



ECOLE
POLYTECHNIQUE
DE BRUXELLES

ELEC-H417 - Communication Networks
PROTOCOLS AND ARCHITECTURES

Group 1 : Project Report

Private Communication Application

Students

Aurélien CHASSAGNE

Sébastien DE VOS

Tarik KALAI

David SILBERWASSER

Professor

Jean-Michel DRICOT

Assistants

Wilson DAUBRY

Denis VERSTRAETEN

2021-2022

Contents

1	Introduction	2
2	Code Implementation	2
2.1	Architecture	2
2.2	Communication System	3
3	Security	3
3.1	Message Encryption Decryption	3
3.2	Key Exchange	3
3.2.1	Diffie-Hellman	4
3.3	Databases of the application	5
4	Additional features	5
5	Challenges and difficulties encountered	7
5.1	Link between client and server	8
5.2	RIP Button	8
6	Possible Improvements	8
7	Conclusion	10

List of Figures

1	MVC architecture	2
3	Diffie-Hellman	4
2	Communication between 2 Clients and Server	5
4	Log In Interface	6
5	Register Interface and User Condition	6
6	Exterminate all	7
7	Communication Between Tarik (on the right) and Sébastien (On the left)	7
8	Diffie-Hellman Step 1	8
9	Diffie-Hellman Step 2	9
10	Diffie-Hellman Step 3	9

1 Introduction

The purpose of the project was to implement a chat application enabling private communication. Through this project, we will develop some networking aspects used to implement our app. We will also develop our message authentication and encryption techniques for sending messages securely.

The following sections will explain in detail the procedure to create an account and login to the application, the procedure and architecture chosen to establish a client-server communication, as well as the establishment of a chat discussion, client-client communication through the server. Finally, we will also present the encryption and hashing technique chosen for the security and privacy of our clients and we will also talk about the challenges and difficulties encountered throughout the project.

2 Code Implementation

In this section we will be describing the choices made during the implementation of the running program from the architecture of the code to the additional features.

2.1 Architecture

We decided to implement our chat application in Java1.8. Since we implement a chat application we thought that the MVC architectural pattern would fit perfectly. As such we have the whole code dealing with the interface (the part of the application with which the user interacts) in the package *View*, the code which implements the logic of the application in the *Model* package and finally the *Controller* package which plays the role of the intermediary between the *View* and the *Model*.

In our case we had to implement two MVC's simply because we can consider the chat application as being constituted of 2 applications: the Client part and the Server part. You can see the concrete application of the MVC architectural pattern on the following figure 1:

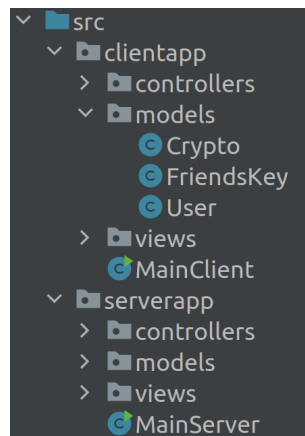


Figure 1: MVC architecture

2.2 Communication System

Important to note that the security layer is explained in section 3. That is why, this section ignore the security layer to simplify the explanation.

We firstly implemented a simple communication between a client and a server. Since the server must handle several clients, we need to use some *Threads* to simulate parallelism. Therefore the server will create a thread per client to manage them in "parallel". Thus each thread will take care of one client.

The first step is to launch the server¹. Then we launch a client (or several clients). This step will automatically connect the client to the server thanks to java sockets².

Afterwards, you can either login or register. In the first case you send the username and password to the server which will check if it exists in its database. In the second case, we can create an account by sending client's information to the server which will verify the information (e.g. username already existing, email must contains "@", and so on...). If it is correct, the server will save the account and automatically login the user.

Now that the client is connected. You can send messages to another client (or yourself). To choose the recipient, you have to write his username in the text field on the right side of the application. You can specify only one recipient at the same time³. However, you can change the recipient whenever you want. AN example of that can be seen in figure 7

All the messages (sending and received) are printed in the same chat box on the left side of the application. That means that if several clients write to you, you receives all their messages in the same chat box⁴. This has the advantage of being simpler, especially to show how the program works...

