

CRYPTOGRAPHY GRADUATE PROJECT

SECURE CHAT ROOM APPLICATION

BY

**PRASHANT VEMULAPALLI
SRAVANTHI MALEPATI
SURYAVAMSI RAMANADHAM**

**Computer Science Department
Georgia State University, Atlanta GA**



ABSTRACT:

Chat applications have become such an integral and popular application on smartphone devices. These applications are used to send messages, images and files for users. The messages sent through this chat application must be secure and that is one of the most important aspects that must be worked on. The aim of this paper is to propose a secure chat application that provides end to end security, that leads to safe and secure exchange of private information between users. In our project we will implement a chat room system with authentication using certificates and message transfer using SSL (Secure Socket Layer). We use keytool (command line utility) to generate and manage keys and certificates, IAIK-JCE APIs to create and sign certificates programmatically with JSSE (Java Secure Socket Extension) to do secure networking.

KEYWORDS:

Secure chat application, Security, Secure session, Secure storage, Protocols,keystore.

INTRODUCTION:

A secure chat room service is a simple chat application. Mobile phones and other electrical devices through which people are able to communicate with each other have been growing rapidly over recent years. They have become such an integral part of our daily lives. Chat applications over recent years have made a major change in social media because of their unique features which attract a large amount of audience. It provides services like text messaging, video calls, images, files, real-time messaging, etc. Chat platforms are supported by cross platforms such as android and iOS.

There are a million users who use these services on a daily basis and to secure those conversations is one of the main challenges involved. These applications involve two types of architectures, mainly peer to peer and client-server networks. In peer to peer each user has his/her own data storage and it does not have any central server. However, for a client-server network, there is a central server and there are dedicated servers and clients involved.

Security and Privacy have become a major issue in terms of how important it is in chat applications but the majority of the people do not take it seriously enough. In a recent test done by EFF also known as Electronic Frontier Foundation, most of the popular messaging applications failed to meet most security standards. Many of these applications could be using these conversations for their own benefits hence there is some sort of privacy breach right there. Most applications only use TLS also known as Transport Layer Security for securing the channel. Usually, the service provider has access to every message exchanged with users in their infrastructure. This helps attackers more easily to gain access and use it against the users.

Therefore to maintain privacy and security messages should be encrypted from sender to receiver and most importantly not even the service providers will have that chance to read those messages.

There are many chat applications that we currently use that have security and privacy concerns. For example, Viber which is an instant messaging and voice over application for smartphones. Recently Viber supported the end to end encryption to their service, but only for one to one and group participations where users are using the latest version.

- In the Viber ios, attachments such as images and videos which are sent via the ios share extension do not support end to end encryption. Viber has other privacy issues such as adding a friend without his knowledge or adding him to groups without his permissions hence proving that there is a privacy threat in this application.
- Whatsapp, which one of the most popular social media application platforms, has recently enabled end to end encryption for its users all over. Developed by Open Whisper System WhatsApp uses part of a protocol, it provides a security verification code that can share with a contact to ensure that the conversation can be encrypted. Whatsapp is not an open source so it is very difficult to trust hence difficult to verify the functioning process.
- Facebook also another popular social media application, here regular chat does not provide end to end encryption only provides secure communication using TLS and it stores all messages on its servers.

