Design Document

Secure Private Group Chat

Anthony Arseneau

Computer Networks Final Project

COMP 4911, April 1, 2024

**Preface**

The following document contains justifications and details of the development process for the Secure Private Group Chat networks final project. No external sources were consulted during the development process beyond those included in this document.

**Design Process**

Step 1

The initial step to create the Secure Private Group Chat was to make a basic client-server group chat application. Here, there were only 3 Java classes named ChatServer, ChatClientHandler, and ChatClient. ChatServer and ChatClientHandler were running on the server side of the application. The role of ChatServer was to listen on port 12000 for any new client connections and create a ChatClientHandler for each of them. Anytime a client wanted to send a message to the chat, the message would be sent to the server to its corresponding ChatClientHandler and would be broadcasted to the rest of the connected clients. Initially, the chat was simply used through the terminal. This user interface was not user friendly.

Step 2

The second step was to create a Java Swing window (AuthenticatorView) where, when staring the application on the client side, it would prompt the user for the username and password. Here, there was no authentication system implemented yet, but the given username was used for showing the user’s name when sending a message. At the same time, a chat view window (ChatView) was also introduced using Java Swing to show the chat contents. The window had a large area to show the messages sent and received, a text field at the bottom to input messages, and a send button to the bottom right. When pressing the button, the message in the text field would disappear and be sent to the server where it would broadcast it.

Step 3

The third step was to implement an authentication system. This was achieved by creating a whitelist.txt file where it would keep all authorized users’ hashed usernames and passwords. The hashing function used was SHA-256 to securely hide users’ information in case of a breach in the server. The credentials for each user were set to be on one line each where the usernames and passwords has a single whitespace separating the two values. Then, before the server granted access to the group chat to the new connection, the first message sent by the user to the server was its credentials. With that information, the server, using a new class named Whitelist implementing verifications, could then reply with either “Yes” or “No” whether the credentials matched with an entry in the whitelist.txt. If the answer was “No”, then the user and server would close connections, and if the answer was “Yes”, then the server would grant the access to the group chat to the user.

Step 4

The fourth step was to implement cryptographic exchange of information. This step was easier said than done. There was the creation of two classes named RSACipher and AESCipher where each implemented the roles of their respective ciphers. The RSA cipher is great to send and receive secure messages when no symmetric key ciphers are established between two parties. Just like in this case, the clients and the server are theoretically far apart with no prior symmetric key exchange. Unfortunately, the RSA cipher encryptions and decryptions are slow and have a limited message length. That is why it was used to only send the first message of the client to the server, which is the credentials, and the response from the server to the client, which includes, if granted access to the group chat, the secret asymmetric key of the AES cipher used for the rest of communications. The AES cipher, compared to the RSA cipher, is much faster to encrypt and decrypt, and is able work with longer messages.

**Client Design**

AuthenticatorView Class

|  |  |
| --- | --- |
| A screenshot of a login form  Description automatically generated  Figure 1 Authenticator GUI with empty fields. | A screenshot of a phone  Description automatically generated  Figure 2 Authenticator GUI with filled fields. Pressing "Submit" will store the values in variables to eventually be sent to the server. A message appears indicating that things are happening in case of a delay. |