SENG 360 A3

Chat Application

Jake Runzer V00797175
Jason Sanche V00349530
Zhengtang (Raymond) Wang V00802086

TECHNICAL DETAILS

The simple-chat application was created in java using the java.net.Socket library with configurable security settings between two parties, in this case ‘server’ and ‘client’ who interact within with a simple JText GUI. Both the client and server live in the same project. We use Java packages to separate the code between client, server, and common. Common is the package where most of the logic is. Client and server packages contain code specific to each program.

When the client and server first connect, they will perform a simple handshake protocol. During this protocol, they will compare chat settings. If these settings are not identical (confidentiality, integrity, and authentication) both sides will disconnect.

We have assume that the server is always up and running. Therefore, it must be running before the client is started. If a client disconnects from the server, the server may remain up and connect to another client.

The port 8888 is used for communication and is required to be open. The code lives in a Github repo, https://github.com/dyadem/simple-chat. However, it is also included in this assignment package.

Security Settings

1. Confidentiality

The *Confidentiality* setting enables the message to be encrypted before being sent. If this setting is selected, during the connection protocol, the client and server will perform a Diffie Hellman key exchange. This exchange will generate a shared secret key accessible to both the client and server. Both parties will use this key to encrypt each message before it is sent using AES encryption in CBC mode. When the message is received, it is first decrypted before being displayed to the user.

The encrypted text being sent is printed to the console (not the GUI).

2. Integrity

The *Integrity* setting enables the message signature and verification. In order to make sure the message is send by the authenticated user, and not modified by an attacker in the middle. The program will generate DSA public and private keys after the client and server are connected and if the *Integrity* setting is selected. Before the message is sent, it will be signed using the sender's private key. The signature of the message is sent alongside the message itself. When a message is received, it will go through the

verification process where the signature is verified using the sender's public key. If the verification is not successful, an error will be displayed to the user.

The signature of the message being sent is printed to the console (not the GUI).

3. Authentication

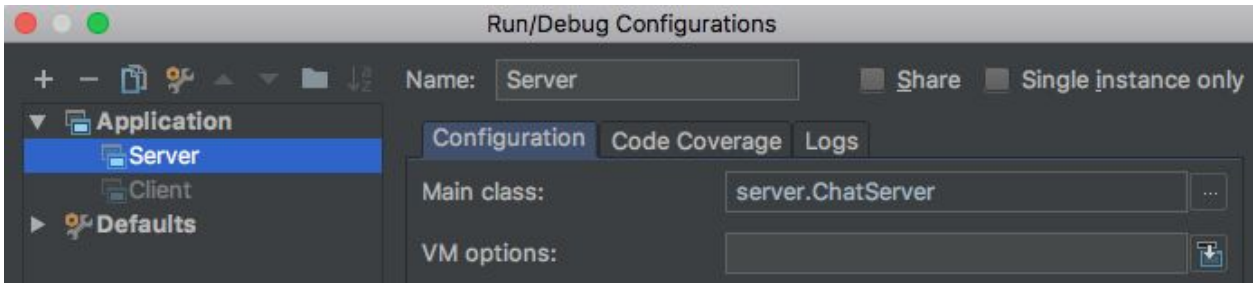
The *Authentication* setting enables mutual password authentication. The system prompts the user for their password. Passwords are hashed with the SHA-512 protocol and salted with a locally defined SALT variable. Passwords are verified in the userLogin function (src/commom/Auth) which takes the name and password, hashes with the generateHash (src/commom/Auth) function and checks against the user's stored password. If they match the user is authenticated.

In this simplified case there are only 2 users (server and client) and their respective password hashes are stored in text files within the project files. In a real-world case, the user would set their password during their user account setup which would be hashed and stored in their respective database entry.