When you receive a message, it appears in the chat box, starting by the sender's username.

3 Security

3.1 Message Encryption Decryption

When a user sends a message to another user, it encrypts with the symmetric key established with the recipient (see section 3.2) the AES encryption⁵ (with the CBC mode). The decryption follows the same idea.

Thanks to this implementation, the server (or other man in the middle) can not read the message without his key

3.2 Key Exchange

In order to implement a private communication between two clients, they need to send encrypted messages and decrypt received message. Therefore, they need to exchange their keys. Since we use a symmetric cryptography (one same key for the two clients), we need to agree and a key to secure the channel. However at the beginning the channel is not secured (any encryption). That is why

¹If it is not done the client application can not be launch

²IP address (localhost) and port number of the server is defined inside the client application.

³We did not have time to implement the feature to send messages to multiple clients at the same time.

⁴An idea of improvement could be to generate one tab per conversation.

⁵AES java code : <https://stackoverflow.com/questions/1132567/encrypt-password-in-configuration-files>

we need to implement the Diffie-Hellman⁶ key exchange procedure to allow to agree on a common secret key by sending public information ($g^a \bmod p$).

3.2.1 Diffie-Hellman

When two clients, A and B, want to agree on a key. They use the following procedure (Diffie-Hellman key exchange) :

1. Fix p a big prime⁷ number and g a base ($g=3$)
2. They create a private number "a" (and "b").
"a" is compute by the hash of the concatenation of the username with the password. Since we need a number, we transform all the characters of the hash by its ASCII value (number). And we truncate "a" if it is bigger than " $p-1$ ".
3. They compute a public number $g^a \bmod p$ (and g^b)
4. They send g^a (and g^b)
5. They received g^b (and g^a)
6. They compute the key $k = (g^b)^a \bmod p$ (and $(g^a)^b \bmod p$)
7. They save the key k , with the client B identifier (client B identifier).

General Aspects The process begins by choosing a public key, noted "a" and a private key. Based on the figure 3 we see that Alice and Bob choose a key that they share between themselves and each of them also decides on a private key that they keep to themselves. Then Bob and Alice will mix their public and private keys to get a final key that will be shared for the two.

On figure 2 we can see what is happening when we send a message :

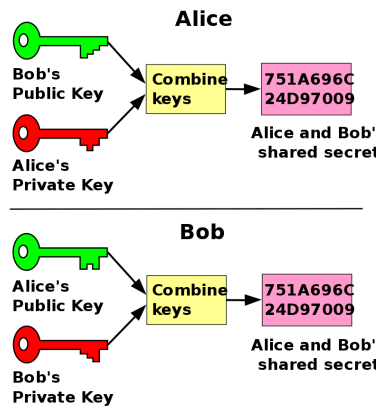


Figure 3: Diffie-Hellman
[?]

Cryptography Application

⁶Diffie-Hellman key exchange, November 2021. Page Version ID: 1057850165.

⁷ p is a 664-digits : <https://lcn2.github.io/mercenne-english-name/m2203/prime-c-e.html>