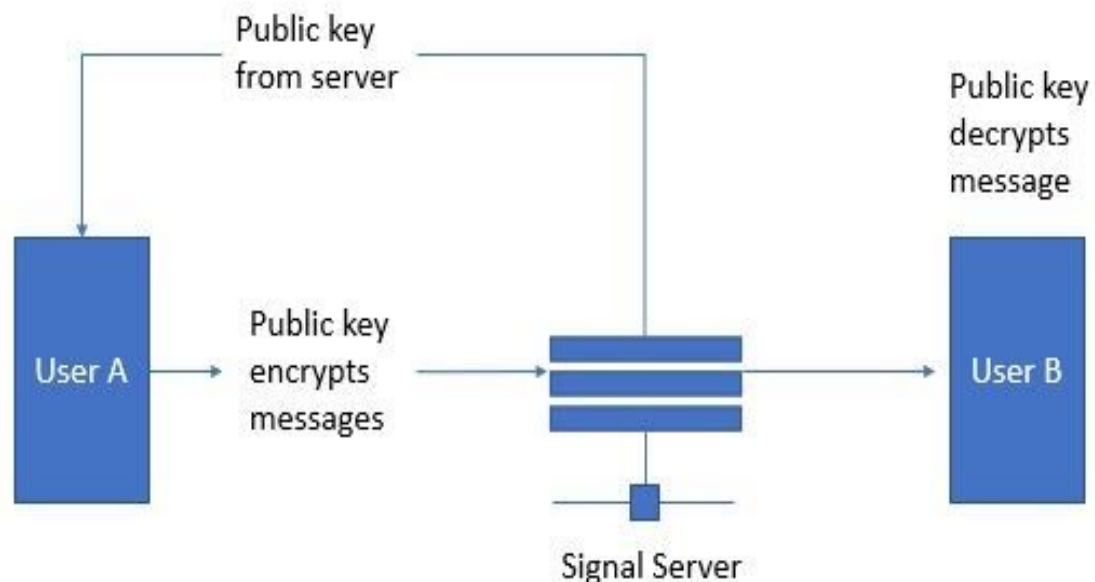


Figure 1: Basic chat application Architecture

PROBLEM STATEMENT:

Chat rooms became a preferred way to support a forum for 2-way conversation or discussion among a group of individuals with interest in a very common topic. Chat applications range from simple, text-based ones to entire virtual worlds. One major disadvantage of the most instant chat application is prone to security attacks. For instance, yahoo messenger is vulnerable to security attacks when instant messages are sent between a source and a destination machine. The reason is the fact that messages which are sent over the network are in a plaintext format with no encryption and decryption protection. During this project, we are implementing a straightforward and secured text-based chat client/server application. To develop an instant solution to enable users to seamlessly communicate with each other. The project should be very easy to use, hence enabling even a novice person to use it.

METHOD DESCRIPTION:

In this project, we will implement a simple client-server chat application. We will add authentication and privacy functionality to the implemented code. On the client-side, when a user sets up the application, the user either selects registration or log-in. On the server-side, the chat server consists of users' server and a message server.

The chat server is a Java application that can accept multiple incoming TCP connections. The chat server maintains a list of current chat rooms and chat clients can move between chat rooms. Messages sent by a chat client are broadcasted to all clients currently connected to the same chat room. This application uses SSL (Secure Sockets Layer) to protect the communication between client and server and utilize the username and password to authenticate members.

Our ultimate goal is clients should receive messages from other people with proper encryption and authentication. For instance, our chat room works in the following steps:

1. We have different chat rooms, among which clients choose the one which they want to enter at the time of connecting the chat server.
2. We have built an application where clients communicate one-to-one by establishing a session key. Here if one-to-one communication is preferred then it is free for everyone to use the chat server or clients can send messages directly to each other.

So we are providing authentication by using either passwords or certificates/signatures. Below are the different types of entities in the chat system:

Protocols

In our application we use protocols to prevent attacks like dictionary attacks etc. we assume that public keys of the servers are present at the client-side. So we deploy MAC in the messaging protocol. Depending on our authentication, the designed protocols are as follows:

Certificates/signatures Authentication:

1. A registration protocol between a client and the certificate authority(CA), where the client obtains its certificate.
2. A login protocol between a client and the chat server.
3. A messaging protocol between a client and the chat server.

Password authentication:

1. An authentication protocol between the client and the authentication server, where the client authenticates itself.
2. A login protocol between client and ticket-granting server.
3. A messaging protocol between client and chat server.

For secure communication we use network sockets. The JCE provides an abstraction for secure sockets in the `java.net.ssl` package and this relieves us from explicitly performing the key exchange, encryption, and integrity of the messages transferred over these sockets.

- **Access control for clients**

Every client who joins the Chat Server can be in one of the two chat rooms that the server supports. On the server-side, these rooms are just logically separate lists of clients in each room, so that it does not interchange the messages from clients in different rooms. A client can be only in one of the two rooms. The access privileges for a client are encoded in the certificate it presents to the server during the SSL connection setup.

- **Setup of the system**

The chat system now consists of three types of entities: chat clients, the chat server and the certificate authority (`CertificateAuthority` class). The Certificate Authority is an online entity that has an encrypted file containing the usernames and passwords of the expected chat clients. Passwords are now stored after salting and hashing them, and verified in a similar manner. Along with the username and password for each client, the Certificate Authority also stores the access permissions for that client. To generate this encrypted file of usernames, passwords, and permissions you can modify and re-use the file encrypter code.