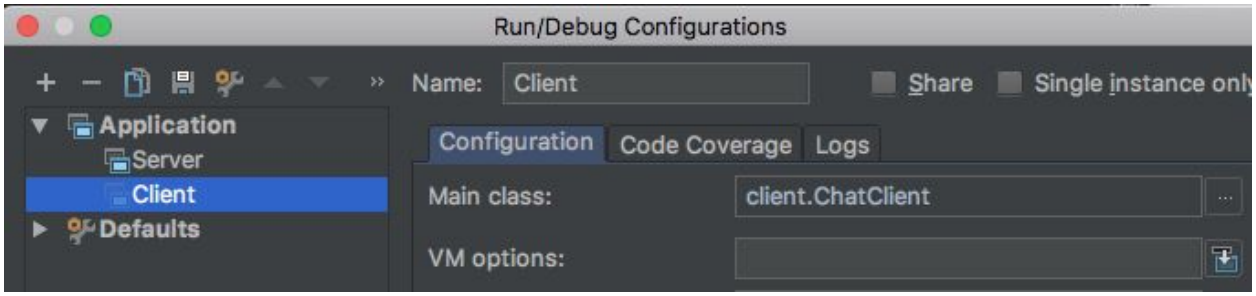
HOW TO COMPILE THE PROGRAM

The program was developed using the IntelliJ IDE (https://www.jetbrains.com/idea/). This project may be imported into the IDE to compile both the client and server programs. We use run configurations to compile both the server and client in the same project.

The server run configuration has the server.ChatServer as the main class.



The client run configuration has the client.ChatClient as the main class



The .jar files in the /jars folder can be run as a standalone application using Jar Launcher for mac 15.01.

HOW TO USE THE PROGRAM

Both programs can be started using the command line.

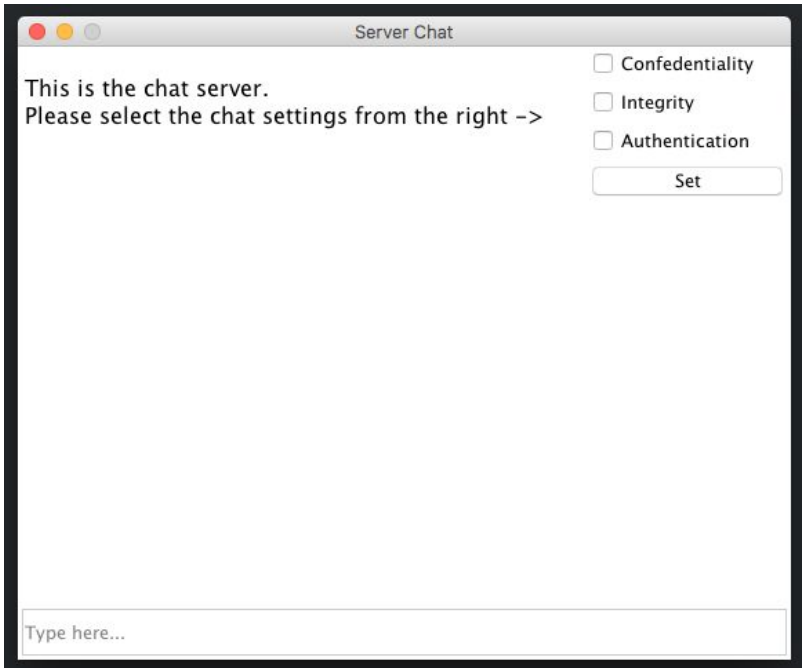
Server

The server must be started first.

