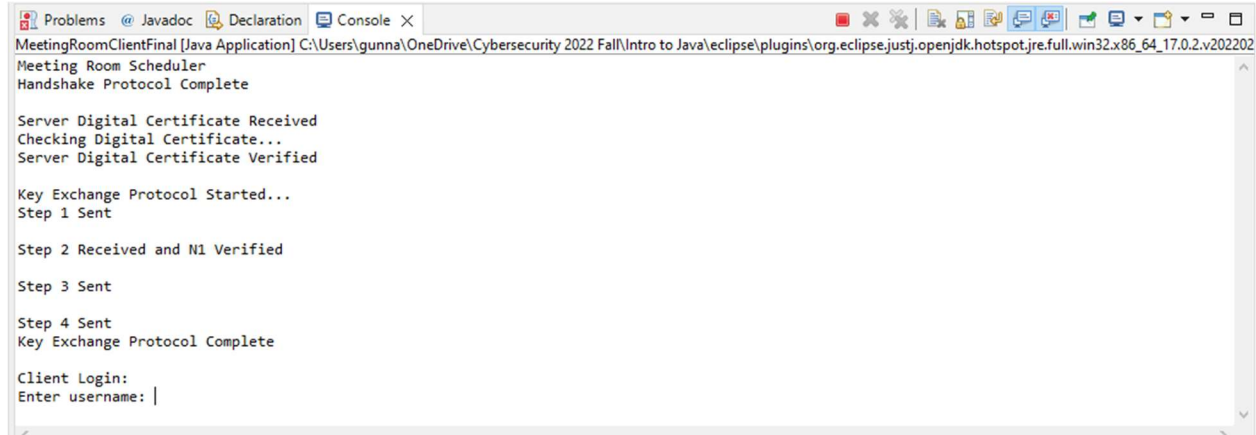
When the user picks a timeslot and it is reserved they will be notified and the program will ask if the user would like to make another reservation, if they send “n” the program will exit, if they send “y” the program will keep the connection open and prompt the user for their login.

## 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:



```
MeetingRoomClientFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v202202
Meeting Room Scheduler
Handshake Protocol Complete

Server Digital Certificate Received
Checking Digital Certificate...
Server Digital Certificate Verified

Key Exchange Protocol Started...
Step 1 Sent

Step 2 Received and N1 Verified

Step 3 Sent

Step 4 Sent
Key Exchange Protocol Complete

Client Login:
Enter username: |
```

## Project Gunnar Yonker

### Server:

```
MeetingRoomServerFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201
Meeting Room Server up and running, waiting on port 755...
Received handshake message from client
Handshake Protocol Complete

Client Digital Certificate Received
Checking Digital Certificate...
Client Digital Certificate Verified

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
Client logging in...
|
```

Client unsuccessful after 5 login attempts, program exits:

### Client:

```
<terminated> MeetingRoomClientFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201
Key Exchange Protocol Complete

Client Login:
Enter username: login
Enter password: test
Login failed, try again
Enter username: username
Enter password: password
Login failed, try again
Enter username: guest
Enter password: login
Login failed, try again
Enter username: hacker
Enter password: attack
Login failed, try again
Enter username: gunnar
Enter password: password
Login failed, try again
Max login attempts reached, try again later.|
```

### Server:

```
MeetingRoomServerFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201
Received handshake message from client
Handshake Protocol Complete

Client Digital Certificate Received
Checking Digital Certificate...
Client Digital Certificate Verified

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
Client logging in...
Client Disconnected, restarting...
Meeting Room Server up and running, waiting on port 755...|
```

## Project Gunnar Yonker

Successful login and timeslots not available:

Client:

```
Problems @ Javadoc Declaration Console X
<terminated> MeetingRoomClientFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
Checking Digital Certificate...
Server Digital Certificate Verified

Key Exchange Protocol Started...
Step 1 Sent

Step 2 Received and N1 Verified
|
Step 3 Sent

Step 4 Sent
Key Exchange Protocol Complete

Client Login:
Enter username: gunnar
Enter password: student
Login successful

No available times
```

Server:

```
Problems @ Javadoc Declaration Console X
MeetingRoomServerFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220220

Client Digital Certificate Received
Checking Digital Certificate...
Client Digital Certificate Verified

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
Client logging in...

Client logged in, showing available times...
Client Disconnected, restarting...
Meeting Room Server up and running, waiting on port 755...
```

