



UNIVERSITÀ DI PISA

Computer Engineering

Foundations of Cybersecurity

Secure messaging

Project Documentation

Clarissa Polidori
Leonardo Poggiani

Academic Year: 2021/2022

Table of Contents

1	Introduction	2
1.1	Requirements	2
1.2	Security protocols used	3
2	Authentication	4
2.1	Protocol	4
2.2	Message format	6
3	See online users	7
3.1	Protocol	7
3.2	Message format	8
4	Request to talk	9
4.1	Protocol	9
4.2	Message format	10
5	Chat	11
5.1	Protocol	11
5.2	Message format	11
6	User manual	12
6.1	Avvio dell'applicazione	12
6.2	Application usage	12

1 | Introduction

In this project is required to implement a secure messaging system through the use of the Openssl toolkit and the use of the C/C++ programming language.

1.1 Requirements

The behavior of the application can be summarized as follows:

- Users authenticate themselves through a pre-registered public key
- After login users can see the users available to chat
- A user can send a request to another user available to chat
- A user who receives a chat request can decide whether to accept it or not
- If the request is accepted an end-to-end encrypted and authenticated chat is started.

The goal is to create a secure communication between two users who want to talk, allowing them to refuse an unwanted request. Another fundamental point is to allow only properly authenticated and registered users to send requests or see the usernames of other users willing to chat, so as to preserve their privacy.

We can also add more details, which will be needed to justify the design choices made later:

- The server and the client must authenticate as soon as the client application is launched. This is done by the server through a public key certified by a Certification Authority while the client authenticates with a public key pre-installed on the server and the corresponding private key is protected by a password on the client.
- After authentication, the client and server must negotiate a symmetric session key.
- The server is "*honest-but-curious*" and this means that:
 - it will not communicate false public keys on purpose.
 - he could try to understand the content of messages exchanged between users.

Then we have security requirements given by the specifications:

- Symmetric key negotiation must provide Perfect Forward Secrecy
- All session messages must be encrypted, authenticated, and protected from replay attacks.

1.2 Security protocols used

The protocols used for communication between client and server and between client and client are summarized below, for more information and explanations you can refer to the source code in particular in the files "*client.h*" and "*server.h*".

For simplicity in the message formats the size preceding each field is not shown. In fact, messages have a structure like this:

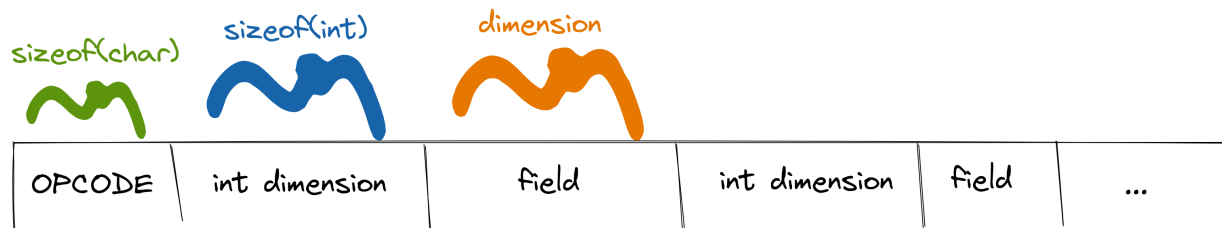


Figure 1.1: Message format

2 | Authentication

2.1 Protocol

In this chapter we are going to justify the design choices made during the authentication phase. In the authentication phase it is the client that takes the first step, requesting a connection to the server via a message. If the connection request is accepted, the client must enter his username and password, which allows authentication through the public key pre-registered on the server. In this way it is possible to send the user's username to the server for authentication. The types of operations that can be performed and their error messages are represented by codes:

```
// opcode
constexpr char AUTH = '1';
constexpr char ONLINE = '2';
constexpr char REQUEST = '3';
constexpr char CHAT = '4';
constexpr char LOGOUT = '5';
constexpr char ACCEPTED = '6';
constexpr char START_CHAT = '7';
constexpr char ERROR_CODE = '8';
constexpr char REFUSED = '9';
```