Navigate to the “jars” directory and run the chat-server.jar file

```
cd jars
java -jar chat-server.jar
```

You should see the following window



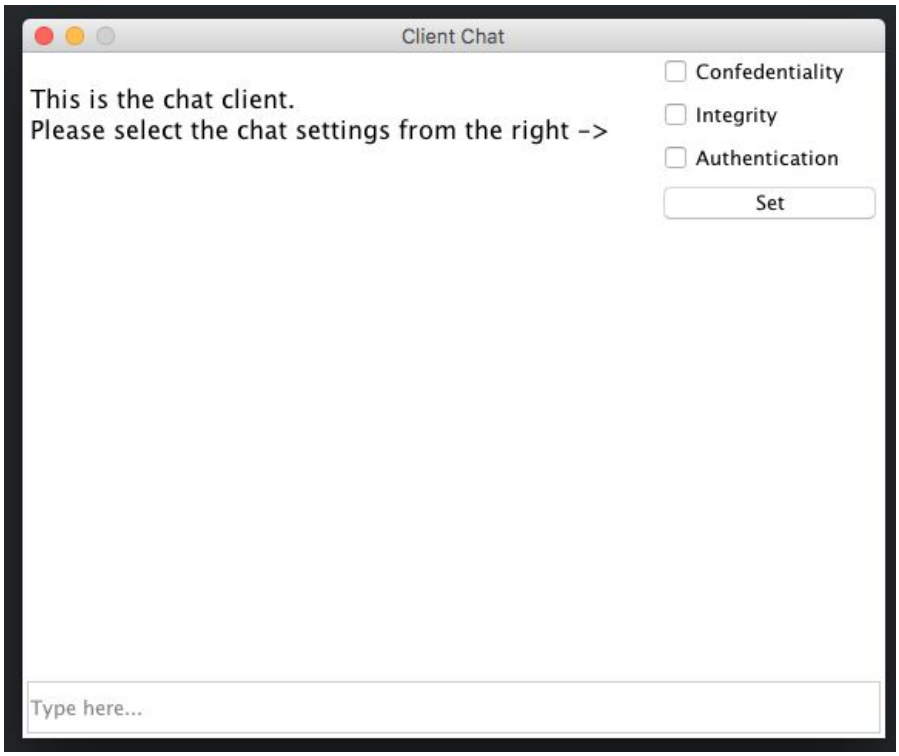
Client

The server must be already running.

Navigate to the “jars” directory and run the chat-client.jar file

```
cd jars
java -jar chat-client.jar
```

You should see the following window

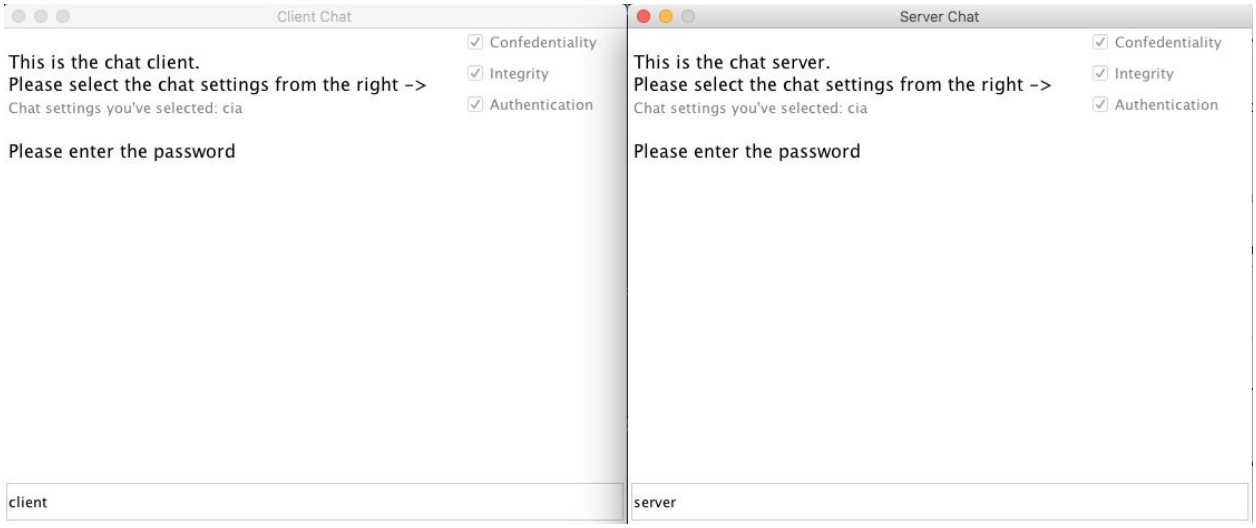


From this point on, both client and server behave very similarly. The only difference being that the server will remain open and waiting to accept connections after the client has disconnected. There are also some minor copy changes between the windows.

