

CS 6349.001 - Network Security

Programming Project

Instant Messaging (IM) System

Aishwarya Singh – AXS200109

Sahil Prashant Kirpekar – SXX200032

Shubham Singh – SXS200413

Vaibhav Tyagi – VXT200018

Problem Statement

The problem statement is to implement a secure Instant Messaging (IM) system that includes a client program and server program.

This report emphasizes on how we have implemented our system, the functional capabilities, security features, threat model, and the attacks considered while designing the system.

The requirements assume that the Server is successfully authenticated, and we need to implement a secure authentication channel for the clients. The objective is to build a communication protocol that is secure against well-known attacks on authentication, confidentiality, and integrity using available security primitives.

In summary, we have implemented the system in Java programming language running on the IntelliJ environment. We have a server program that generates private and public keys and multiple instances of a client program that helps run a secure communication channel using a shared session key generated by the server after successful authentication.

We have used AES 128-bit for the encryption of keys and messages. We take a 128-bit encrypted bit message, encode it to UTF-8 encoding, and then send the message to the other side. The other side decodes using the encoding and then uses the same symmetric key to decrypt the message.

Now let's look at the project implementation along with their functional and security capabilities.

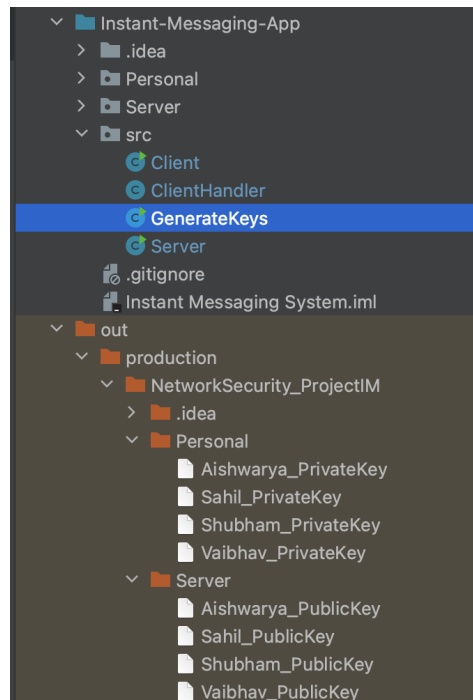
Implementation

1. Keys Generation

- Every Client has its own private key, which we generate at the start of running the project. The public key of the CliClient was also registered with the Server during the same above run.
- We generate the keys by running the file `/src/GenerateKeys`, which the keys while used by the main program and then generated in

`out/production/NetworkSecurity_ProjectIM/Server` – public keys

`out/production/NetworkSecurity_ProjectIM/Personal` – private keys



2. Server

- The Server now holds the responsibility of authenticating the Clienting the keys generated.
- The Server can hold multiple sessions for different pairs of clients in parallel.
- Once the client communication has been completed, the Server still holds the Client its authenticated list.
- The Server generates a session key which is shared between the two requested clients, thereby establishing a direct communication link between them.
- We have a function called startServer() that opens the socket to start the threads for clientHandlers in the file src/Server.java
- We will look at the work in detail in point 5.

```
public void startServer()
{
    try
    {
        while(! ss.isClosed())
        {
            Socket socket = ss.accept();
            System.out.println("New client is connected");
            ClientHandler clientHandler = new ClientHandler(socket);
            clientHandlers.add(clientHandler);
            map.put(clientHandler.clientUsername, clientHandler);
            verifyClient(clientHandler);
            clientHandler.selectClientForConversation(clientHandlers);
            Client.clientHandlers.add(clientHandler);
            System.out.println(clientHandlers.size());
            Thread thread = new Thread(clientHandler);
            thread.start();
        }
    }
}
```