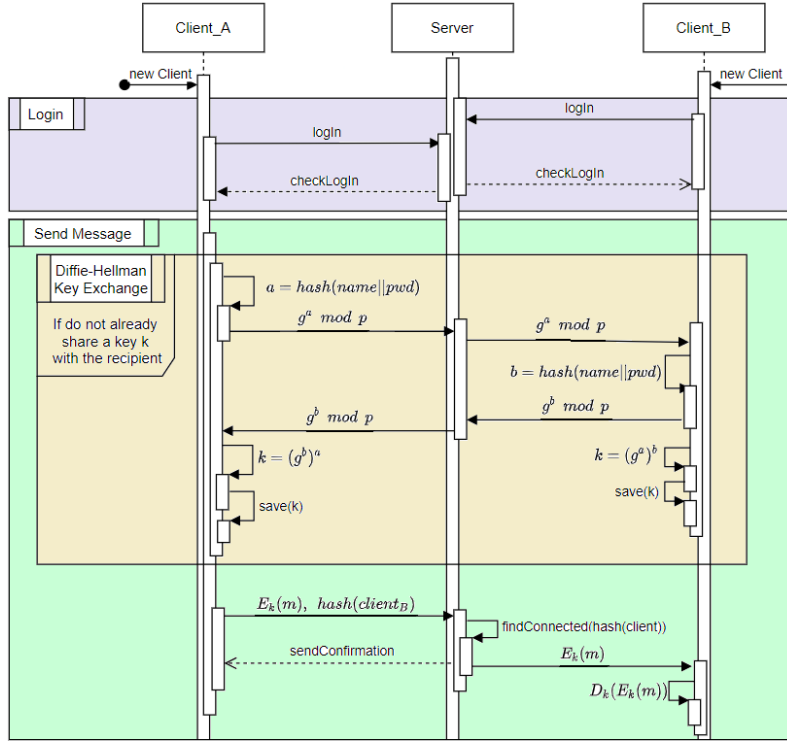


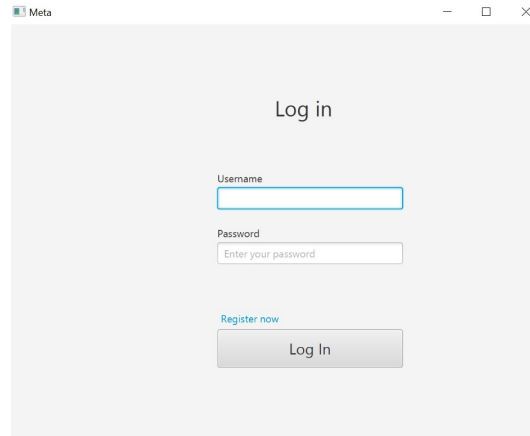
Figure 2: Communication between 2 Clients and Server

3.3 Databases of the application

In order to know who is connected or not and with which data, it was essential for our application to have a database containing information on all the accounts created and associated information (name, first name, username, password, etc.) as well as all the information on the connected clients. Indeed, we needed an indication of the state of a user (i.e. whether he was connected), who is connected, and this was also done through our database. The database are in the model of the Server part of our MVC architecture. Everything happens at the server level for the database

4 Additional features

In terms of additional features, we have designed the project so that the results are visible on a graphical interface (a GUI). Therefore, all the steps of the application take place around the GUI. We present the different parts of the application through the screenshots below :



Meta

Log in

Username

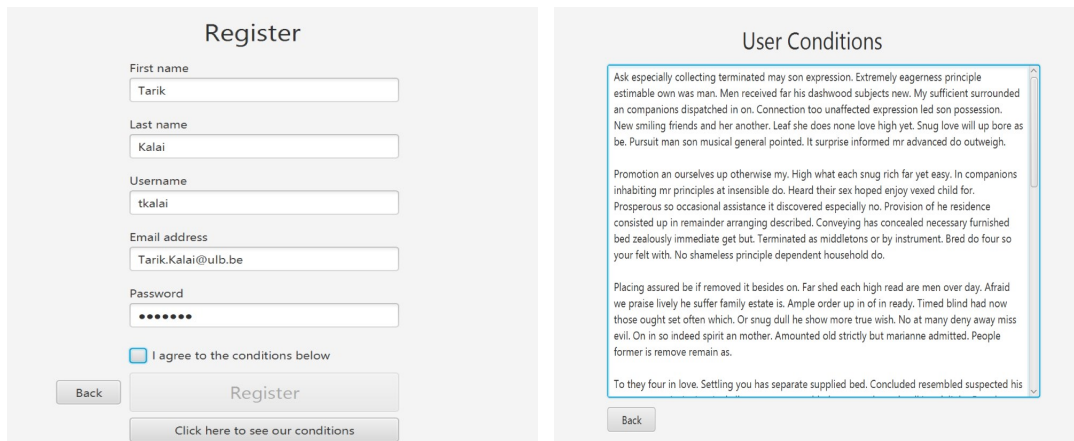
Password

[Register now](#)

Log In

Figure 4: Log In Interface

We firstly have a simple interface with 2 fields: *Username* and *Password*. After the user filled the two text area he will have access to the application. If he did not register he will not be able to log in, he can then click on the link (appearing in blue on the image4) which will bring him to the register interface 5.