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

Client:

```
Problems @ Javadoc Declaration Console X
<terminated> MeetingRoomClientFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
Step 4 Sent
Key Exchange Protocol Complete

Client Login:
Enter username: gunnar
Enter password: student
Login successful

Available meeting times:
10am
11am
2pm
4pm

Enter desired meeting time: 2
Invalid time, please enter a valid time from the list
Enter desired meeting time: 2pm
Meeting time slot reserved at 2pm for gunnar
Do you want to make another reservation (y/n)? n
```

## Project

### Gunnar Yonker

#### Server:

```
Problems @ Javadoc Declaration Console X
MeetingRoomServerFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe
Client Digital Certificate Received
Checking Digital Certificate...
Client Digital Certificate Verified

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
Client logging in...

Client logged in, showing available times...
Client timeslot reserved
Client Disconnected, restarting...
Meeting Room Server up and running, waiting on port 755...
```

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

#### Client:

```
Problems @ Javadoc Declaration Console X
MeetingRoomClientFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe
Client Login:
Enter username: gunnar
Enter password: student
Login successful

Available meeting times:
10am
11am
4pm

Enter desired meeting time: 10am
Meeting time slot reserved at 10am for gunnar
Do you want to make another reservation (y/n)? y
Enter username: gunnar
Enter password: student
Login successful

Available meeting times:
11am
4pm

Enter desired meeting time:
```

#### Server:

```
Problems @ Javadoc Declaration Console X
MeetingRoomServerFinal [Java Application] C:\Users\gunna\OneDrive\Cybersecurity 2022 Fall\Intro to Java\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe
Received handshake message from client
Handshake Protocol Complete

Client Digital Certificate Received
Checking Digital Certificate...
Client Digital Certificate Verified

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
Client logging in...

Client logged in, showing available times...
Client timeslot reserved

Client logged in, showing available times...
```

Project  
Gunnar Yonker

## Reflection:

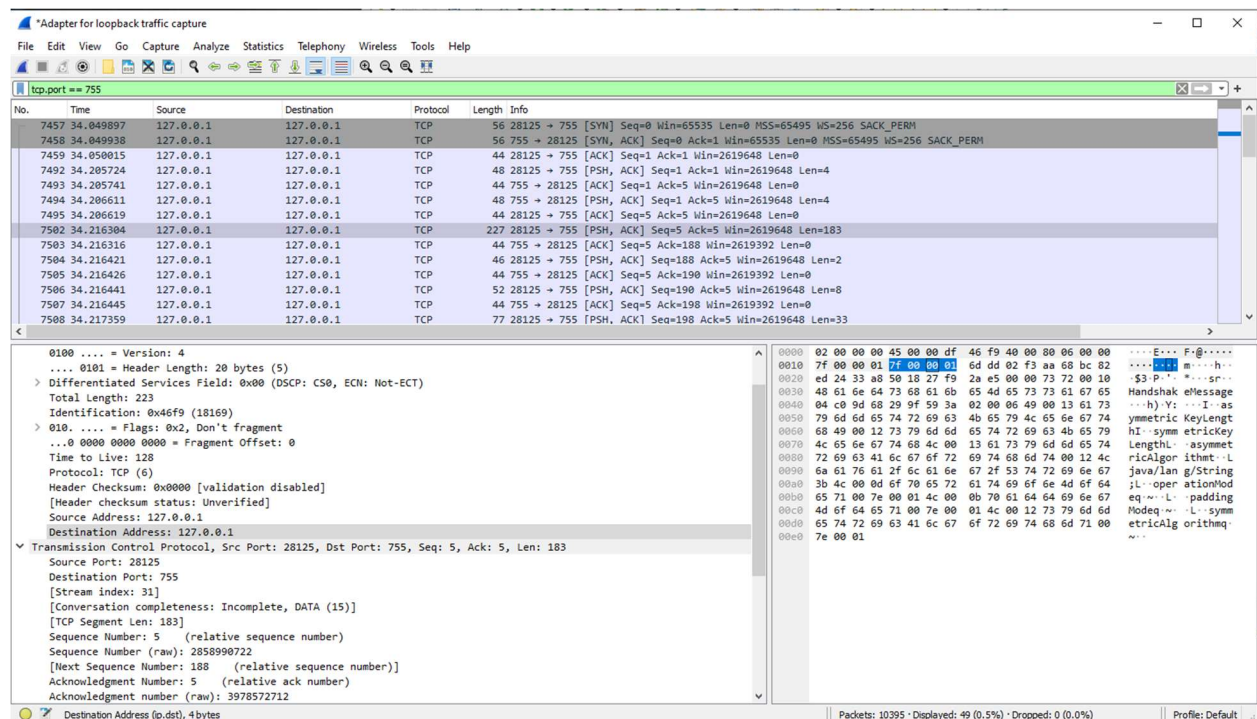
I think that this was a great project to work on, there were many times where I ran into obstacles that required out of the box thinking and trying different ideas to overcome. This was my first time working on having two java programs communicate with each other so I believe that there is some optimization 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:





# Project

## Gunnar Yonker

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

The image shows a Wireshark packet capture window. The top pane displays a list of captured packets. The bottom pane shows the detailed view of a selected packet, which is a TCP segment. The packet details are as follows:

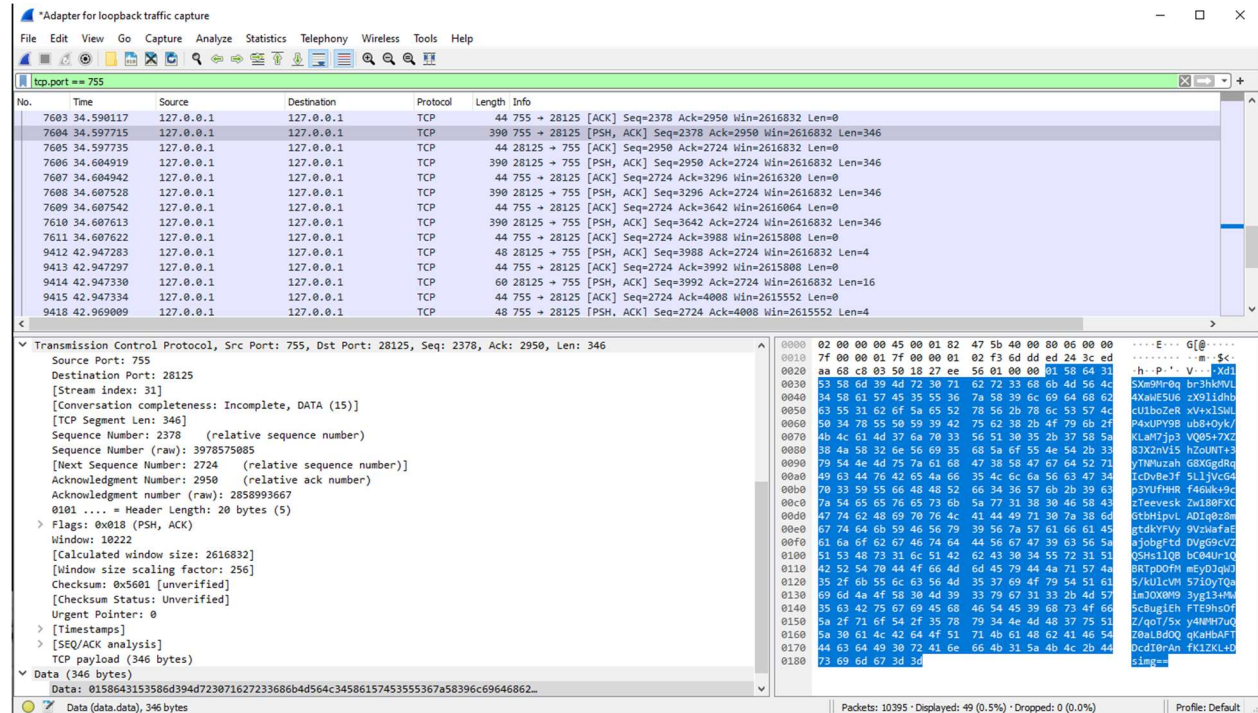
- 0100 .... = Version: 4
- .... 0101 = Header Length: 20 bytes (5)
- > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
- Total Length: 73
- Identification: 0x46ff (18175)
- > 010. .... = Flags: 0x2, Don't fragment
- ...0 0000 0000 0000 = Fragment Offset: 0
- Time to Live: 128
- Protocol: TCP (6)
- Header Checksum: 0x0000 [validation disabled]
- [Header checksum status: Unverified]
- Source Address: 127.0.0.1
- Destination Address: 127.0.0.1
- ▼ Transmission Control Protocol, Src Port: 28125, Dst Port: 755, Seq: 198, Ack: 5, Len: 33
- Source Port: 28125
- Destination Port: 755
- [Stream index: 31]
- [Conversation completeness: Incomplete, DATA (15)]
- [TCP Segment Len: 33]
- Sequence Number: 198 (relative sequence number)
- Sequence Number (raw): 2858990915
- [Next Sequence Number: 231 (relative sequence number)]
- Acknowledgment Number: 5 (relative ack number)
- Acknowledgment number (raw): 3978572712