When a client starts up it first connects (through SSL) to the Certificate Authority. Note that at this point the client utilizes no certificate for making the SSL connection. This means that the SSL connection provides only one-way authentication of the Certificate Authority (CA) to the client. Now the client transmits its username, password and public key to the CA. The CA verifies the username and password (after salting and hashing appropriately). If this verification

is successful, it generates a certificate, by signing the public key of the client with the CA's private key. This certificate contains the access permissions for the client as well.

The client uses the newly issued certificate from the CA, to connect to the ChatServer using SSL. Note that this time around the SSL connection will provide authentication in both directions and hence, you will not need any password-based authentication between the client and the server. Based on the access permissions in the certificate presented by the client, the server will add it to the appropriate Chat Room.

IMPLEMENTATION:

The following steps are required to perform the implementation:

1) **Registration:** Since there is no database where the users information is being kept, all the users must register using the Certification Authority. The username of the user and the public key of the RSA of the user is sent to the Certification Authority. Then the Certification Authority checks if the user is registered. If he or she is then well good , and if not then the new certificate will be signed by the Certification Authority for the users public key and return it to the user and store the details in its keystore to remember that in fact the user did register.



Figure 2: Registration

2) **Client-Server key exchange and mutual authentication:** To perform encryption and mutual authentication we use Diffie-Hellman key exchange. When authenticating the user's identity who wants to chat the server uses certificates and digital signatures signed by the Certification Authority, and the same goes for the user to verify the identity of the server. Thus during the key exchange the user completes the login process to verify his or her certificate. The DH key pair is created by the server and sends the user the DH public key which is signed by its own certificates, DH parameters and RSA private key, all of this while the client is connected to the server. The client then checks that the server is signed by the Certificate Authority. By encrypting his/her DH public key with the RSA public key of the server the user sends his

certificate to the server. The user's certificate is signed by the certificate authority, verified by the server in the direction of these packets from the client, and the signed data is correct signature using the user's RSA public key. Then the DH public key is decrypted by the server with the RSA private key.

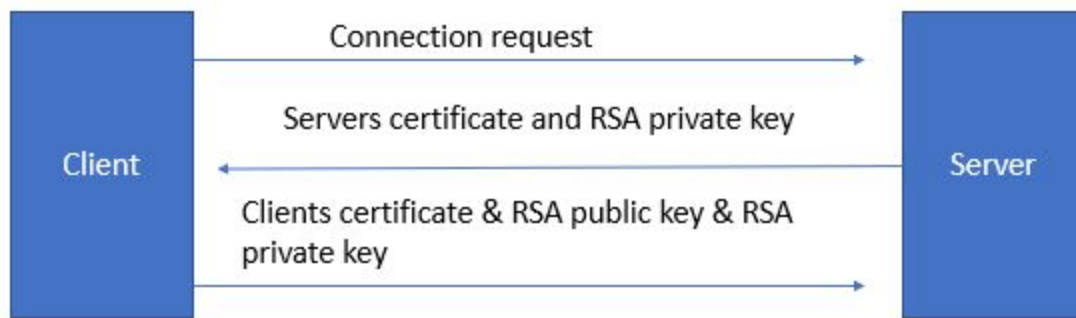


Figure 3: client server key exchange and mutual authentication

3) **Joining a room:** When the user first enters the room, the keys are produced by the room keys. The connection between the server and the client is encrypted using the session key which is decided during the exchange of keys between the server and the user.

Using the encrypted connection, the user sends the id of the room which he wants to join and the server joins the user to the requested room and sends the room's shared key to the user using the same encrypted connection.

