Communication Networks project: Secure

Ravi Agarwal and Yde Sinnema

December 23, 2021

1 Introduction

This project applies some concepts taught in the course of Communication Networks to the practical example of a chat room. To this end, an application is made on which people can have conversations. A special attention lies on security: user authentication and message encryption are important parts of the assignment.

The app is programmed using Python with some additional libraries on top of it. This language has been chosen because both group members are familiar with it and because a large amount of online documentation is available.

The app is called Secure as a reference to the focus of the assignment on the security of the communication. Secure is a word that is strongly related to the content of this course.

2 Architecture

The largest files in the repository are server.py and client.py. The former is supposed to be executed on one terminal and instantiates a server via which the communication happens. The latter can be run on a larger number of terminals, allowing them to connect to the server, enter the chat room and send messages through a graphical user interface. For the networking we use python library socket which uses TCP as the communication protocol. TCP ensures each message is delivered once and in the correct order.

2.1 Registration and Login

On the client side, the first step is to register or login. A new user can register using the Register button and then fill out the username and password of his/her choice. Once registered, an user is allowed to login using the Login button with the same details used during the registration, then continue to the chatting screen.

The first window is as displayed as shown below, figure 1. Clicking on one of the buttons opens a new window. The Register window is shown as figure 2, figure 3 shows the Login window.

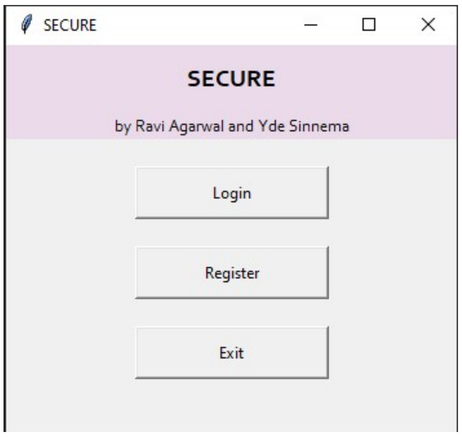
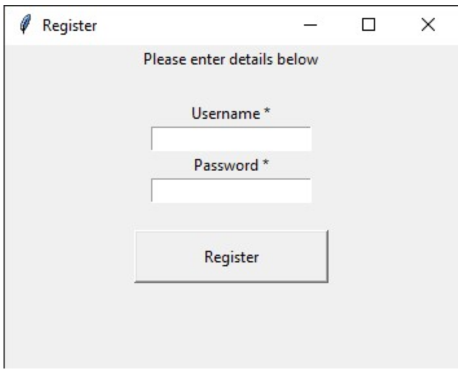


Figure 1: Screen capture of the home page

After registration, the user details like username and password are stored in a file of which the name is the same as the username. As storing the password in plain text is not secure, we store the


Register

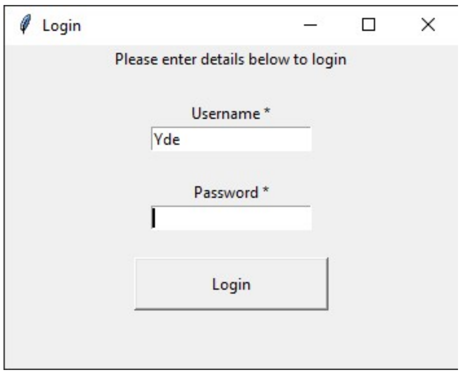
Please enter details below

Username *

Password *

Register

Figure 2: Screen capture of the registration window


Login

Please enter details below to login

Username *

Yde

Password *

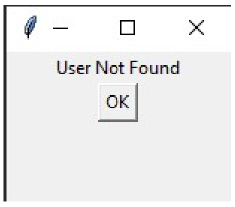
Login

Figure 3: Screen capture of the login window

hashed password in the file. Hashing of the password is simply done in python using the hashlib library.

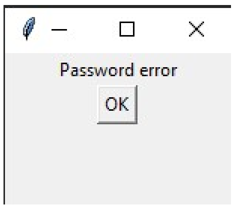
The registered client generates a private and a public key using the rsa library. The two numbers that make up the public key are stored in the same file as the username and the hashed password.

