Chat Application Project
TkTalk

By Saukun Thika You and Truc Nguyen

COMP3825: Network Information Assurance

4/27/2025

Introduction

This report details the implementation of a client/server reliable chat application in the Python programming language. This project aimed to create a secure messaging application hosted by a server to send reliable messages to multiple clients on an easy user interface with socket programming, SSL/TLS encryption, Tkinter library, and threading.

Design

We chose to use Python for its incorporation of many libraries for socket programming, threading, and the Tkinter GUI. We have two files for this application: server.py and client.py.

Server Side

In the server.py file, we implement our server. We have 7 primary functions we use: handle_client(), handle_private_message(), broadcast(), send_active_users(), remove_client(), stop_server(), and start_server(). At the top of our code, we import the libraries socket, threading, signal, sys, and ssl. Then we have our host and port that we will use for connection. We also included dictionaries to hold client and username and our SSL server self signed certification and server private key. The server code is structured into 7 functions that are called when the server starts.

- Within the **handle_client** function, we listen for incoming messages and check the message if it starts with "/@", "/u", and ".exit" and route them to the corresponding functions. We use the client_socket that our server receives to decode the message. If there is any error, we remove the client's socket. We did this by using the while loop set to True.

- In the **handle_private_message**, we parse the message into 3 chunks to check for the first part to equal "/@", then find the second chunk in the username dictionary. If found, send the message, if not give a rejection.
- In **broadcast** function, it checks if the client_socket is equal to the sender_socket to ensure that the message is not broadcasted to the sender. Then it sends the message to all users except the sender. There is error and exception handling in case the client doesn't exit.
- In **send_active_users** function, it sends the client socket a list of all active users by sending the usernames in the username dictionary.
- In **remove_client** function, it removes the client's username from the username dictionary and the client from client dictionary. Then, call the broadcast function to announce that the user has left and closes the client's socket.
- In **stop_server** function, the server is shut down by receiving the Control + C signal which was implemented by importing signal library, setting the signal, importing the sys library, and doing sys.exit to close it down.
- In **start_server** function, it calls the socket library to create the server's socket, bind the host and port to the server's socket, listen for connections, use the SSL library to wrap the socket with encryption, shutdown the server by calling stop_server if signal is received, and accepting client's connection. Then threading library is used to create new threads on every new client that runs the handle_client function.

These seven functions was developed and aimed to be a functional and secure server.

Client Side

In the client.py file, we have the implementation of a secure connection of the client socket to the server with Tkinter GUI. The technologies used are Python3 with libraries such as **socket** for TCP connection, **ssl** for secure socket wrap, **threading** for handling incoming messages, and **tkinter** for building user interface. We have 5 functions such as connect(), receive_messages(), append_message(), send_message(), and on_close().

- **connect()** is our main function with the purpose of creating and setting up a secure connection to the server. We first create a client socket then wrap it into SSL then connect to the server using the server's IP address and Port Number. The first message received from the server is a prompt for username, where we automatically append a random 4-digit number to avoid duplication for users with the same name. The next message received from the server is a welcome message with a set of instructions.
- For the **receive_messages()**, it uses threading to listen for incoming messages from the server or messages from other users and updates the chat box in real time. If the message that is received starts with the client's username, it will not be shown to prevent repeating your own message again twice. The receive thread helps ensure GUI does not freeze while waiting for messages.
- **send_message()** sends the messages that are typed by the users within the chat box to the server. If the message is flagged as a private message (starts with /@), it won't be sent again to avoid redundancy. If the message starts with ".exit", we will stop the receiving thread and shut down the GUI window.
- **append_message()** is a helper function that adds messages into the chat window. It enables the insertion of new messages, allowing users to type, and automatically scrolls

to the bottom to show the most recent message. This function is being used in other multiple main functions to help display messages.

- **on_close()** is a function that handles when the user manually closes the window to prevent crashes or not properly closing client sockets. When the user clicks on the "X" button, it will stop all background receiving thread and try to signal the server for a disconnection by sending a ".exit" as well as closing the client socket. Finally, the GUI is destroyed safely.
- The client side also includes GUI codes.

