SECG4

Report security computer project

Marika Winska (55047), Oscar Tison (55315)

Professor: R. Absil

Haute École Bruxelles-Brabant
École Supérieure d’Informatique

Table of Contents

1. Introduction

The main of this project is to implement a chat application where users are able to communicate with each other. To communicate with each other the 2 users have to be contact of each other. If the first user is a contact of the second but the second user is not a contact of the first one, then the messages should not pass. The acquire the security needed every message sent to another contact will be signed. When a user receives a message he first checks whether the message is indeed a message from the sender.

2. Model

The application is divided in 2 main parts. The client side and the server side. 2 users communicate with each other through the server. So the client sends the message to the server and the server then sends the message to the addressed user. The contact part of the application is fully managed by the server. The server does not send the messages to the clients if the 2 users are not contacts of each other.

2.1 Client

The client class is the part of the application that runs on the computer of each user of the app. When this client is run first the connection with the server is made. The address and port of the server is known by every user. If there is no file with the same name as the username a new user is created into the server. To do this the client first generate a private key and the corresponding public key. After this the client sends his public key to the server. A new user can be created on the server. When there is a file on the computer of the user with the same name as the username it means that this client is already a user of the chat app and is known by the server. When it is the case, the client charges the content of the file and transforms it into a private key. After this, the client signs a message with this private key. The server can verify the signature of this message, because at registration the public key was given. If the signed message corresponds to the public key, the user is authenticated and is logged in. When the client wants to add a contact, he has to type add:NameOfContact. The server will respond with the public key of the added contact. When the client wants to send a message to someone he has to type “@namePerson message”, the message then will be signed with his private key and sent to the server. The server then sends it to the right person. The client has a second class, the ReceivedMessagesHandler. This is a thread that will permanently read the data in the input stream. The input stream contains all the data that was sent by the server to the client. This handler checks the messages from the server. When a client asked to add a contact, he receives a message back from the server with the public key of the contact. The handler writes this public key into a file with as name *nameOfContact*Public.key, this makes it possible to have access to the public keys of the contact even after being logged out. Another possible message of the server is a message from a contact. This message will have the following structure #sender#message#signedMessage. The client handler will be able to check if the signedMessage corresponds to the message of this particular sender. This also means that the message cannot contain a #-sign. The client handler will only display the message if the message was successfully verified. When the message is verified, it will also be stored into a file, so when the client is successfully logged in, he can see his previous messages.

2.2 Server

The server class manages the server side of the application. The server has a socket and port that is known in every client. The clients have to connect to this socket. When a new user is created, the server creates an User. The user class contains a nickname, a public key, an input and output stream and a list of contacts. In the server class there is a list of users. When a client asks to register to the application, a new user is added to the list of users in the server. The server must run at every moment because it contains the list of users of the app. This means that with this implementation we are not able to provide a system were there are already users in the system, the users must be added after the creation of the server. When a client tries to log in to the server, the server checks whether the signature can be verified with the public key in the list of users. If the login is successful, a message is sent to all his contacts to warn them that he is logged in. If the client was successfully logged in a user or registered as a new user, a new UserHandler is created.

This thread will read every message sent by the logged in client to the server and do the right thing with it. When the sent message is a request to add a contact, the server sends a notification to the contact to add and responds to the client with the public key of the contact to add. This public key is found in the list of users of the server. This public key can be shared with no problems, it contains no sensitive information and can be shared without doubt. When the message was a request to delete a contact, the contact is just removed from the list of contacts. When the handler received a message starting with ‘@name’, it means that a user tries to send a message to the user “name”. The handler just asks the server to send the message to the addressed user. The user just checks if the users are contacts of each other before sending the message. If it is not the case, the message is not sent. When the user sends ‘logout’ to the server, the server sends a message to all his contacts to say the user just logged out.

The class User represents the users of the app. Every user has a list of contacts and a public key and a nickname. All this information is sufficient to log the contact in.

3. Conclusion

The main goal of this application was make multiple users communicate in a secure way with each other. To achieve this goal every user has a private key and a public key. When a user sends a message, he signs this message with his private key. The addressed user can then verify the message with the public key of the sender. Thanks to this, the addressed user is 100% certain of the integrity of the received message.

4. References

1. Stack Overflow URL: https://stackoverflow.com/questions/52384809/public-key-to-string-and-then-back-to-public-key-java (consulted on 29/05/2021)
2. Github URL: https://github.com/pchampio/java-chat (consulted on 20/05/2021)
3. Baeldung URL: https://www.baeldung.com/a-guide-to-java-sockets (consulted on 17/05/2021)