Figure 2.1: Operational codes

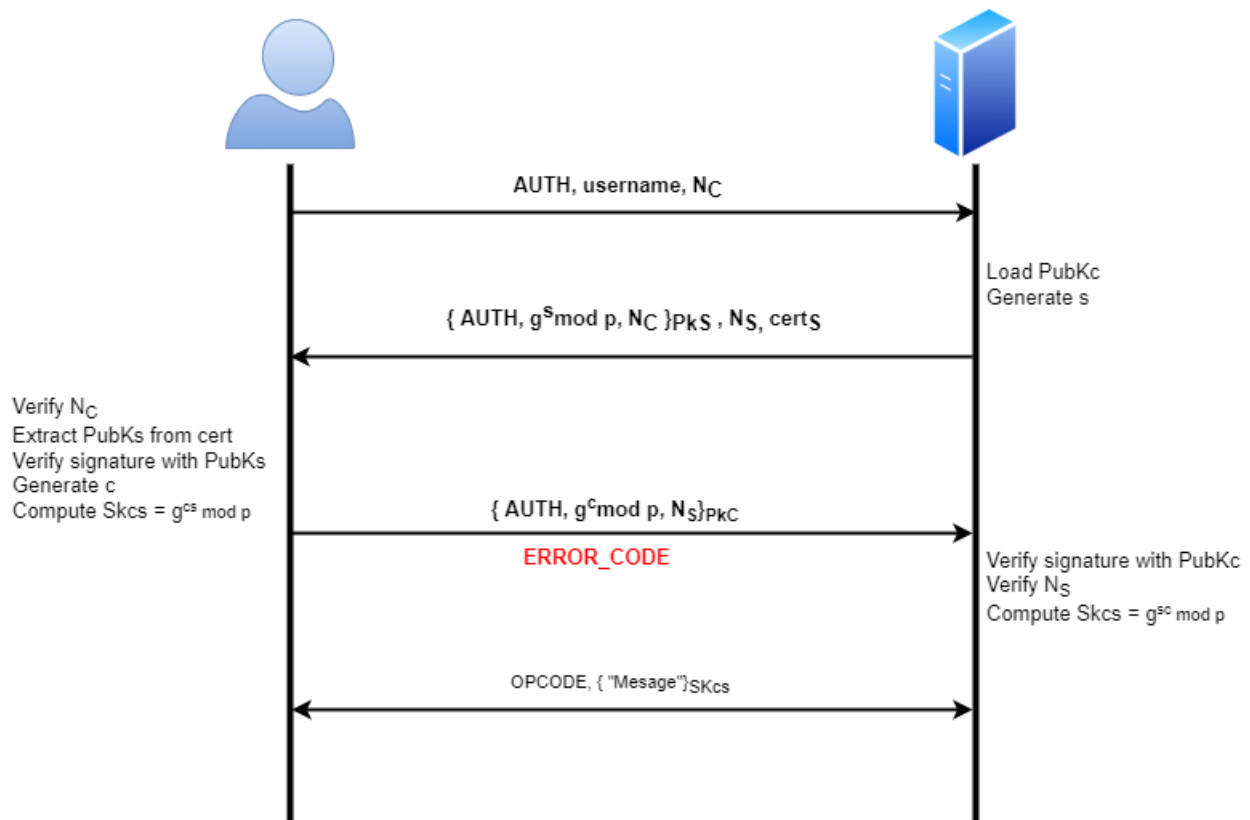


Figure 2.2: Client-Server authentication phase

At this point the client and the server can generate the symmetric key that allows the secure and encrypted communication between them. The session key between the two communicating parties is established using *Ephemeral Diffie-Hellman Key Establishment* protocol to ensure perfect *Forward Secrecy property*.

The entire communication is protected from replay attacks through the use of randomly generated nonce. In particular two nonce are used one generated by the server N_s and one generated by the client N_c .

The messages for the construction of the session key are also signed to prevent *man-in-the-middle* attacks that may compromise the communication between clients and server.

In this way, if two users want to exchange messages with each other they can use the server as a "trusted third party" to send the message to the other client so that there is no direct communication between two clients.