GUI

For the user experience, we implemented a Graphical User Interface with a built-in Python library called **Tkinter.** We set up by starting off with creating a chat window, then added a scrollable chat display. There is an input box for message entry. When user first open up the application, a prompt for username input will pop up. Lastly, we binded some keys to ease user's interaction such as "Enter" key to send messages, and "X" button to shut down the app.
The GUI code itself is independent from the server, however it does rely on information coming from or going to the server during runtime in order to send or receive real chat data.

Bonus features

Feature 1: Threading

We used threading to implement chatting between multiple clients. Threading refers to the ability of a program to execute multiple parts of its code independently and simultaneously, enabling parallelism and multitasking. Threading allows the server to handle multiple clients at once. On the server side, a new thread is created for each client connection that runs handle_client() function in parallel, so that each client can send and receive messages without having to worry about the server waiting. On the client side, threading is used to listen for incoming messages and ensure GUI does not freeze while waiting for messages.

Feature 2: SSL/TLS encryption

We used SSL/TLS library to encrypt our data before being transmitted over the internet, making it unreadable to anyone who might intercept it. We implement this by verifying the identity of the server and use a configuration file, san.cnf, to generate the public certificate and private key. The public certificate contains public key and information about the server, and is used by the client to verify the server's identity and encrypt data. The private key stays secret on the server and is used to decrypt data sent by the client side. We set up SSL context and wrap both the client and server socket in SSL.

Evaluation Results:

The chat application was tested for functionality, reliability, error handling, and security.

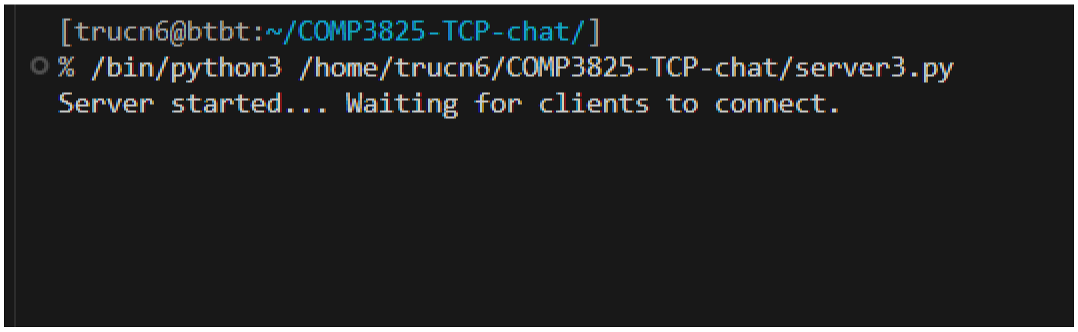
- Functionality was the first to be checked. Every function runs properly with no errors, and if errors are detected, they are handled with. We ensured that clients were able to establish and maintain a stable connection with the server. Public and private messages are sent successfully and received without any errors or missing fragments. The application handles client disconnection gracefully without impacting the server or other clients.
- Reliability is the second to be checked. Messages were delivered without loss, duplication, or out of order arrival. The server could handle multiple clients sending messages without crashing or threading confusion between 5 clients.

- Error handling ensured that every edge case and unexpected errors would not crash the program but instead handled gracefully. Using try blocks, the application is able to print the exception and try to run the code again.
- Security was implemented with SSL encryption. Using self signed certificate and private key, we ensured that packets being sent could not be sniffed nor eavesdropped on.

This application is implemented with simple features and functionalities. It is reliable in a small scale setting. For further improvements, we want to improve the scalability so that more than 5 clients can be connected, additional chatting features, and more functionality.