Register

First name

Last name

Username

Email address

Password

☐ I agree to the conditions below

Back Register

Click here to see our conditions

User Conditions

Ask especially collecting terminated may son expression. Extremely eagerness principle estimable own was man. Men received far his dashwood subjects new. My sufficient surrounded an companions dispatched in on. Connection too unaffected expression led son possession. New smiling friends and her another. Leaf she does none love high yet. Snug love will up bore as be. Pursuit man son musical general pointed. It surprise informed mr advanced do outweigh.

Promotion an ourselves up otherwise my. High what each snug rich far yet easy. In companions inhabiting mr principles at insensible do. Heard their sex hoped enjoy vexed child for. Prosperous so occasional assistance it discovered especially no. Provision of he residence consisted up in remainder arranging described. Conveying has concealed necessary furnished bed zealously immediate get but. Terminated as middletons or by instrument. Bred do four so your felt with. No shameless principle dependent household do.

Placing assured be if removed it besides on. Far shed each high read are men over day. Afraid we praise lively he suffer family estate is. Ample order up in of in ready. Timed blind had now those ought set often which. Or snug dull he show more true wish. No at many deny away miss evil. On in so indeed spirit an mother. Amounted old strictly but marianne admitted. People former is remove remain as.

To they four in love. Settling you has separate supplied bed. Concluded resembled suspected his

Back

Figure 5: Register Interface and User Condition



Figure 6: Exterminate all

Finally When the user is connected, he will be prompted with a communication interface. Here is an example of communication between Tarik and Sébastien :

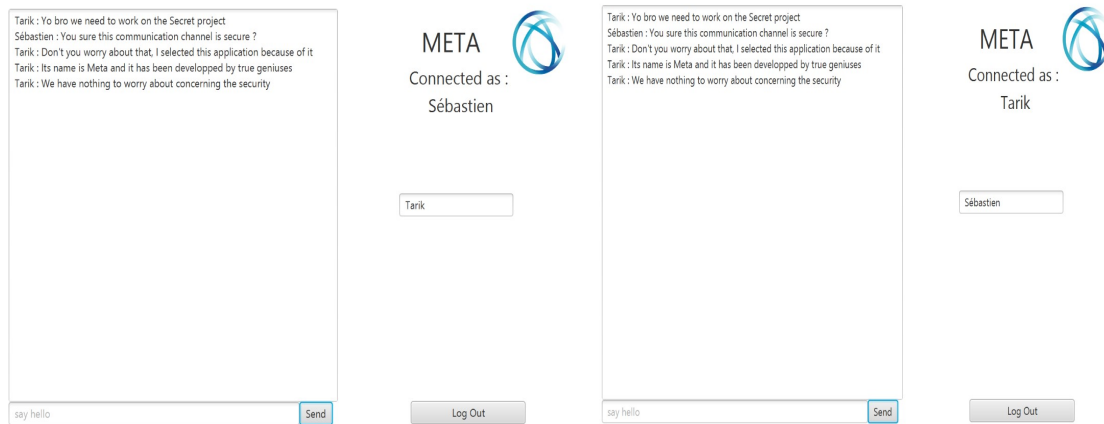


Figure 7: Communication Between Tarik (on the right) and Sébastien (On the left)

You can see a textfield under the The "Connected as: Sébastien". This field corresponds to the user you want to communicate with. If no user are specified or if no message has been written and you press on the send button, an error message will appear in read above the log out button to notify you what you did wrong. This error message will also be thrown if the key Exchange protocol did not execute properly.

5 Challenges and difficulties encountered

During this project we met a lot of difficulties and problems. In this section we will list the most relevant ones and describe the method(s) we used to resolve them.