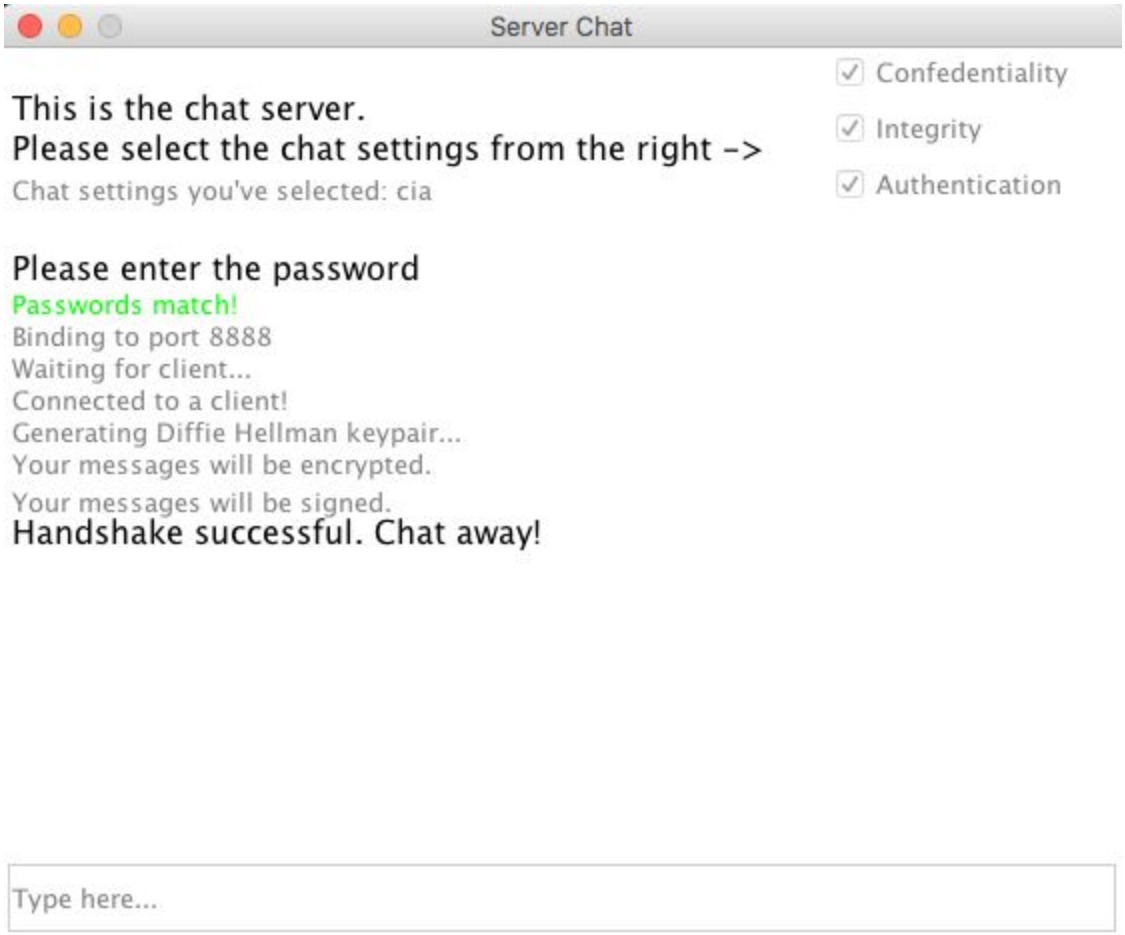
Selecting Settings

On both client and server, the chat settings need to be selected. Any combination of the settings can be chosen, but they need to be the same otherwise the handshake fails and both client and server disconnect. This guide will walk through using all three chat settings.