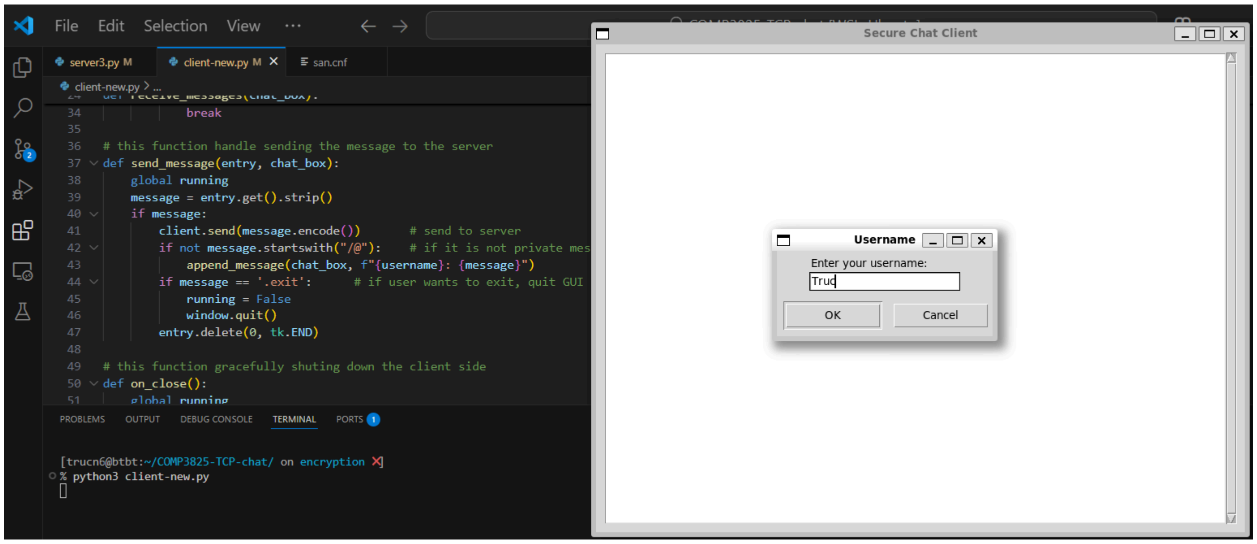
Demo of Application:

- Starting up the server:



```
[trucn6@btbt:~/COMP3825-TCP-chat/]
% /bin/python3 /home/trucn6/COMP3825-TCP-chat/server3.py
Server started... Waiting for clients to connect.
```

- When a client connects

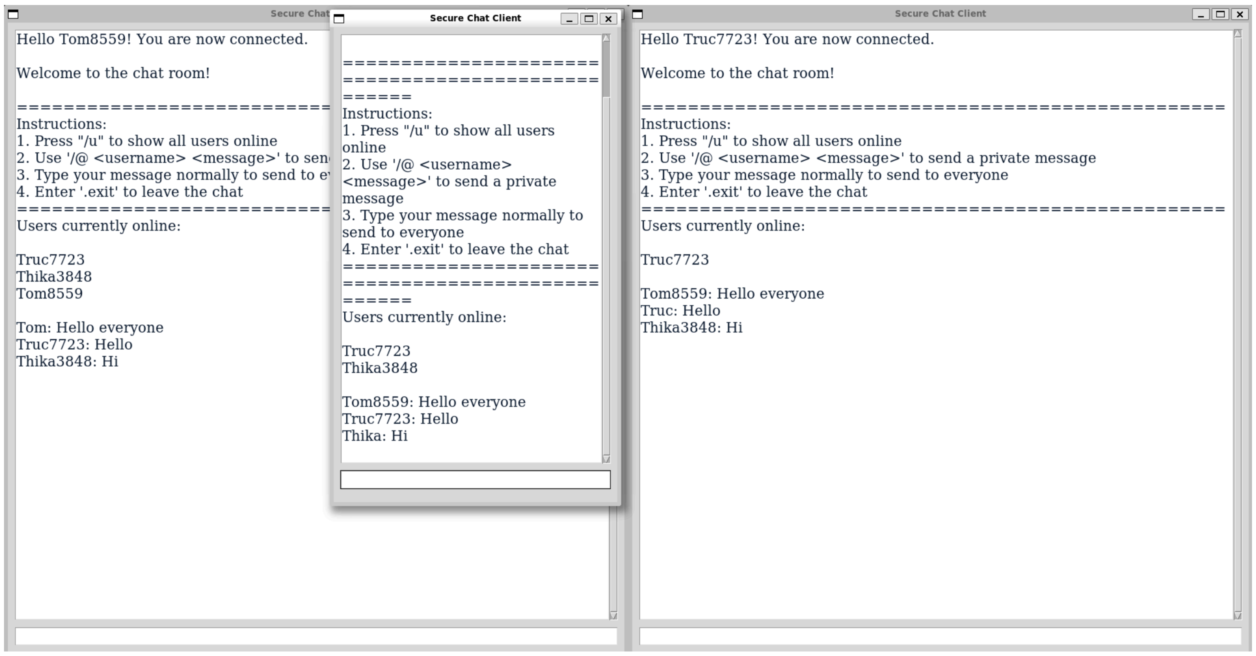


```python
def receive_messages(chat_box):
            break

# this function handle sending the message to the server
def send_message(entry, chat_box):
    global running
    message = entry.get().strip()
    if message:
        client.send(message.encode())       # send to server
        if not message.startswith("/@"):    # if it is not private mes
            append_message(chat_box, f"{username}: {message}")
        if message == '.exit':       # if user wants to exit, quit GUI
            running = False
            window.quit()
        entry.delete(0, tk.END)

# this function gracefully shuting down the client side
def on_close():
    global running
```

```
[trucn6@btbt:~/COMP3825-TCP-chat/ on encryption ✗]
% python3 client-new.py
```