3. Client

- A client can obtain a list of other clients from the Server and communicate with at most one of them.
- A client can be either in one of the following states:
 1. **Idle** – waiting for other clients to do IM
 2. **Busy** – currently involved in an IM with another Client.
- The Server will establish communication between the clients only if the other CliClient in idle state.
- Clients involved in the IM communicate directly with each other until the session is completed. Once completed, they communicate back to the Server and may reach out directly to another client.
- In one session, only two clients are allowed to participate, but there can be multiple sessions in parallel for different pairs of clients.
- The main implementation for bringing the client threads running is inside a function called run() in the src/ClientHandlers.java file.
- The messages sent are encrypted using the AES algorithm
- We will look at the work in detail in point 5.

```
public void sendMessage(){  
  
    try{  
        bufferedWriter.write(username);  
        bufferedWriter.newLine();  
        bufferedWriter.flush();  
  
        Scanner sc = new Scanner(System.in);  
        while(socket.isConnected()) {  
  
            String messageToSend = sc.nextLine();  
            // System.out.println(session_key);  
  
            // if session key is stored  
            if(session_key != null) {  
                messageToSend = encryptMessage(messageToSend, session_key);  
            }  
  
            bufferedWriter.write(messageToSend);  
            bufferedWriter.newLine();  
            bufferedWriter.flush();  
        }  
    }  
}
```

4. Security:

- The messages and the session keys used for authentication are all encrypted using the AES 128-bit algorithm.

```
public String encryptText(String msg, PrivateKey key)
    throws NoSuchAlgorithmException, NoSuchPaddingException,
        UnsupportedEncodingException, IllegalBlockSizeException,
        BadPaddingException, InvalidKeyException {
    this.cipher.init(Cipher.ENCRYPT_MODE, key);
    return Base64.encodeBase64String(cipher.doFinal(msg.getBytes( charsetName: "UTF-8")));
}
```

```
public String decryptText(String msg, PublicKey key)
    throws InvalidKeyException, UnsupportedEncodingException,
        IllegalBlockSizeException, BadPaddingException {
    this.cipher.init(Cipher.DECRYPT_MODE, key);
    return new String(cipher.doFinal(Base64.decodeBase64(msg)), charsetName: "UTF-8");
}
```

- The security requirements have been met in the following way:

The authentication mechanism uses using public key cryptosystem called RSA, as stated in the requirements.

Client:

1. Authentication:

- The program generates a private key for the registered Clients. Now, the Clients has its own private key, which it will use to authenticate itself to the Server. The Client's the initial connection request to the Server for which the Server generates a nonce to send back to the Client.

2. Confidentiality:

- We are using a public key cryptosystem called RSA. The nonce, when sent back to the Server, is encrypted with the Client's private key which is only known to the Client making the transmission secure in case of any attack.

3. Freshness:

- Every time a client must be authenticated, we will get a new session key, which will be only valid for a session, and once the client exits

the communication channel, the session key will expire, thus safeguarding it from any replay attacks.

4. Integrity:

- Using public key cryptography, we will maintain integrity as the corresponding private/public keys can encrypt/decrypt the message with the added nonce first sent by the Server.

Client-to-Server:

1. Authentication:

- The client authenticates themselves to the Server using RSA mechanism. The client's private key, along with the generated nonce sent by the Server upon a connection request, is sent back to the Server. Once authenticated, the Server shares a session key if both the requested clients are in an idle state.

2. Confidentiality:

- During authentication, all the communications between the Client and the Server are based on the RSA mechanism.
- Encryption of the messages between the Clients and the Server after authentication uses the AES mechanism. The session keys are generated, and messages are encrypted and decrypted through the AES algorithm.

3. Freshness:

- Each time the Client logs in, the Server generates a new session key, and there will be no room for any replay attacks as the Server will forget the session key once the Client logs out, and no attacker can use the replay attacks using any of the previous session keys.

4. Integrity:

- Since the session key is only known to the Server and Client, there can be no alteration in the message during the transmission, thus ensuring the integrity of the communication.

Client-to-Client:

1. Authentication:

- The Client have authenticated themselves to the Server and are ready for direct communication.

2. Confidentiality:

- Same as the above section, since we use public key cryptography, the tokens can only be decrypted by the two clients. Once establishing the

session, the two clients only know the secret key, and message confidentiality is maintained.