2.2 Message format

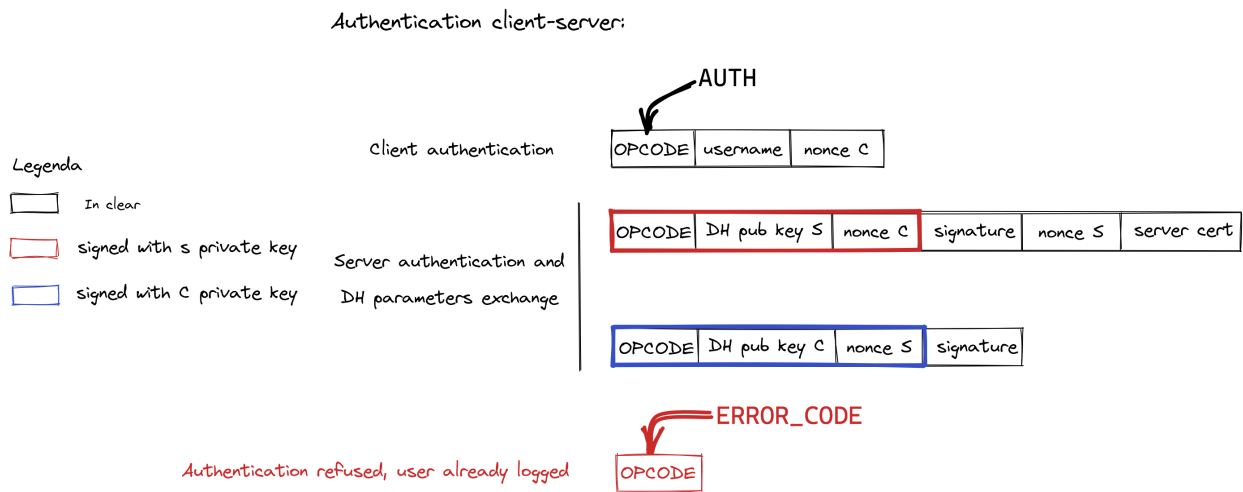


Figure 2.3: Message format for client-server authentication.

From now on, all the messages in a session between clients and server are encrypted with the session key and authenticated using the authentication encryption. In particular, all the messages are been encrypted using AES with a key length of 128 bits as a block cipher and Galois counter mode as the encryption mode. For each symmetric encryption a random generated IV is used, while the AAD is constituted by the opcode, the IV and a counter that counts the number of messages sent by a certain party; the latter is useful to avoid replay attacks during a single session.

3 | See online users

To request the list of online users the client sends a message with opcode *REQUEST* to the server which retrieves the list from an internal data structure. The server creates the response message by adding to the size of the list, each username of a connected client preceded by the size of the username.

The client will first read the size of the list and, if it is different from zero, it will start to read the provided username and it will print it on the screen.

3.1 Protocol

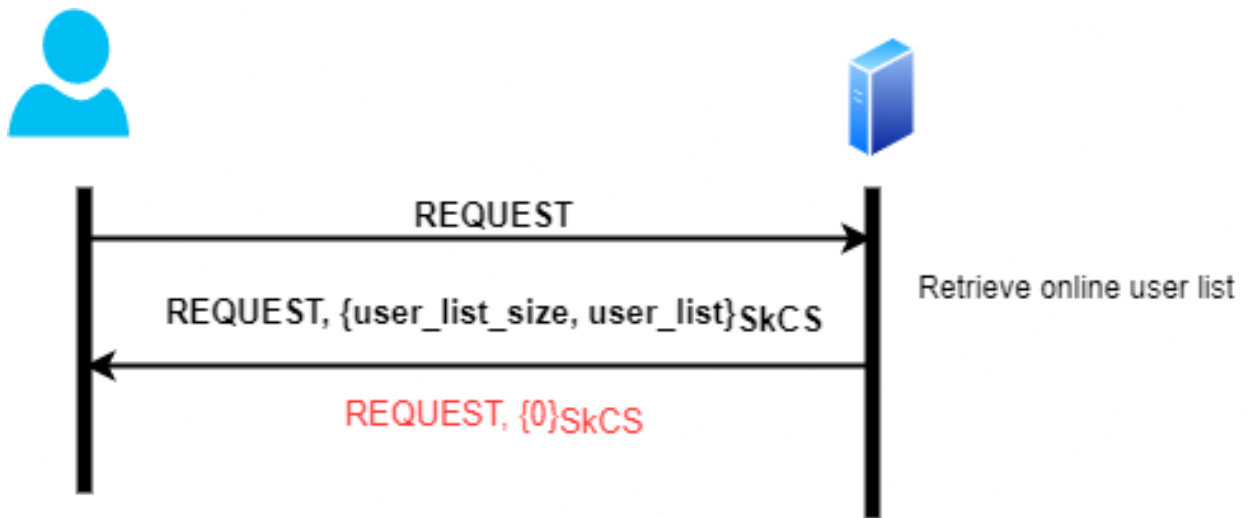


Figure 3.1: Protocol for requesting online users.