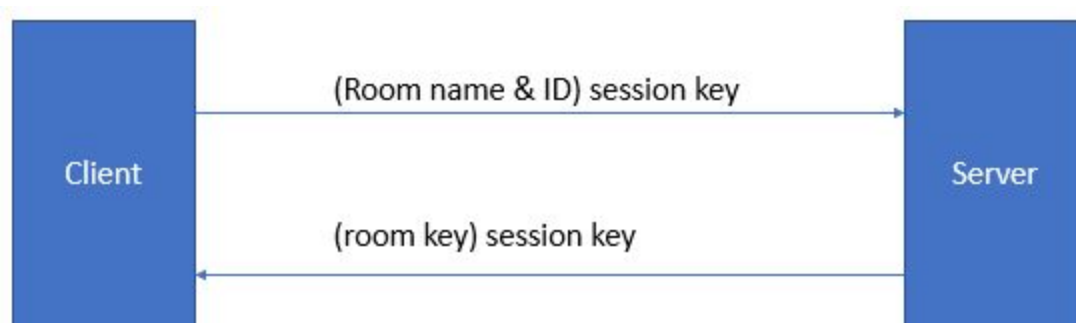


Figure 4: Room information

4) **Messaging:** The user continuously sends messages to and receives messages from the server. The messages are encrypted and decrypted using the room key which is obtained when joining the room. A MAC is added to each message which is SHA256 hash encrypted with the room key of the message. The server sends the encrypted message and MAC, which it receives from a client to all clients in the same room.

There are two types of room, one is public and the other is private. Comparatively many users can access the public room but only two people can join a private room and discards the subsequent requests.

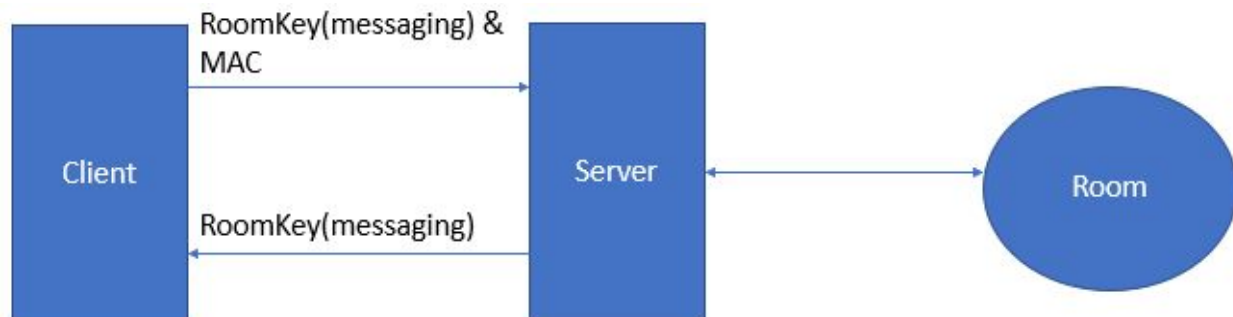


Figure 5: Messaging Architecture

RESULTS AND DISCUSSION:

CertificateAuthority.java file starts up the Certificate Authority where trusted certificate authority is denoted by CA. Certification Authority has a self-signed certificate. Running CA:CertificateAuthority file results in running up our project and the below screenshot is the reference for CertificateAuthority file. As long as the communication happens between client and server CertificateAuthority file should be in running state at the background.

The screenshot shows the Eclipse IDE with the `CertificateAuthority.java` file open. The code defines a `CertificateAuthority` class with methods for starting up, getting port number, getting output area, and a main method. The console output at the bottom shows the program running successfully.

```

124 public int startup(String _ksFileName,
125                   char[] _privateKeyPass,
126                   int _caPort) throws IOException, KeyStoreException, CertificateException, NoSuchAlgorithmException, UnrecoverableKeyException {
127     portNumber = _caPort;
128     keystoreFileName = _ksFileName;
129     privateKeyPassword = _privateKeyPass;
130
131     FileInputStream keyStoreStream = new FileInputStream( "/Users/sravanthimalepati/Downloads/Secure-Chat-Room-Service-master/ChatRoom-Service/Chat/keystores/KeySt
132     keyStore = KeyStore.getInstance( KeyStore.getDefaultType() );
133     keyStore.load( keyStoreStream, privateKeyPassword );
134
135     PrivateKey privateKey = (PrivateKey) keyStore.getKey( "ca", privateKeyPassword );
136     Certificate certificate = keyStore.getCertificate( "ca" );
137     KeyPair keyPair = new KeyPair( certificate.getPublicKey(), privateKey );
138
139     _layout.show( _appFrame.getContentPane(), "ActivityPanel" );
140
141     CertificateAuthorityThread _thread = new CertificateAuthorityThread( this );
142     _thread.start();
143     return CertificateAuthority.SUCCESS;
144 }
145
146 public int getPortNumber() {
147     return portNumber;
148 }
149
150 public JTextArea getOutputArea() {
151     return _activityPanel.getOutputArea();
152 }
153
154 public static void main(String[] args) {
155     CertificateAuthority ca = new CertificateAuthority();
156     ca.run();
157 }
  