Login first checks if the file is available with the username. If yes, then it verifies the username and hashed password stored in the file with the same name as that of the username. If file is not available with the same username it displays an error message. It also throws an error if password is not matched with the corresponding username. The following figures 4, 5 shows the same.


User Not Found

OK

Figure 4: Message shown when an unknown username is entered


Password error

OK

Figure 5: Message shown when an incorrect password is entered

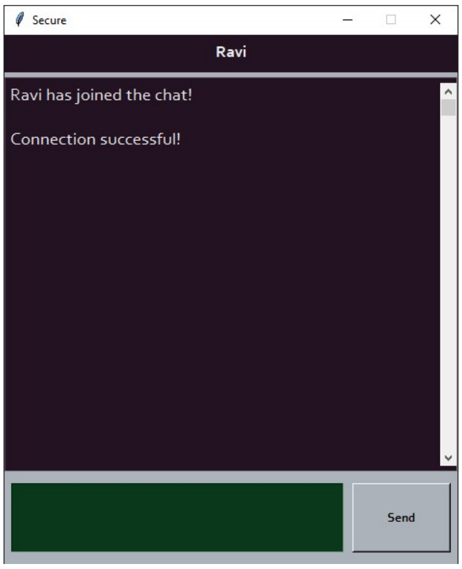
Before continuing to the chatroom, some handshaking needs to be done. Initially, when the client requests connection to the app, the server sends a message asking for the name of the user. When this message has been replied, the identity of the user can be verified.

For the process of user authentication, the server generates a random message called nonce. It also fetches the client's public key from the user database. The nonce can then be encrypted with the public key and sent. The client retrieves the nonce using its private key and sends it back to the server, which can finally conclude if the user is who he claims to be. Attempts were made to include this procedure in the code. However, faced with the problem that there was no suitable place to securely store the private keys, it has not been implemented.

After the identity of the user has been confirmed, he is allowed to receive the symmetric key used for the encryption of the messages, generated by the server using the cryptography library. Ideally, this key should be asymmetrically encrypted before it is sent to the client. Due to the same issue, this could not be done. Instead, the key is sent as it is.

2.2 Chatting

Once logged in successfully, the user is taken to the chat screen as shown in figure 6.

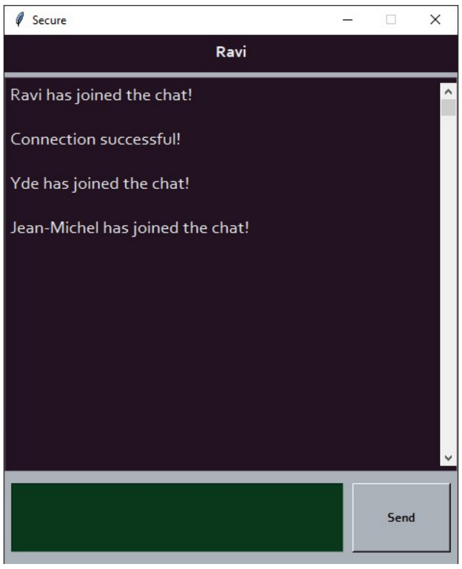


Ravi has joined the chat!

Connection successful!

Figure 6: Chatroom interface when a user enters

Multiple users can be logged in as shown in figure 7 and are able to talk to each other.



Ravi has joined the chat!

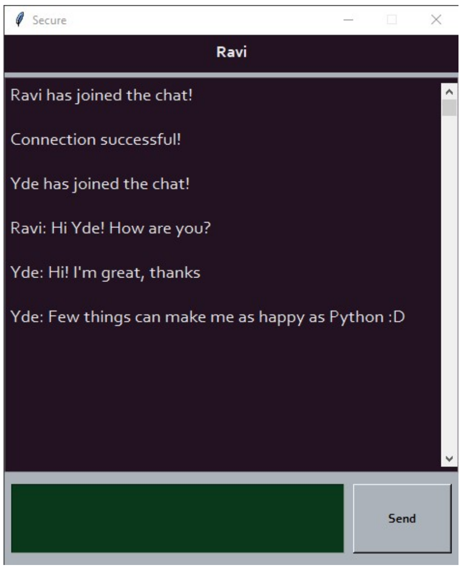
Connection successful!