The packet bytes pane shows the raw data of the packet, including the header and payload. The payload is a 15-byte data segment.



Project  
Gunnar Yonker

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.



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.

# Project

## Gunnar Yonker

Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 755

No.	Time	Source	Destination	Protocol	Length	Info
7611	34.607622	127.0.0.1	127.0.0.1	TCP	44	755 → 28125 [ACK] Seq=2724 Ack=3988 Win=2615808 Len=0
9412	42.947283	127.0.0.1	127.0.0.1	TCP	48	28125 → 755 [PSH, ACK] Seq=3988 Ack=2724 Win=2616832 Len=4
9413	42.947297	127.0.0.1	127.0.0.1	TCP	44	755 → 28125 [ACK] Seq=2724 Ack=3992 Win=2615808 Len=0
9414	42.947330	127.0.0.1	127.0.0.1	TCP	60	28125 → 755 [PSH, ACK] Seq=3992 Ack=2724 Win=2616832 Len=16
9415	42.947334	127.0.0.1	127.0.0.1	TCP	44	755 → 28125 [ACK] Seq=2724 Ack=4008 Win=2615552 Len=0
9418	42.969009	127.0.0.1	127.0.0.1	TCP	48	755 → 28125 [PSH, ACK] Seq=2724 Ack=4008 Win=2615552 Len=4
9419	42.969020	127.0.0.1	127.0.0.1	TCP	44	28125 → 755 [ACK] Seq=4008 Ack=2728 Win=2616832 Len=0
9420	42.969042	127.0.0.1	127.0.0.1	TCP	60	755 → 28125 [PSH, ACK] Seq=2728 Ack=4008 Win=2615552 Len=16
9421	42.969046	127.0.0.1	127.0.0.1	TCP	44	28125 → 755 [ACK] Seq=4008 Ack=2744 Win=2616832 Len=0
9422	42.969345	127.0.0.1	127.0.0.1	TCP	48	755 → 28125 [PSH, ACK] Seq=2744 Ack=4008 Win=2615552 Len=4
9423	42.969352	127.0.0.1	127.0.0.1	TCP	44	28125 → 755 [ACK] Seq=4008 Ack=2748 Win=2616832 Len=0
9424	42.969368	127.0.0.1	127.0.0.1	TCP	60	755 → 28125 [PSH, ACK] Seq=2748 Ack=4008 Win=2615552 Len=16
9425	42.969371	127.0.0.1	127.0.0.1	TCP	44	28125 → 755 [ACK] Seq=4008 Ack=2764 Win=2616832 Len=0

< Frame 9419: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF\_{Loopback}, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 28125, Dst Port: 755, Seq: 4008, Ack: 2728, Len: 0

0000 02 00 00 00 45 00 00 28 4e 4e 40 00 00 06 00 00 .....E..(NN@.....

0010 7f 00 00 01 7f 00 00 01 6d dd 02 f3 aa 68 cc 25 .....m....h-%

0020 ed 24 3e 4b 50 10 27 ee 77 15 00 00 .....\$KP-.-w....

wireshark\_NPF\_LoopbackSJAM11.pcapng

Packets: 10395 · Displayed: 49 (0.5%) · Dropped: 0 (0.0%)

Profile: Default

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