3.2 Message format

See Online Users:

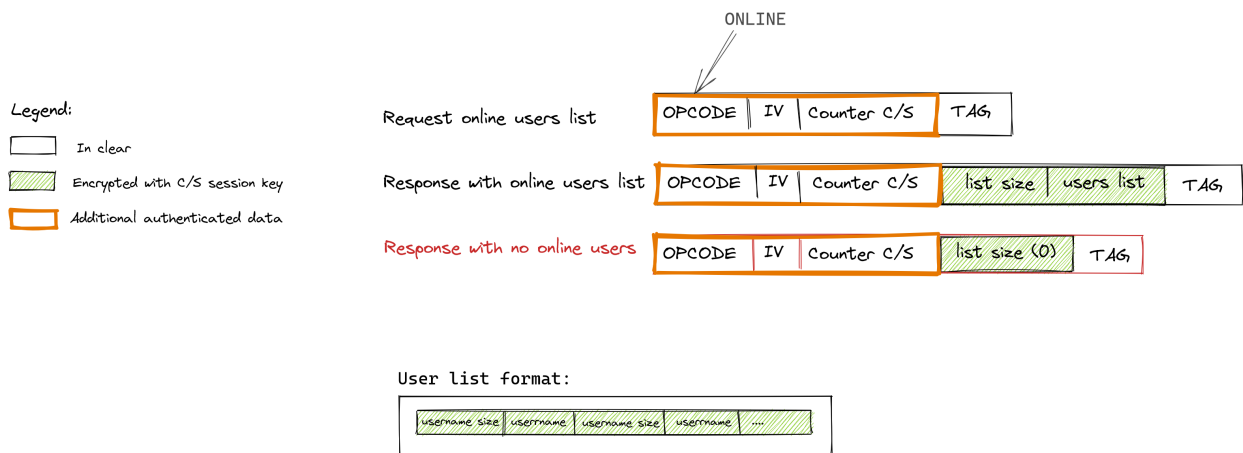


Figure 3.2: Message format for online users request.

4 | Request to talk

To start a chat it is sufficient to select the corresponding option from the menu and enter the username of the corresponding user. The request to talk and the resulting client-client authentication are based on a protocol similar to that of client-server authentication. Using the key establishment provided by Diffie-Hellman, two random values a and b are generated to the two clients, from which the *Diffie-Hellman public keys* are derived. The messages for the construction of the session key are also signed by clients to prevent the intrusion of the server which is *honest but curious* and therefore could act as a *man-in-the-middle* in the communication between the clients. Again, we use randomly generated nonces to ensure message freshness and protect key generation communication from replay attacks. Once the public keys are exchanged between the two clients, the session key valid for the session between client and client is generated. Once the session key is established, the two clients can communicate securely and then initiate a chat.