Enter your username:
Truc

- Message chatting between three clients (everyone online)

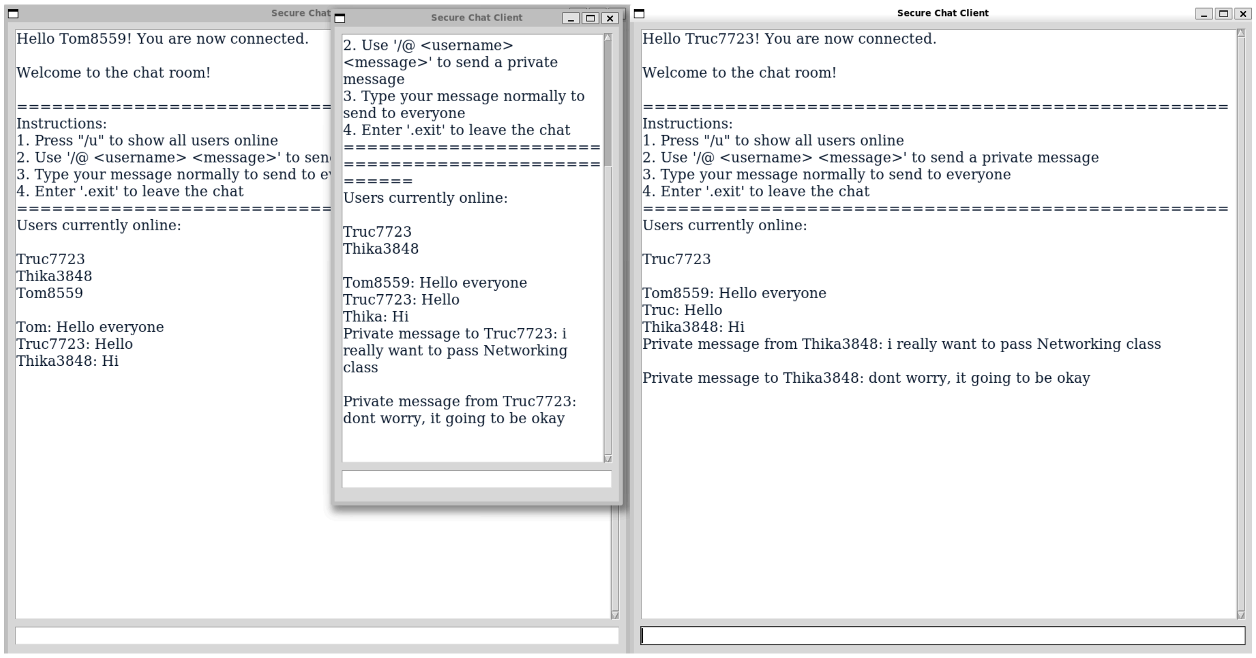


```
Hello Tom8559! You are now connected.

Welcome to the chat room!

============================
Instructions:
1. Press "/u" to show all users online
2. Use '/@ <username> <message>' to sen
3. Type your message normally to send to e
4. Enter '.exit' to leave the chat
============================
Users currently online:

Truc7723
Thika3848
Tom8559

Tom: Hello everyone
Truc7723: Hello
Thika3848: Hi
```

```
======================
======================
======
Instructions:
1. Press "/u" to show all users
online
2. Use '/@ <username>
<message>' to send a private
message
3. Type your message normally to
send to everyone
4. Enter '.exit' to leave the chat
======================
======================
======
Users currently online:

Truc7723
Thika3848

Tom8559: Hello everyone
Truc7723: Hello
Thika: Hi
```

```
Hello Truc7723! You are now connected.

Welcome to the chat room!

=================================================
Instructions:
1. Press "/u" to show all users online
2. Use '/@ <username> <message>' to send a private message
3. Type your message normally to send to everyone
4. Enter '.exit' to leave the chat
=================================================
Users currently online:

Truc7723

Tom8559: Hello everyone
Truc: Hello
Thika3848: Hi
```

