Université Libre de Bruxelles

# ELEC-H417 - Project Assignment

*Auteur:*
BELYAZID Ali
Jellal Ihssane
Kozlov Vladimir
Rizk Julien

*Professeur:*
Daubry Wilson
Verstraeten Denis
*Cours:*
ELEC-H417

21 december 2021

# Contents

# 1 Introduction

The objective of this project was to design a chat application allowing private communication. The application should be built as follows:

- It has been requested to implement a server based on a centralized architecture so that the server manages the sending of the various packets. The server will take care of most of the tasks such as user creation and authentication. These 2 steps are done by registering and verifying the username and password. In addition to storing the names and passwords of the different users, it will have to store the conversations that the users exchange between them, each conversation will have to be encrypted. Thus each conversation will be assigned its own key. And therefore, in general, the server must be able to ensure the exchange between the users while guaranteeing confidentiality.

- Once connected, the users will use what is called a client which will allow the transfer of information between the user and the server and vice versa. This transfer of information therefore involves, first of all, the entry of a user name and a password for either authentication or connection. Once this step has been passed, the different users should be able to receive and exchange messages privately without any other user logged in at the same time being able to see the exchange. And therefore in general, the client must be able to send information on the chosen encryption key and on the messages to be transmitted to one or more users.

# 2 Main Architecture

## 2.1 Client registration, authentication and login

### 2.1.1 Client

For the first part, in the client.py file, if the connection to the server was successful, the authentication step can start with "def connexion_database (client_socket)" which will display a message on the terminal asking the user if he already has an account, if the response entered is' no 'he will be asked to enter his new identifier and a password that will be sent to the server. When asking the clients for the password, it will first be encrypted before sending it to the server to be stored.

### 2.1.2 Server

The server also uses different instantiated variables to contain information about connections such as the variable "connections = []" which is an empty list and which will store all connections on the server, "total _connections=0" which is a counter of the total number of connections and "HEADER _LENGTH = 256" which ....

Other variables are used to manage the conversations this time and are "conversations_opened =[ ]" and "conversations=[ ]" which list as their names indicates, the open conversations, and the whole conversation respectively.

Thanks to the method def run(self), the server will be able to retrieve all the information it needs to establish the client's connection to the application:

First of all, if the answer to the question whether the client already has an account is no, then the server will ask the client for a new username and password which it will store in the file dico.json. The database in this file is a dictionary type whose keys are the usernames and whose values are the corresponding passwords stored in encrypted form in order to keep the confidentiality of the clients, indeed the server will not be able to decrypt the usernames' passwords, because it does not have any keys. The keys are generated by the client and each time the client creates a new username, then a new .txt file will be created whose name is username.txt and whose content will be the key corresponding to the username. And then the password sent from the client to the server will be encrypted with this key.

Then, if the answer to the question whether the client already has an account is yes, then the server will ask the client again for his username and password in order to verify this data. The verification of this data will be done with the check_if_username_in_database_server method which will open the database file with the load_database method and will check if the data is in the dictionary.

### 2.2 Start a conversation

#### 2.2.1 Client

For this part, once the client is connected to the application, a question will be asked: "Who do you want to start a conversation with?" the client will have to enter the username of the person he wants to talk with, then a series of steps will be checked in the server part, specified in the next section. Then the client will have to accept or refuse to talk privately through a key generated especially for the conversation in question. If the client accepts, he will have to wait for the answer of his correspondent before being able to continue, however if he refuses, it will be a backtracking.

#### 2.2.2 Server

At the server level, after receiving the response from the client to know who he wants to talk to, the server will first check if the corresponding username is in the database and will also check if he is connected, otherwise an error message will be sent to the client. A third condition is that the username given by the client is not itself in order to avoid monologues. Everything said above is implemented for the 2nd client. Once these two conditions are satisfied for the 1st and 2nd client, a key will be sent to both clients to know if they accept to communicate through this key, two possibilities are opened: if both accept, then both clients can start to communicate, if at least one of them says no then they both go back to the previous step. Once the two are ready to talk, the key corresponding to the conversation will be stored in the file keys_dico.json in a dictionary type whose keys are the names of the two members of the conversation and whose values are the keys, if the two members are already in the dictionary, then no new key will be created, on the contrary the already present key will be taken for the conversation.

### 2.3 Conversations

#### 2.3.1 Client

Once the conversation is created, the client will be able to send messages in a decrypted way to his correspondent thanks to the key he accepted in the previous step. In fact, each message sent will be encrypted so that the server cannot read them and will be decrypted thanks to the same conversation key at the correspondent, so all the messages sent between one and the other will be confidential. All this happens in the following functions: def send which will send sockets from client to server and def receive which will receive sockets from server. Obviously since each client has its own thread, these 2 functions will work simultaneously from one client to another.

#### 2.3.2 Server

At the server level, the server will receive the messages from the 2 clients and send them each time to the other, always in an encrypted way. The server has 2 objectives at this level, to send the messages it receives to the other, and to store them in a text file that we create beforehand, this text file will have as a name the name composed of the 2 usernames, and its content will be written in an encrypted way with the name of the user who sent the message and the time of sending of the message that will be placed just before the message. Once one of the two clients leaves the conversation by writing the word exit, the conversation will stop.

## 3 Creativity

### 3.1 Client registration, authentication and login

For creativity, we have implemented several recursive functions, which aim to make the application easier to use for the client. If the client does not have an account and proposes a username already used in the database, then we will ask him to enter a new username and so on. The same principle is done by asking for the password corresponding to the username, if the client is mistaken about the password, he can then propose another password.

### 3.2 Start a conversation

In the same way as above, many recursive functions have been implemented to make it easier for the client to start a conversation, e.g. if the client gives a wrong username, which is not in the database, or is not logged in, or has put in his own username, then the method will remember itself so that the client doesn't have to go through all the steps of identification and reconnection again. Moreover, as for the validity of the conversation key, if one of the two clients does not accept the key, then both clients will go back to the step where they must enter a username to start a conversation.

### 3.3 Conversations

One of the advantages of our implementation is that several clients can connect at the same time and can communicate two by two, in fact the use of threads allows each client to have its own independence. Indeed, if the following users are connected: A, B, C and D, then the conversations A with C and B with D can take place simultaneously. And consequently the messages captured by the server will be stored in an encrypted way in the file corresponding to the conversation.

## 4 Difficulty encountered

Perhaps one of the biggest challenges we've spent a lot of time on is implementing recursive functions in the code, because this has to be written synchronously in the server.py file and in the client.py file, because that we send information from client to server and vice versa the 2 functions must have almost the same structure to avoid at a time that the server does not receive the information at the right time and in the right place and the same for the client. Another difficulty is the satisfaction of the 2 clients for a certain key generated, in fact to be able to allow the client to communicate with each other, they must both accept. But the problem is the case where only one of the two refuses the conversation key, it is therefore necessary to be able to synchronize the 2 responses of the 2 individuals while they choose their choice with because they each have their own threads. The solution found for this problem is the implementation of a global variable which will aim to receive the validation and / or the refusal of the 2 clients

## 5 Conclusion

In conclusion, thanks to this project we have learned a lot about the link between client and server and especially the way they communicate with each other. This project also taught us a lot about data encryption and their confidentiality, when it came to a client's password and messages transferred during a conversation.