4.1 Protocol

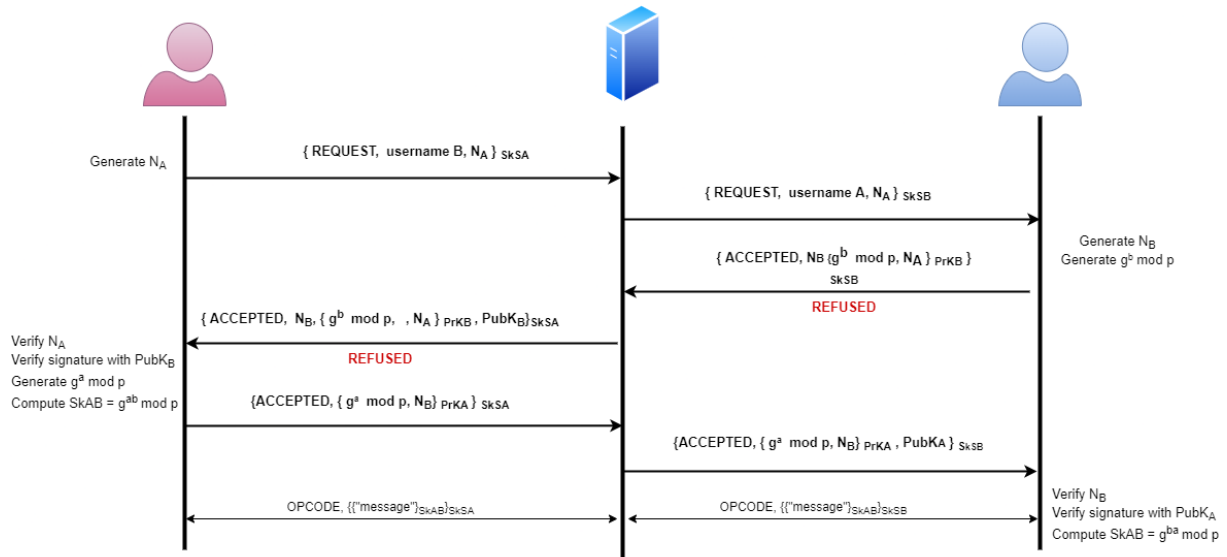


Figure 4.1: Request to talk and client-client authentication phase

4.2 Message format

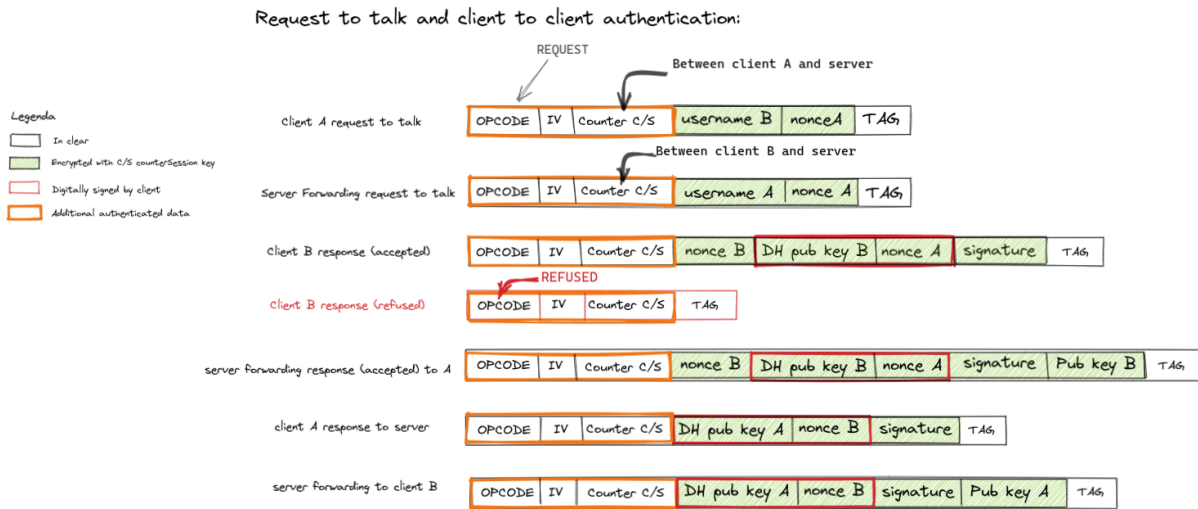


Figure 4.2: Message format for request to talk and client-client authentication.

From now on, all the messages in a session between two clients are encrypted with the session key and authenticated using the authentication encryption. In particular, all the messages are been encrypted using AES with a key length of 128 bits as a block cipher and Galois counter mode as the encryption mode. For each symmetric encryption a random generated IV is used, while the AAD is constituted by the IV and the counter that counts the number of messages sent by a certain party; the latter is useful to avoid replay attacks during a single session.

5 | Chat

Once the session key is established between the two clients, the protocol used for the chat is very simple: client A first encrypts the message with the session key with client B and then encrypts it again with the session key with the server. In this way the server, which operates in "honest-but-curious" mode, has no way to read the messages exchanged between the two clients because it is not able to obtain the session key between the clients.