```

Console Output:

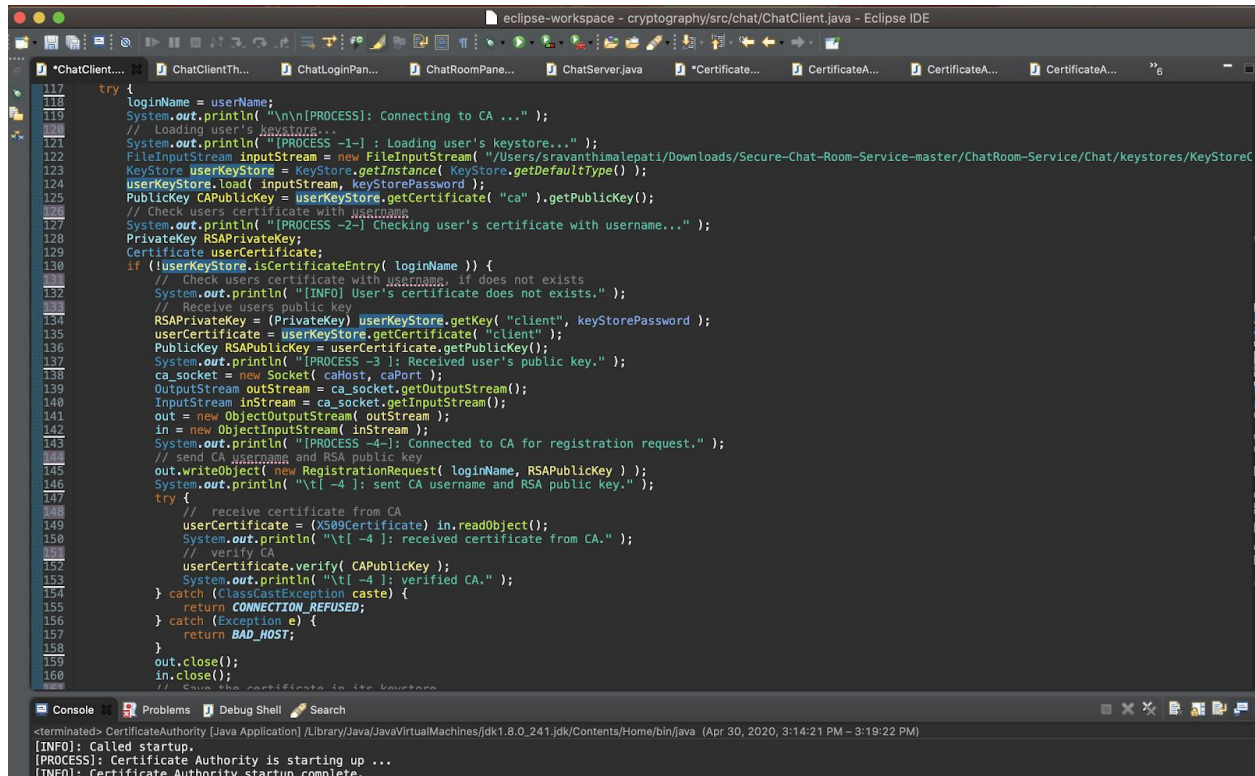
```

<terminated>: CertificateAuthority [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home/bin/java (Apr 30, 2020, 3:14:21 PM ~ 3:19:22 PM)
[INFO]: Called startup.
[PROCESS]: Certificate Authority is starting up ...
  
```


Figure 6: Code for Certificate Authority

CertificateAuthorityThread.java accepts connections from the clients, verifies their username-passwords, and issues certificates to them whereas **CertificateAuthorityActivity.java** displays the activity of the client.