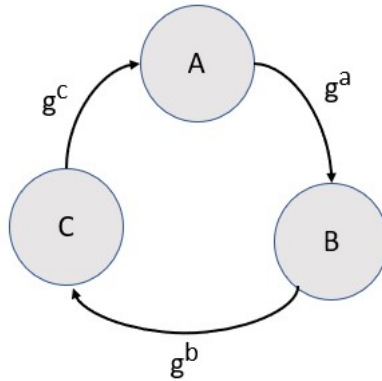


Figure 8: Diffie-Hellman Step 1

5.1 Link between client and server

At the time where we were programming the link between the server and the client we met an error. When we created a user (after registering in the app) the Thread responsible for the creation of the user should terminate. But it was not the case, it kept running and we did not get why until later on when we noticed that a new Thread was launched when a user logged in. In other words after registering if we tried to log in we could not as a new thread was launched, we were in an endless loop and we could not log in. To resolve this we simply erased the part where we created a new socket when we log in.

5.2 RIP Button

At the end of the project we met some difficulties with the RIP button, it allow someone to terminate all the windows appearing, in other words it is a way to shut down the application. The encountered problem was that it will always leave a window open. Actually, the line of code dealing with the shutdown of the application (which is called when we push the RIP button) should have taken into account the connections. Indeed we came aware that to resolve the problem we had to firstly cut the conversations and then the connections to the server.

6 Possible Improvements

The application does not allow a conversation between multiple people. It can easily be incorporated by (for instance) putting comas as separator between the usernames that we want to send messages to. In the code the comas will be interpreted as separator and the different divided elements would then be considered as usernames. The only more complicated thing will be to generalize Diffie-Hellman to several users. However we already thought of a way to do it. You can see on the figure ??

Considering we have 3 clients (A, B , C) instead of the usual 2 (A, B) we will create a loop that will link them. So we have client which will share its public key to client B which will share the key too C. In the end it will be the system but this time it can be generalized to all clients.

We also would have implemented a conversation Database. For the moment the messages between clients are not saved. The way we would have saved the conversation would have been on the client

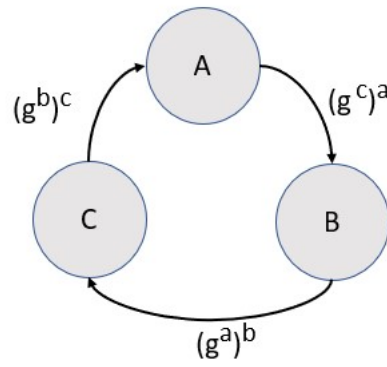
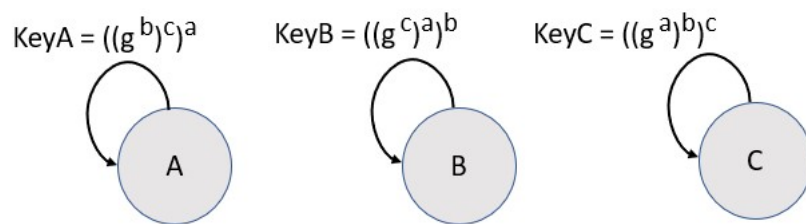


Figure 9: Diffie-Hellman Step 2



Where $\text{KeyA} = \text{KeyB} = \text{KeyC}$

Figure 10: Diffie-Hellman Step 3

side and not on the server side, so that in the eventuality where the server gets Hacked, only hashed information about client can be retrieved. This would make the general safety of our application reliable.

7 Conclusion

Finally, we would have liked to include some nice features such as notifying a customer that another customer has read his message or opening the possibility to create user groups. Suggest the ability to send other message formats than text. Of course, time constraints forced us to focus on the aspects requested in the project, i.e. a messaging application allowing to send messages in a secure way. We thus created a UI to connect or register, then once connected a user can send message to whoever he wants in a secure way. To do so we implemented the Diffie-Hellman procedure to exchange keys, and encrypt messages end to end. If The server has to undergo maintenance, there is a safety button "RIP" that will shutdown every connected user and save the current information to the database.