The clients can then exchange messages securely until one of them decides to abort the chat using the escape sequence ":q!".

5.1 Protocol

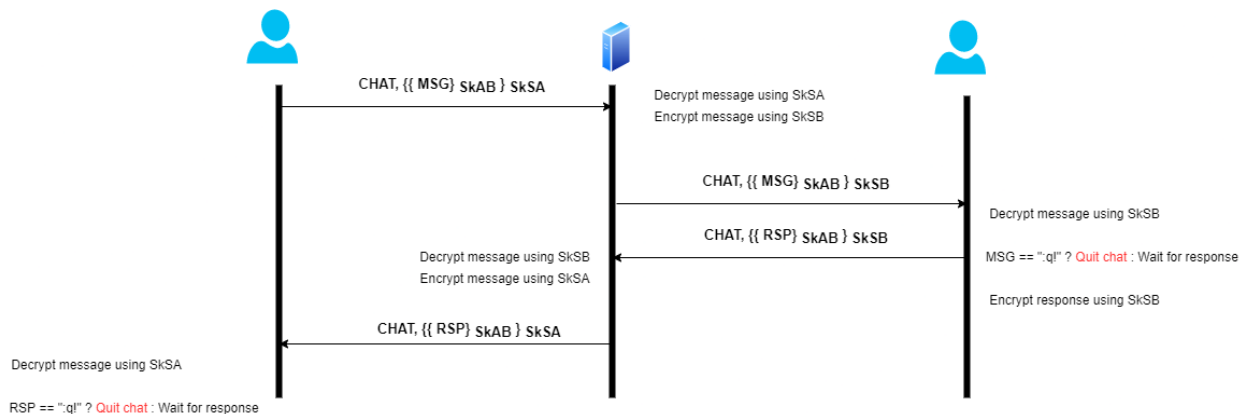


Figure 5.1: Protocol for chatting.

5.2 Message format

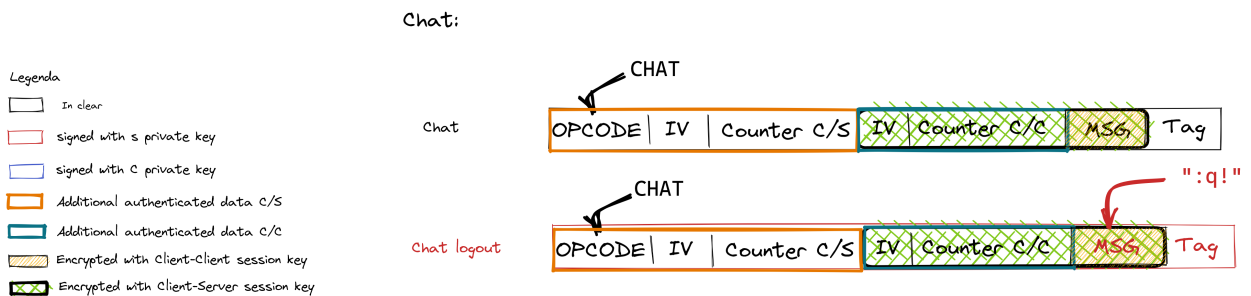


Figure 5.2: Message format for chat.

6 | User manual

This is a simple user manual for the designed messaging app.