3. Freshness:

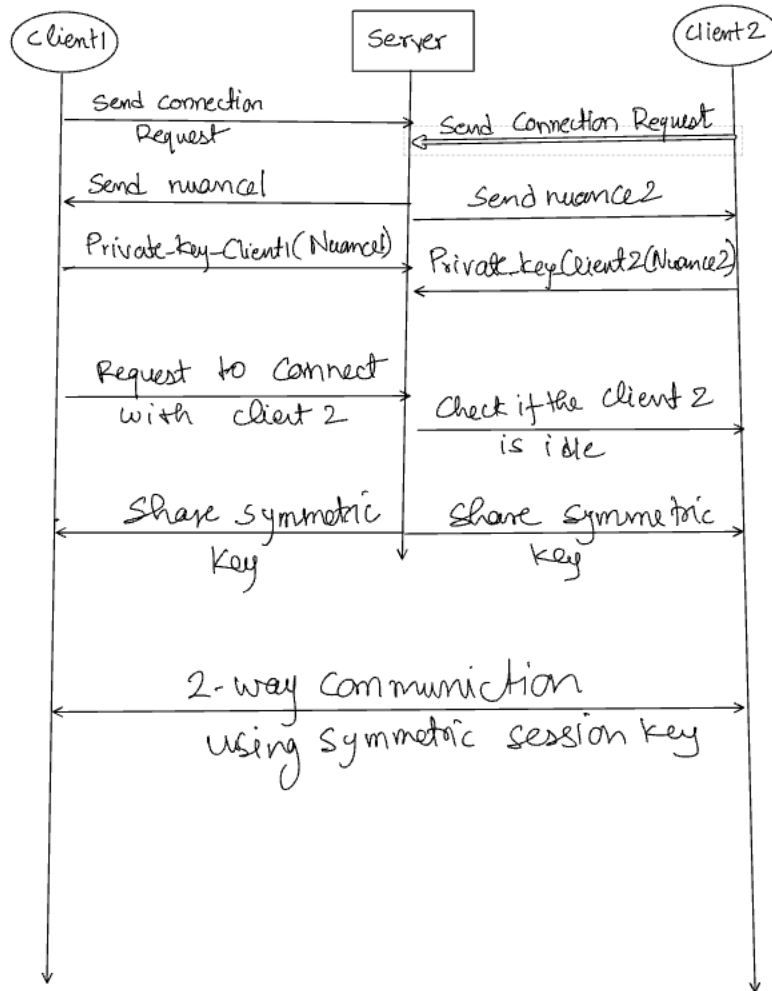
- Each time a client creates a new session, the Server generates a secret key for the clients. Since this secret key is generated randomly by the Server, there is no chance of replay attacks. We will use secret keys to ensure that if one of the clients logs out, the other Client knows of the session expiry.

4. Integrity:

- Encryption of all the messages will be done using the secret key known to the two clients, thus saving the integrity of the communication.

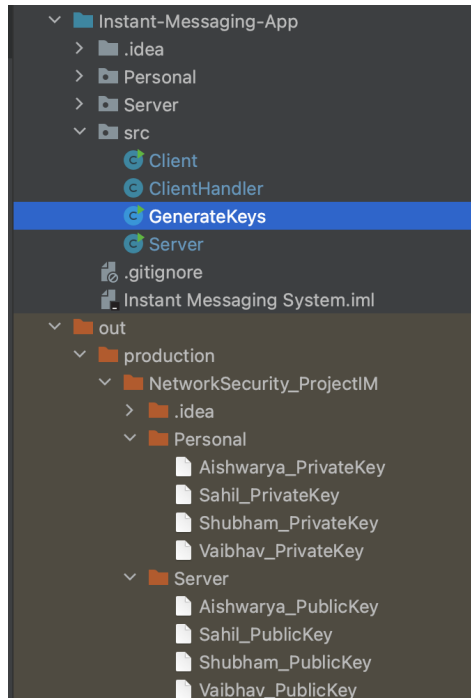
5. Working on the IM

Here is the system architecture design for our project implementation. We shall look at it step-by-step with running commands to understand it better.



Steps to do the IM:

- **Step 1:** Run the src/GenerateKeys.java file to generate private and public keys in the out/production/NetworkSecurity_ProjectIM Folder.

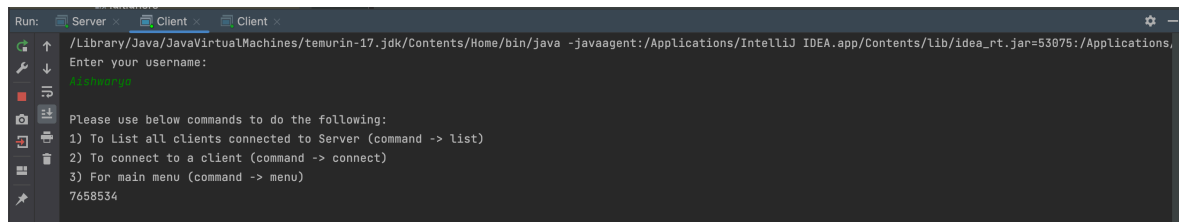


- **Step 2:** Start the Server by running the `/src/Server.java` file. This will make our Server up and running. Server is now waiting for a Client request.