- Private messaging between two clients

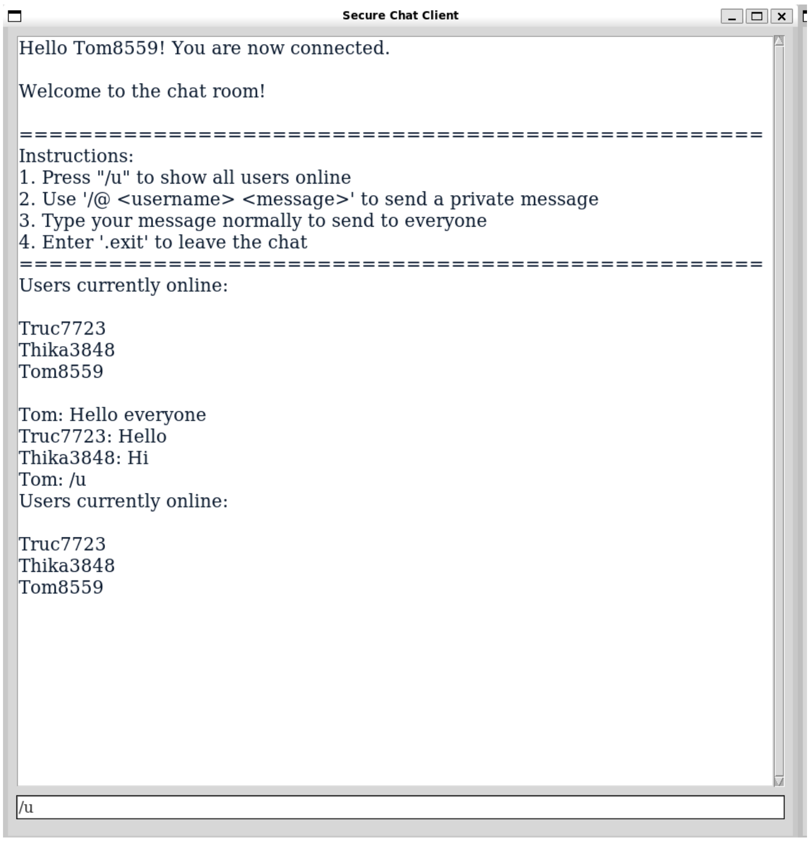


```
Hello Tom8559! You are now connected.

Welcome to the chat room!

==============================
Instructions:
1. Press "/u" to show all users online
2. Use '/@ <username> <message>' to sen
3. Type your message normally to send to e
4. Enter '.exit' to leave the chat
==============================
Users currently online:

Truc7723
Thika3848
Tom8559

Tom: Hello everyone
Truc7723: Hello
Thika3848: Hi
```

```
2. Use '/@ <username>
<message>' to send a private
message
3. Type your message normally to
send to everyone
4. Enter '.exit' to leave the chat
======================
======================
======
Users currently online:

Truc7723
Thika3848

Tom8559: Hello everyone
Truc7723: Hello
Thika: Hi
Private message to Truc7723: i
really want to pass Networking
class

Private message from Truc7723:
dont worry, it going to be okay
```