6.1 Avvio dell'applicazione

Two simple scripts have been written to start debugging the application through *valgrind* which allows you to detect errors involving memory allocation:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The text inside is a shell script for starting a server with valgrind.

```
#!/bin/sh

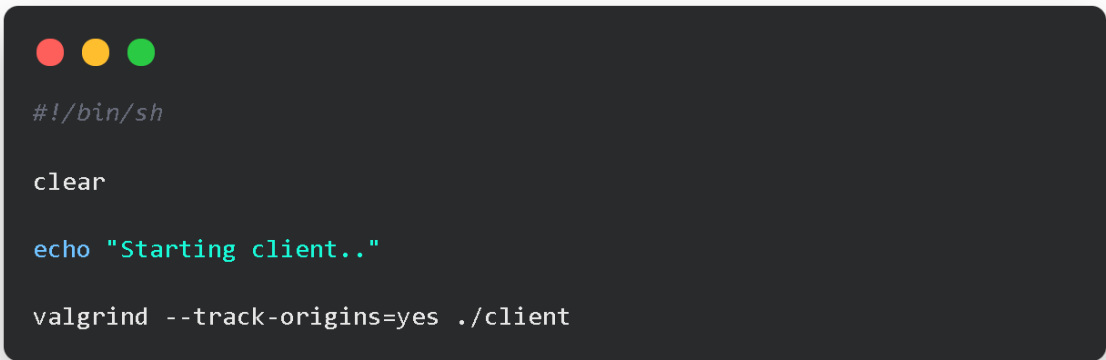
kill -9 $(ps -ef | grep "valgrind" | grep -v grep | awk '{print $2}')

make clean
make
clear

echo "Starting server.."

valgrind --track-origins=yes ./server &
```

Figure 6.1: Scripts to run the server

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The text inside is a shell script for starting a client with valgrind.

```
#!/bin/sh

clear

echo "Starting client.."

valgrind --track-origins=yes ./client
```

Figure 6.2: Script to run the client

6.2 Application usage

First, a message appears asking you to enter your username.

```
Welcome!  
  
Please type your username -> leonardo
```

Then you must enter the password chosen when creating the private key, an incorrect password will lead to an error and logout from the application.

```
Welcome!  
  
Please type your username -> leonardo  
  
Fine! Now insert you password to login  
Insert password ->
```

Once you have entered the correct password, a menu appears from which you can enter the required function. You can enter:

- **1** To display the *Online users list*.
- **2** To send a *Request to talk* to another online user.
- **3** To *logout*.

If you select **1** you will see the list of users currently online excluding the user making the request:

```
Hi! This is a secure messaging system.  
What do you want to do?  
1) See online people  
2) Send a request talk  
3) Logout  
  
Choose a valid option ->  
1
```

```
Hi! This is a secure messaging system.  
What do you want to do?  
1) See online people  
2) Send a request talk  
3) Logout  
  
Choose a valid option ->  
1  
See online users to talk  
  
[LOG] list received  
--- online users ---  
| leonardo - clarissa |
```

If you select **2** you will be asked to enter the username of the user you want to contact. If the user is not available to chat or is no longer online, an error message will be displayed. If the user is available you will be put on hold waiting for his answer.

```
Hi! This is a secure messaging system.  
What do you want to do?  
1) See online people  
2) Send a request talk  
3) Logout  
  
Choose a valid option ->  
2  
Send a request to talk  
  
Type the username -> leonardo
```

The contacted user will then receive a request that he can accept or reject. If the request is rejected, the connection will not be made and the menu will be displayed again to choose the next action:

```
Hi! This is a secure messaging system.
What do you want to do?
1) See online people
2) Send a request talk
3) Logout

Choose a valid option ->

[LOG] Received request to talk**
Do you want to talk with clarissa? (y/n)
n
refused :(
Hi! This is a secure messaging system.
What do you want to do?
1) See online people
2) Send a request talk
3) Logout

Choose a valid option ->
```

```
Hi! This is a secure messaging system.
What do you want to do?
1) See online people
2) Send a request talk
3) Logout

Choose a valid option ->
2
Send a request to talk

Type the username -> leonardo
[LOG] nonce generated
*** waiting for response ***
we're sorry, your request was rejected :(
Hi! This is a secure messaging system.
What do you want to do?
1) See online people
2) Send a request talk
3) Logout

Choose a valid option ->
|
```

```
[LOG] Received request to talk**
Do you want to talk with gaia? (y/n)
y
```


If it is accepted instead the chat will start.

```
-----Chat-----  
you can insert ':q!' to exit the chat!  
-----
```

To exchange a message, just type it and press send.

```
-----Chat-----  
you can insert ':q!' to exit the chat!  
-----  
leonardo:  
*****  
ciao *  
*****  
[ ]
```

To exit the chat, just type ":q!" and press enter. Both users will then be disconnected from the application.

```
-----Chat-----  
you can insert ':q!' to exit the chat!  
-----  
ciao  
  
gaia:  
*****  
ei come va? *  
*****  
:q!  
  
[LOG] user ending the chat  
[LOG] chat ended
```