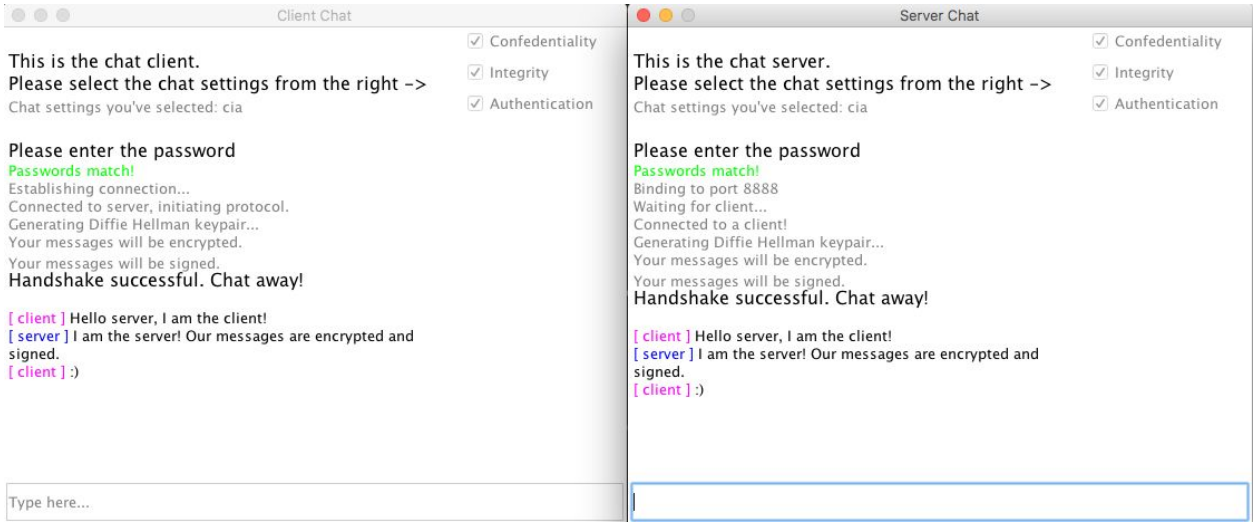
If the *Authentication* setting is selected, you will need to enter a password.



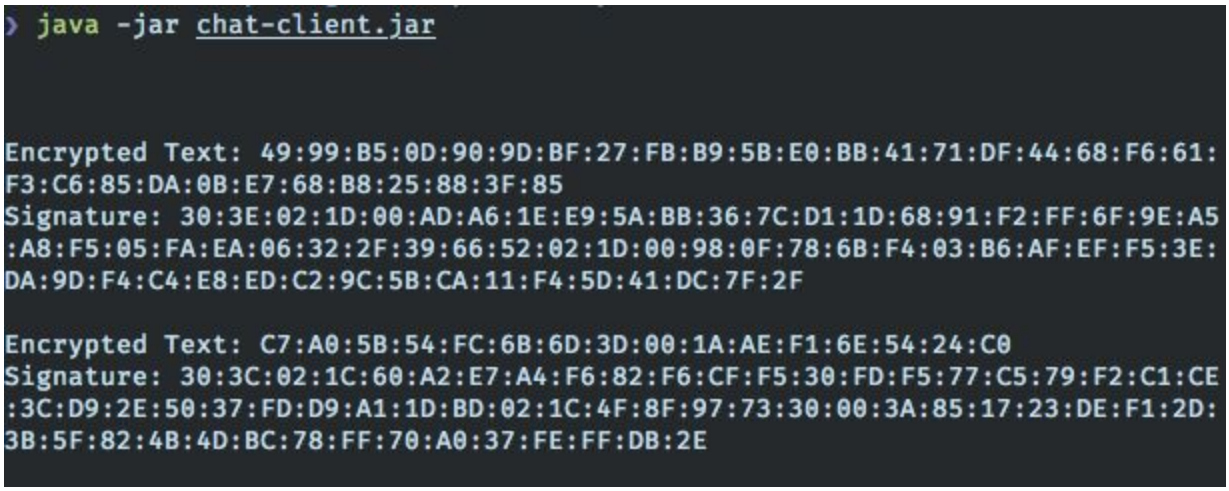
The password for the Client program is “client” and the password for the Server program is “server”. After authentication you should see a window similar to this.