ChatClient.java class for the client entity and chat clients are denoted by C1, C2, ...Cn. Obtains chat certificate from the CA and uses it to connect to the Chat Server. We also have **ChatClientThread.java**: Receive posted messages from the server.



```
117 try {
118     loginName = userName;
119     System.out.println("\n\n[PROCESS]: Connecting to CA ...");
120     // Loading user's keystore...
121     System.out.println("[PROCESS -1-]: Loading user's keystore...");
122     FileInputStream inputStream = new FileInputStream("/Users/sravanthimalepati/Downloads/Secure-Chat-Room-Service-master/ChatRoom-Service/Chat/keystores/KeyStoreC");
123     KeyStore userKeyStore = KeyStore.getInstance("KeyStore.getDefaultType()");
124     userKeyStore.load(inputStream, keyStorePassword);
125     PublicKey CAPublicKey = userKeyStore.getCertificate("ca").getPublicKey();
126     // Check users certificate with user's name
127     System.out.println("[PROCESS -2-]: Checking user's certificate with username...");
128     PrivateKey RSAPrivateKey;
129     Certificate userCertificate;
130     if (!userKeyStore.isCertificateEntry(loginName)) {
131         // Check users certificate with user's name, if does not exists
132         System.out.println("[INFO] User's certificate does not exists.");
133         // Receive users public key
134         RSAPrivateKey = (PrivateKey) userKeyStore.getKey("client", keyStorePassword);
135         userCertificate = userKeyStore.getCertificate("client");
136         PublicKey RSAPublicKey = userCertificate.getPublicKey();
137         System.out.println("[PROCESS -3-]: Received user's public key.");
138         ca_socket = new Socket(caHost, caPort);
139         OutputStream outputStream = ca_socket.getOutputStream();
140         InputStream inputStream = ca_socket.getInputStream();
141         out = new ObjectOutputStream(outStream);
142         in = new ObjectInputStream(inputStream);
143         System.out.println("[PROCESS -4-]: Connected to CA for registration request.");
144         // send CA username and RSA public key
145         out.writeObject(new RegistrationRequest(loginName, RSAPublicKey));
146         System.out.println("\t[-4-]: sent CA username and RSA public key.");
147         try {
148             // receive certificate from CA
149             userCertificate = (X509Certificate) in.readObject();
150             System.out.println("\t[-4-]: received certificate from CA.");
151             // verify CA
152             userCertificate.verify(CAPublicKey);
153             System.out.println("\t[-4-]: verified CA.");
154         } catch (ClassCastException caste) {
155             return CONNECTION_REFUSED;
156         } catch (Exception e) {
157             return BAD_HOST;
158         }
159         out.close();
160         in.close();
161         // Save the certificate in its keystore
```

Figure 7: Code for Client's Certificate Authority

ChatServer.java: Chat server is denoted by S and ChatServer accepts chat requests (via secure connection) from clients.

ChatServerThread.java: Receive messages from the clients and post messages to the appropriate chat room.

```
eclipse-workspace - cryptography/src/chat/ChatServer.java - Eclipse IDE

33 this.port = port;
34 CLIENT = -1;
35 clients = new HashMap<>();
36 roomKeys = new HashMap<>();
37 serverSocket = null;
38 InetAddress inetAddress = InetAddress.getByName( null );
39 hostname = inetAddress.getHostName();
40 FileInputStream inputStream = new FileInputStream("/Users/sravanthimalepati/Downloads/Secure-Chat-Room-Service-master/ChatRoom-Service/Chat/keystores/KeySt
41 KeyStore serverKeyStore = KeyStore.getInstance( KeyStore.getDefaultType() );
42 char[] serverKeyStorePassword = "123456".toCharArray();
43 serverKeyStore.load( inputStream, serverKeyStorePassword );
44 CAPublicKey = serverKeyStore.getCertificate( "ca" ).getPublicKey();
45 // Getting Servers certificate and RSA Private key from Servers KEYSTORE
46 certificate = serverKeyStore.getCertificate( "server" );
47 char[] SERVER_KEY_PASSWORD = "123456".toCharArray();
48 RSAPrivateKey = (PrivateKey) serverKeyStore.getKey( "server", SERVER_KEY_PASSWORD );
49 // Generating room key
50 roomKeyGenerator = KeyGenerator.getInstance( "AES" );
51 roomKeyGenerator.init( 128 );
52 } catch (UnknownHostException e) {
53     hostname = "0.0.0.0";
54 } catch (IOException | CertificateException | UnrecoverableKeyException
55         | NoSuchAlgorithmException | KeyStoreException e) {
56     e.printStackTrace();
57 }
58 }
59 public static void main(String args[]) {
60     try {
61         int port = Integer.parseInt( "10500" );
62         ChatServer server = new ChatServer( port );
63         server.run();
64     } catch (NumberFormatException e) {
65         System.out.println( "[ERROR]: Usage: java ChatServer host portNum" );
66         e.printStackTrace();
67     } catch (Exception e) {
68         System.out.println( "[ERROR]: ChatServer error: " + e.getMessage() );
69         e.printStackTrace();
70     }
71 }
72 }
73 }
74 }
75 }
76 }
```