Yde has joined the chat!

Jean-Michel has joined the chat!

Figure 7: Three clients have logged in to the app

An user can type a message in the text box and then has to click on the send button in order to send the message.



Ravi has joined the chat!

Connection successful!

Yde has joined the chat!

Ravi: Hi Yde! How are you?

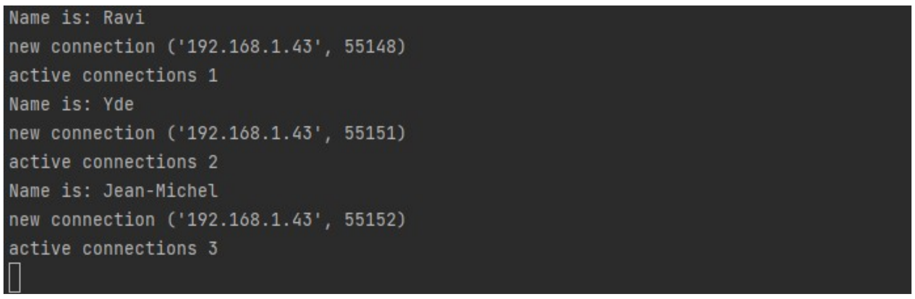
Yde: Hi! I'm great, thanks

Yde: Few things can make me as happy as Python :D

Figure 8: A conversation between two users of the app

To the user, it may look like the messages are sent in plain text. However, there is an encryption stage hidden behind the scenes. The client encrypts every message with the symmetric key belonging to the conversation, generating an illegible sequence of bytes. This is what gets sent to the server, and then to the other clients. After arrival, the same key can be used for decryption and the original message can be displayed to all users.

As seen in the server terminal (figure 9) the server does not have access to any of the messages being sent in the chat room.



```
Name is: Ravi
new connection ('192.168.1.43', 55148)
active connections 1
Name is: Yde
new connection ('192.168.1.43', 55151)
active connections 2
Name is: Jean-Michel
new connection ('192.168.1.43', 55152)
active connections 3
```

Figure 9: The server keeps track of the active connections

3 Creativity

One extra feature which we added in our project is to use a nickname before chatting. Users are allowed to use a nickname and chat with the other users. This allows users to use funny nicknames without having them to create a new account. This is shown below on figure 10.

The background color of the chatting screen was selected as the date (23-12-21) which is in RGB format. It gave us a purple color. It could be a nice idea to have the background color change every day according to the date.

4 Challenges

The main challenges have been mentioned in the sections above. Unfortunately, not all of the problems that appeared could be resolved. The challenge-response authentication scheme, for example, is not included in the project. A larger, better developed structure would be necessary to deal with these problems.

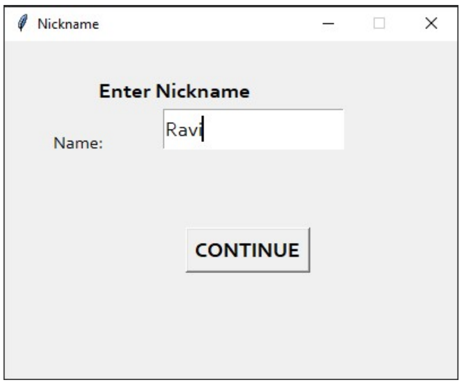


Figure 10: Screen capture of the nickname selection window

5 Conclusion

A chat room app was designed using python and necessary libraries suitable for python. It follows some important principles of network security. The passwords are never stored in the original format. The messages are encrypted and can be only read by the clients and not the server.

Through this project, we have been able to approach the subjects of the course from another angle. Many lessons were learned about programming and secure communication. Besides that, the importance of having a good team to work with also manifested itself. All in all, the project has enabled us to gain valuable experience in different areas.