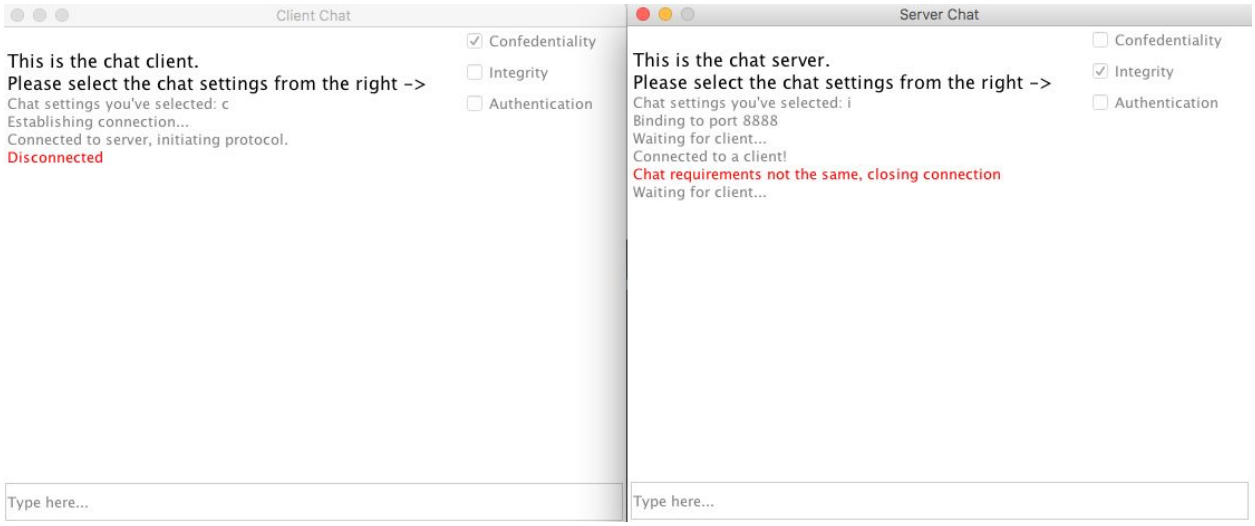
The client and server have successfully connected. You can now chat using the text box at the bottom.



In the terminal you should be able to see the hex value of the signature and encrypted text being sent (note: you will not see anything if confidentiality or integrity is not selected).



If the chat settings differ for the client and server, you will see the following window.



Code Structure

The code is organized into the following package structure

- client
- server
- common

The client and server packages containing code specific to the client and server programs respectively. The common package is where most of the login belongs.

In the common package there are the following files

- Auth.java
    - The majority of the code needed for this assignment. This class has static methods which deal with user authentication, encryption/decryption, hash generation, and signing/verifying signatures
- Chat.java
    - The majority of the logic for connection, sending, and receiving chat messages
- ChatProtocol.java
    - Code which contains a state machine for the initial client <-> server connection.
- ChatSettings.java
    - Class which holds the chat settings
- DiffieHellman.java
    - Static methods for generating a shared secret key using the diffie hellman key exchange

- Mesasge.java
    - The message object that is serialized and sent over the socket
- Utils.java
    - Utility static methods, such as converting a byte array to a hex string
- ChatWindow.java
    - The code for the chat window itself. Manages sending showing new messages and statuses.
- ChatWindow.form
    - The IntelliJ GUI form file. Used to build the GUI.