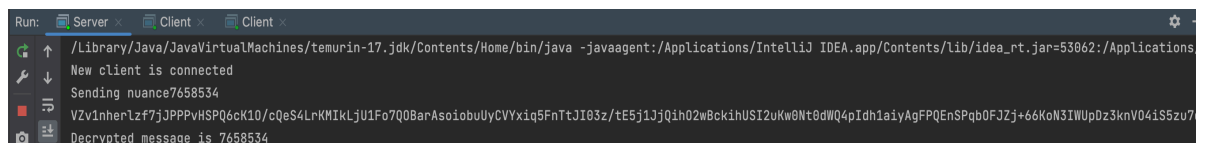


- **Step 3:** Start a Client by running `/src/Client.java`. It will ask you to enter a username. After entering it will send a connection request to the Server.

Client-side:



Server Side:



- ❖ The Client - *Aishwarya*, first sent out a connection request to the Server- “New client is connected.”
 - ❖ The Server sent out a nonce 7658534 to the Client - “Sending nonce7658534”
 - ❖ The Client on receiving the nonce sent out the nonce back to the Server encrypted with its private key – “VZv1nherlzf7jJPPpvHSPQ6cK1O/cQeS4LrKMikLjU1Fo7QOBarAsoiobuUyCVYx iq5FnTtJI03z/tE5j1JjQihO2wBckihUSI2uKw0Nt0dWQ4pldh1aiyAgFPQEnSPqb OFJZj+66KoN3IWUpDz3knVO4iS5zu7qyHxSPWenfOg=”
 - ❖ The sever successfully authenticated the Client it was able to decrypt the nonce – “Decrypted message is 7658534”
- **Step 4:** Now Authentication for one Client is successful. Let’s do the same for another Client – *Vaibhav*, by running another instance of the Client.java file. You will see a new message on Server Side that says – “New client is connected.”

Client Side:

```
Run: Server Client Client
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=53087:/Applications,
Enter your username:
Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect)
3) For main menu (command -> menu)
3380784
```

Server Side:

```
Run: Server Client Client
New client is connected
Sending nuance7658534
VZv1nherlzf7jJPPpvHSPQ6cK1O/cQeS4LrKMikLjU1Fo7QOBarAsoiobuUyCVYx iq5FnTtJI03z/tE5j1JjQihO2wBckihUSI2uKw0Nt0dWQ4pldh1aiyAgFPQEnSPqbOFJZj+66KoN3IWUpDz3knVO4iS5zu7c
Decrypted message is 7658534
New client is connected
Sending nuance3380784
Hyf/ij8PJdAeciqt5YX8kLhmJAqm+0SQh7euJ9dYIAyDhRXZhmU71QKPKfGbrCku5Br8dX5DBgrawv+Stsoc6i+tQc24vxAhLqr3rdzDb/7EAX0ePzW23x9aIM5MP69fHEygDCf8eShPfhRvBTSVRonHQ70VZM.
Decrypted message is 3380784
```

- **Step 5:** There is a list of commands that we can run to start the IM.

```
Run: Server x Client x Client x
Aishwarya
Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect)
3) For main menu (command -> menu)
7658534
list
List of Available Users:[Aishwarya, Vaibhav]

Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect client2_Name)
3) For main menu (command -> menu)
|
```

1. By running the first command *list*, we can see the list of available users in the pool -> [Aishwarya, Vaibhav]
 2. Now to connect from client Aishwarya to Vaibhav, we should run the second command -> connect client2_name, i.e., connect Vaibhav.
- **Step 6:** After we run the second command, the Server shares a secret session key with both clients, and now we establish a direct communication link between the two clients.

