ELEC-H417

COMMUNICATION NETWORKS

Project : Secure Messaging Application

Authors

Hind BAKKALI TAHIRI

Maciej CZUPRYNKO

Luka GIAPRAKIS

Professor

Jean-Michel DRICOT

School year 2021-2022

Contents

1 Introduction

In the "Communication Networks" course, students were asked to implement a messaging application. This application is divided in two parts : server and client. This division ensures that two users cannot communicate directly with each other but must go through a centralized instance which is the server, through the client. A user should be able to create a new account, login, communicate with every other user who has an account and display every messages between him and the other users. This project was coded in Java.
This report will first explain the server and client implementation. It will then describe how the communication between both is done.

2 Server implementation

The server is the bridge between all the users and the database. When a user wants to access his contacts and display or send messages, he sends a request to the server that communicates with the already existing database to execute the corresponding query. The implementation is done using a $Server$ class, which contains a database. The $start()$ function of this class will create connections to the database and try to associate the users with a different connection each to be able to manipulate several users at the same time. The connection is done with the $Socket$ class implemented in Java. The attribute pool of this class contains threads that correspond to the connections, this number is limited, so there can't be too many users connected at the same time.

Then, for each new connection, the server sends a verification token so that the user can be certain he is connected to the right server. After that the login or registration procedure is handled. Finally, the connection waits for the user to make a query to which the server then responds. Further details one the procedure are given in Section 4.

3 Client Implementation

For the Client implamentation, javafx was used to display the three pages of the application. The classes in the "View" repository communicate with the corresponding classes in the "Controller" repository. The functions in this repository call functions in the $ClientSideServer$ class, which corresponds to the connection to the server for the client. For example, when a user wants to log in to the application, he will enter his username and password in the corresponding boxes, these strings will be sent to the login controller, which will call different functions and finally the function $onLogInAsked$ in the $HeadController$ class. This function will call many functions, and here in particular, the function $newUser$ which will send a request to add a new User to the sever through the socket connection. The server will send

the attributes to execute a query in the database function. Which in this case will be to add a new user.

4 Communication between the server and the client

The communication between the server and the client is done in three parts :

- Authentication and the establishment of a secure link with the server
- Login/Registration to the server
- Communication with the users

All of the above uses asymmetric cryptography keys to assure security and data integrity. Indeed, in the first phase, first the server sends and identification token containing the date as well as the ip address of the client which is signed with the private key of the sever. This way, the user can confirm that he was connected to the right server using the public key of the server, which is distributed with the client software. Moreover, the message contains the date and the ip address so that way an attacker can not reuse previous messages since they change at each connection, and with each user.

Then, the user having confirmed that the server is authentic, sends an symmetric key which will be used for the login or registration. This is encrypted using the server's public key.

The registration is pretty straight forward, the user first asks if the username is taken, if it is he needs to try again, if not he sends his password, username and a public key that he generates and which the user keeps the pair stored in a file. This will come in handy for later use. On the other hand, the password is stored as a hash in the database using PBKDF2 hashing algorithm.

For the log in procedure, the username with the password is sent to the server so he can verify if the two match the ones in the database. If that is the case the user is logged in.

After that comes the final part where the user communicates with others. To do this, the user generates with each message a new symmetric key which he encrypts with the destination user's public key. The latter is given by the server since it was transmitted to it by the user on registration (initially, it was planed the symmetric key was reused for a period of time and only then changed but itw was not implemented due to time constraints). Moreover when the server communicates with the users, he signs the message each times and does not do any more encryption since the messages are already encrypted by the users.

To get the messages, the user query all the messages for a given contact. The database stores the encrypted messages as well as one encrypted symmetric key twice, once encrypted by the public key of the sender of the message and once by the receiver. This way both users can access it.

To get the contacts, the user has to query for all of them.

To recapitulate, first the user verifies that he is connected to the right server with it's public key. Then he establish a secure channel for the login by sending an encrypted symmetric key. The latter is used for the registration and login procedure and is generated for each new connection. Then, the users communicate by sending each other an encrypted message by a symmetric key, which is generated for each message and sent by being encrypted with the public key of the receiver.

5 Challenges

The biggest challenges we faced were the bugs that appeared when we merged the communication part and the implementation of the client and the server. They consumed a lot of time and in the end we could not resolve all of them. Therefore we could not make our app fully functional. There is an exception when the client tries to fetch data which prevents further code execution. However the initial procedure is working as intended to the point just after the login.

6 Previous working version

When adding encryption and undirect communication between client and server, unsolvable errors were raised. Before that, a version with direct access from client to server was implemented. This version can be found in commit e39ae3a1b35f0aa67e368aa424a0a3c0484af40e ("Merge remote-tracking branch 'origin/master' into master")

7 Conclusion

In conclusion, we tried to implement a messenger application using a combination of symmetric and asymmetric cryptography for the message exchange. The server can handle multiple simultaneous connections. Moreover the user can authenticate the server using it's public key and then communicate safely by exchanging a symmetric key generated by the user. Normally, when logged in, the user should then get access to his contacts from where he could send them encrypted messages by generating symmetric keys and sharing the encrypted version with the receiver, of whom he would get the public key from the server. Each exchange with the server and a logged user being signed by the sender using his private key. However due to unresolved bugs the fetching of contacts could not happen.