Figure 8: Code for Server receiving the secure connection

DEMO SCREENSHOTS:



Figure 8: Output of CertificateAuthority

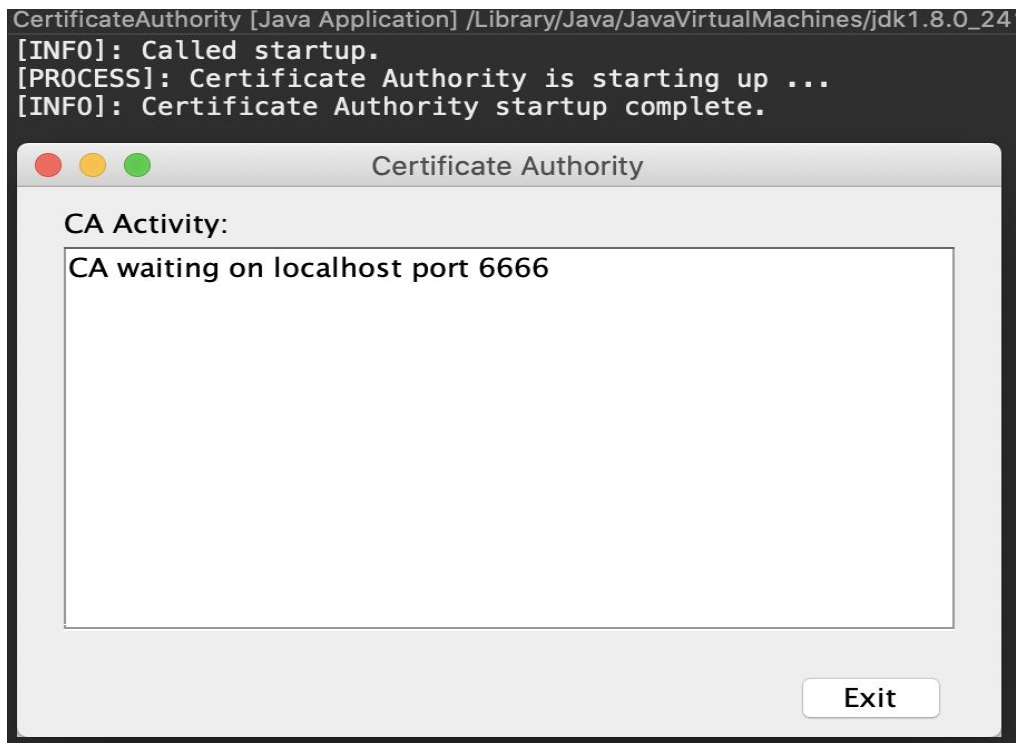


Figure 9: Server is waiting

The image shows a "Chat Room" window with a title bar containing red, yellow, and green buttons. The window displays a "Welcome to Chat" message. Below the message are several labeled input fields: "Username:" with the value "sravanthi", "Room Name:" with "roomA", "Room Type:" with "public", "KeyStore File Name:" with "/Users/sravanthimalep", "KeyStore Password:" with "*****", "Server Host Name:" with "localhost", "Server Port:" with "10500", "CA Host Name:" with "localhost", and "CA Port:" with "6666". The "CA Port" field is currently selected. At the bottom center is a "Connect" button.