Server Side:

```
list
{Aishwarya=src.ClientHandler@2735500f, Vaibhav=src.ClientHandler@78b39adf}

connect Vaibhav
Wants to talk to: Vaibhav
```

Client1 (Aishwarya) side:

```
Client1 (Aishwarya)
Enter a valid choice

Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect client2_Name)
3) For main menu (command -> menu)
kfqtUYegin2yzAA87uCpv/rgmP90tNeZHpBnY2YallpUCS2z2FI8WLqrTW1lNkYLcoao3XYjyRADJ5cpYZXy0LLP3kaqwrgrtIkE2KP6+LC2K/LcJ3nNH5IXMJb10Prdx9WpNr0nqhkbvA7j1/4yg+ZWCz3hx1D
3380784
```

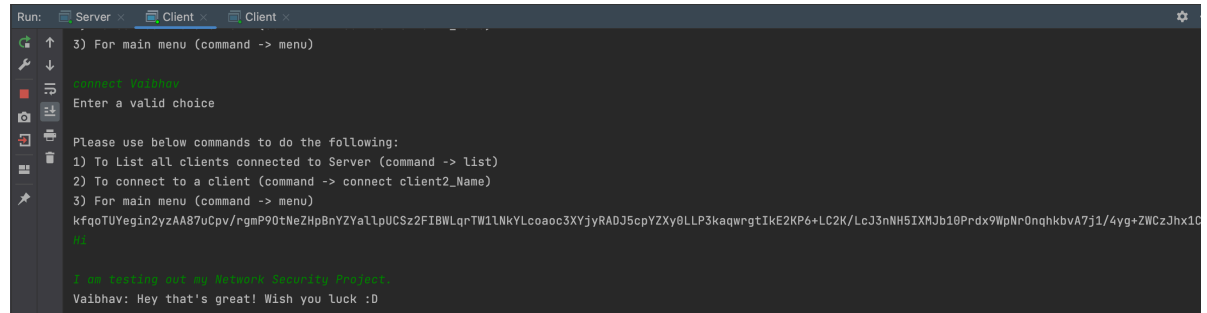
Client2 (Vaibhav) side:

```
Run: Server x Client x Client x
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=53987:/Applications
Enter your username:
Vaibhav

Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect)
3) For main menu (command -> menu)
y42CepE1Ifzfa0gq/LibbsJb7qIidQRqzNtkPVSXwtjW0rB+ps40PxiFyVkn3JhXUT4VAa0+RF5miITMlg1HhgNt8VK9z415Hd5bR+rMGhNzDxcStv+YsCfThYj7xmnd+8LHeMoXSSMRgc18QNaR8H5ekficZRO
```

- **Step 7:** We are now all set to start the IM. Let's send a message Hi from Client 1 to Client 2 and one from Client 2 to client 1.

Client 1 (Aishwarya) side:



```

Run: Server x Client x Client x
3) For main menu (command -> menu)

connect Vaibhav
Enter a valid choice

Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect client2_Name)
3) For main menu (command -> menu)
kfqoTUYegin2yzAA87uCpv/rgmP90tNeZHpBnYZYallpUCS22FIBWLqrTW11NkYLcoaac3XYjyRADJ5cpYZXy8LLP3kaqwrgtIkE2KP6+LC2K/LcJ3nNH5IXM3b18Prdx9WpNr0nqhkbvA7j1/4yg+ZWCzJhx1C
Hi

I am testing out my Network Security Project
Vaibhav: Hey that's great! Wish you luck :D
  
```

Client 2 (Vaibhav) side:



```

Run: Server x Client x Client x
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=53087:/Application
Enter your username:
Vaibhav

Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect)
3) For main menu (command -> menu)
3380784
y42CepE1Ifzfa8gq/LibbsJb7qIidQRqzNtkPVsXwtjW0rB+ps40PxiFyVkn3JhXUT4VAa0+RF5m1ITMlg1HhgNt8VK9z415Hd5bR+rMGhNzDxcStv+YsCfThYj7xmnd+8LHeMoXSSMRgc18QNaR8H5ekficZR
Aishwarya: Hi
Aishwarya:
Aishwarya: I am testing out my Network Security Project.
Hey that's great! Wish you luck :D
  
```

- **Step 8:** We have now successfully established direct communication from Client to Client. Now if one of the Client wants to exit the session, they can simply type the keyword *bye* in the CLI and return back to the Server.

Here, Client1 has entered *bye* in the chat to exit.