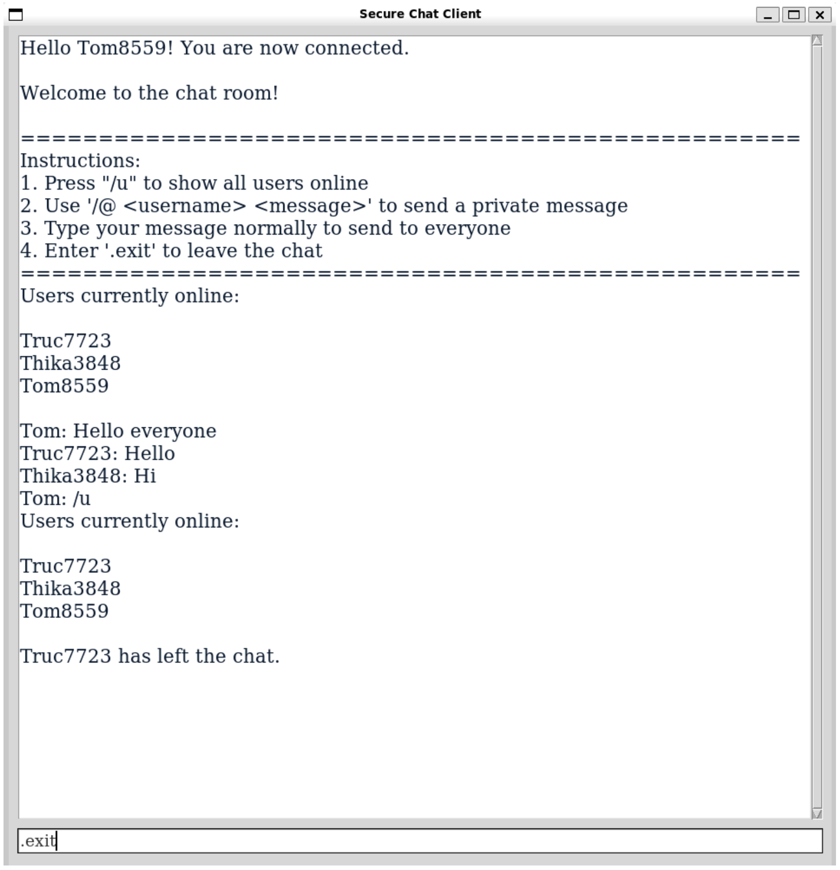
```
Hello Truc7723! You are now connected.

Welcome to the chat room!

==================================================
Instructions:
1. Press "/u" to show all users online
2. Use '/@ <username> <message>' to send a private message
3. Type your message normally to send to everyone
4. Enter '.exit' to leave the chat
==================================================
Users currently online:

Truc7723

Tom8559: Hello everyone
Truc: Hello
Thika3848: Hi
Private message from Thika3848: i really want to pass Networking class

Private message to Thika3848: dont worry, it going to be okay
```

- **/u** to see active users list



```
Hello Tom8559! You are now connected.

Welcome to the chat room!

==================================================
Instructions:
1. Press "/u" to show all users online
2. Use '/@ <username> <message>' to send a private message
3. Type your message normally to send to everyone
4. Enter '.exit' to leave the chat
==================================================
Users currently online:

Truc7723
Thika3848
Tom8559

Tom: Hello everyone
Truc7723: Hello
Thika3848: Hi
Tom: /u
Users currently online:

Truc7723
Thika3848
Tom8559

/u
```