Figure 10: Login Page for Chat Room

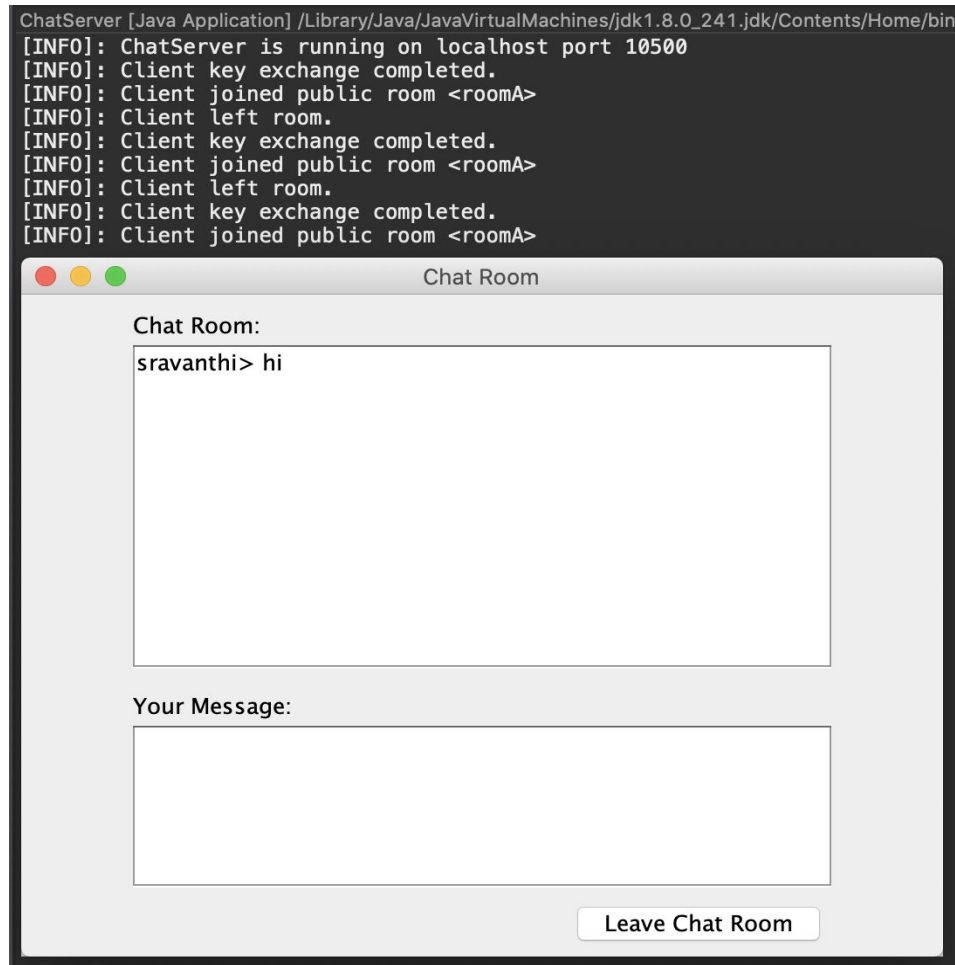


Figure 11: Client communication in Chat Room

CONCLUSION:

For any chat application there is always a drawback and that drawback will lead to the attackers finding a way to breach into the application hence there is always a risk for a secure communication. This issue should be taken into serious consideration as any attacker can toggle with a user's data and use it against the user in all sorts of ways. Based on the performance evaluation we can see that there is a secure end to end communication between the users. There is a lot of scope to further enhance the security features in this chat application using different methods.

REFERENCES:

1. Ash Read, "How Messaging Apps Are Changing Social Media," 2016. [Online]. Available: <https://blog.bufferapp.com/messaging-apps>.
2. Most popular messaging apps 2017 | Statista," 2017. [Online]. Available: <https://www.statista.com/statistics/258749/most-popularglobal-mobile-messenger-apps/>.
3. D. Moltchanov, "Client/server and peer-to-peer models: basic concepts," 2013.
4. ViberEncryption Overview." [Online]. Available: <https://www.viber.com/security-overview/>.
5. WhatsApp inc, "WhatsApp security whitepaper," p. 10, 2017.
6. T. Susanka, "Security Analysis of the Telegram IM," p. 70, 2016
7. Keytool: <http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html>
8. Java.security.cert:
<http://docs.oracle.com/javase/6/docs/api/java/security/cert/package-summary.html>
9. Sun Tutorial on Socket Programming:
<http://java.sun.com/docs/books/tutorial/networking/sockets/>
10. Sun Tutorial on Thread Programming:
<http://java.sun.com/docs/books/tutorial/essential/threads/>
11. T. Whitepaper, "Messenger Secret Conversations," 2016.
12. B. O. B. Kamwendo, "Vulnerabilities of signaling system number 7 (ss7) to cyber attacks and how to mitigate against these vulnerabilities. bob kamwendo," vol. 7, no. 7, 2015.