Client 1(Aishwarya) side:

Client2 (Vaibhav) side:

The clients are back in the pool and can connect again if they need to chat.

This is how our IM system works!!

We can also have multiple client sessions running in parallel. We can follow the same steps to run another session and check the client pool by running the command *list* on any Client's interface.

E.g., 1: A new client3, *Sahil*, sends a connection request to Client1, *Aishwarya*, and the server checks if the Client is busy or not. In this case, let's say, *Aishwarya* is already in an IM session with *Vaibhav*. The connection request will fail.

Client3 (Sahil) side:

```
Run: Server x Client x Client x Client x Client x
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Application
Enter your username:
Sahil
Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect)
3) For main menu (command -> menu)
2616779
list
List of Available Users:[Sahil, Aishwarya, Vaibhav, Shubham]

Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect client2_Name)
3) For main menu (command -> menu)
connect Aishwarya
Aishwarya is busy! Connection not established.
```

E.g., 2: Since the above connection request failed, the client 3, *Sahil*, now wants to connect to Client 4, *Shubham*. It sends a connection request to Client 4.

Client3 (Sahil) side:

```
Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect client2_Name)
3) For main menu (command -> menu)
connect Shubham
eeuxkWPm93ip9uSBW0jC7T62ac6ccgMK6foQa1F6gGwcgj+cS9Qtr0ShB3UjHJLjiJF0worfm2A2Snbtj1hM9TPcefiI/3tLpi4kLm
Hello Shubham!
Shubham: Hey Sahil, what's up?
```

Client3 (Shubham) side:

```
Run: Server x Client x Client x Client x Client x
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applicat
Enter your username:
Shubham
Please use below commands to do the following:
1) To List all clients connected to Server (command -> list)
2) To connect to a client (command -> connect)
3) For main menu (command -> menu)
3250491
f7NZ1QPsXfrakc+i1KWfeLDTTc496vJ80faEUUC725gMnL2y2VPL4iKZPn1AR+5TYJ6L21o2iq0wNeDahyvZuMH14Ejd
Sahil: Hello Shubham!
Hey Sahil, what's up?
```

Point to note: We currently have two pairs of Client sessions running in parallel!!

```
Please use below commands to do the following:  
1) To List all clients connected to Server (command -> list)  
2) To connect to a client (command -> connect)  
3) For main menu (command -> menu)  
2616779  
list  
List of Available Users:[Sahil, Aishwarya, Vaibhav, Shubham]
```

Results

This is how we have built our IM system. To conclude, overall security has been achieved through the following:

1. **Authentication:** We have used the public key cryptography and RSA mechanism to achieve authentication in the system. The private keys generated are only known to the Client the nonce sent by the Server is encrypted with the Client's private key and sent back to the Server, which can simply decrypt it using that Client's public key, thereby achieving successful authentication.
2. **Confidentiality:** The private keys generated are only known to the Client. The transmission of the nonce along with the encrypted private key will be safe for the Server to decode. The messages during client communication are also AES 128 but encrypted with a unique session key.
3. **Integrity:** Encryption of all the messages will be done using the secret session key known to the two clients using the AES algorithm, thus saving the integrity of the communication.
4. **Freshness:** For every new session between the clients, a new session key is generated. Since the session key is randomly generated by the Server, the channel is secure against replay attacks if a client logs out, the other Client is intimated about the expiry of the session with a message saying that the Client left the chat.

Team Contribution

1. Aishwarya Singh – AXS200109
 - Participated in the initial design discussion
 - Worked on client authentication using RSA
 - Tested the communication between client and server
 - Helped in creating the report

2. Sahil Prashant Kirpekar – SXK200032
 - Participated in the initial design discussion
 - Worked on creating a unique session key using AES
 - Tested the generation of a distinct session key for each new session between two clients.
 - Worked on the report
3. Shubham Singh – SXS200413
 - Contributed in initial design discussion
 - Created the public and private keys of each client
 - Worked on authentication and distribution of the session keys to the clients securely.
 - Tested the session key distribution between the clients and client authentication using nonce
 - Helped in the report
4. Vaibhav Tyagi – VXT200018
 - Contributed in initial design discussion
 - Created Client Server Communication
 - Worked on managing the client requests using client handler
 - Tested successful communication between two clients using encryption/decryption with session key.
 - Helped making the report