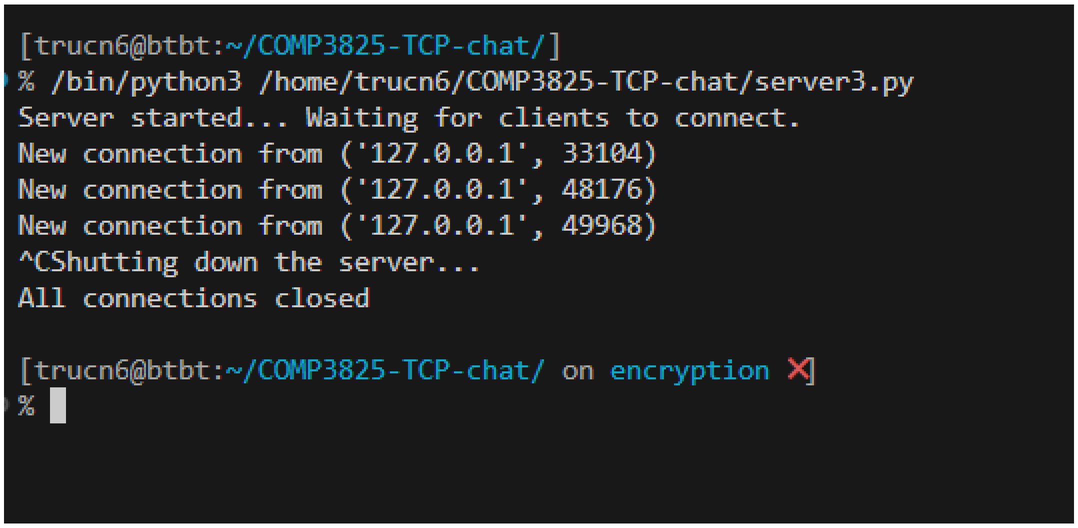
- **.exit** to close connection


Secure Chat Client

```
Hello Tom8559! You are now connected.

Welcome to the chat room!

================================================================
Instructions:
1. Press "/u" to show all users online
2. Use '/@ <username> <message>' to send a private message
3. Type your message normally to send to everyone
4. Enter '.exit' to leave the chat
================================================================
Users currently online:

Truc7723
Thika3848
Tom8559

Tom: Hello everyone
Truc7723: Hello
Thika3848: Hi
Tom: /u
Users currently online:

Truc7723
Thika3848
Tom8559

Truc7723 has left the chat.
```

```
.exit
```

- Server shutting down



```
[trucn6@btbt:~/COMP3825-TCP-chat/]
% /bin/python3 /home/trucn6/COMP3825-TCP-chat/server3.py
Server started... Waiting for clients to connect.
New connection from ('127.0.0.1', 33104)
New connection from ('127.0.0.1', 48176)
New connection from ('127.0.0.1', 49968)
^CShutting down the server...
All connections closed

[trucn6@btbt:~/COMP3825-TCP-chat/ on encryption ✗]
%
```

Final Code:

You can access our final code with two ways: use the github link to see our github repo or use the zipped file we uploaded to canvas with this file

Github link: https://github.com/thikayou/COMP3825-Tktalk.git

Instructions on how to install and run the program:

Method 1: Using zipped file

1. Download the **.zip** file and unzip the file
   After unzipping, the folder will contain
   - san.cnf
   - server.py
   - client.py

2. cd to the project directory (Tktalk)

3. Install all dependencies
   - Tkinter is usually built in, however, for Linux user you can install it again just in case
   - Make sure Python3 is installed
   - Run the following code in your terminal:

```
sudo apt-get install python3-tk
```

4. Configure server IP-address in san.cnf file
   Note: Modern versions of OpenSSL require IP to be listed as Subject Alternate Name (SAN) in the certificate. Please make sure that the CN field and your hostname in the server.py and client.py files are the same. An example of running on a local machine, if HOST = 127.0.0.1, CN must also need to be configured as 127.0.0.1, not ‘localhost’.

   The CN field will be originally configured with a local host address like below. If you want to run the program on different machines, please change it to the appropriate IP address. The alt_name field is the additional IP addresses that the certificate is valid for.

- Please open the san.cnf file, add in your server IP address and save the file

```
[ req_distinguished_name ]
CN = 127.0.0.1
```

```
[ alt_names ]
IP.1 = 127.0.0.1
```

5. Use san.cnf to generate certificate and key
    - Please run the following command (all at once) in your terminal within the same folder as the project files

```
openssl req -x509 -nodes -days 365

-newkey rsa:2048

-keyout key.pem

-out cert.pem

-config san.cnf
```

6. Run the server code
```
 python server.py
```

7. Run the client code
```
python client.py
```

Method 2: Git Hub

1. Clone github repository to your local machine
```
git clone git@github.com:thikayou/COMP3825-Tktalk.git
```

    or

```
git clone https://github.com/thikayou/COMP3825-Tktalk.git
```

2. cd into the project folder (Tktalk)
3. Please repeat the steps from method 1: step 3 to step 7