UNIVERSITÀ DI PISA

Computer Engineering
Foundations of Cybersecurity

ONLINE MESSAGING SERVICE

Project Documentation

Gaia Anastasi
Matteo Guidotti

Academic Year 2020/2021

Contents

1 Introduction

This project is about the implementation of an online messaging model used for the **secure** communication between two pre-registered clients. After the authentication with the server, a user can see the list of online users and (s)he can decide to send a request to another online user to start a conversation. If the receiver user accepts its request the server will send them both the public key of the other client. The two clients authenticate each other and generate a new session key that they will use to encrypt their session messages. If they want to stop chatting they need to logout from the application. Each user can have just one active chat at a time.

2 Design choices

When a user logs in the application he must authenticate itself with the server. The client-server authentication is performed using the client and the server public keys. Supposing that the new logged client is called Alice, the communication between Alice and the server for the authentication and the establishment of the session key is shown in the image below.

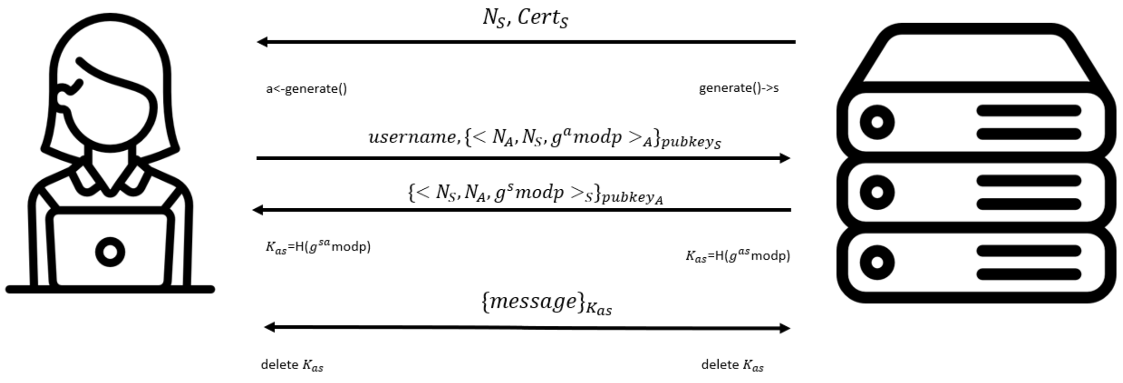
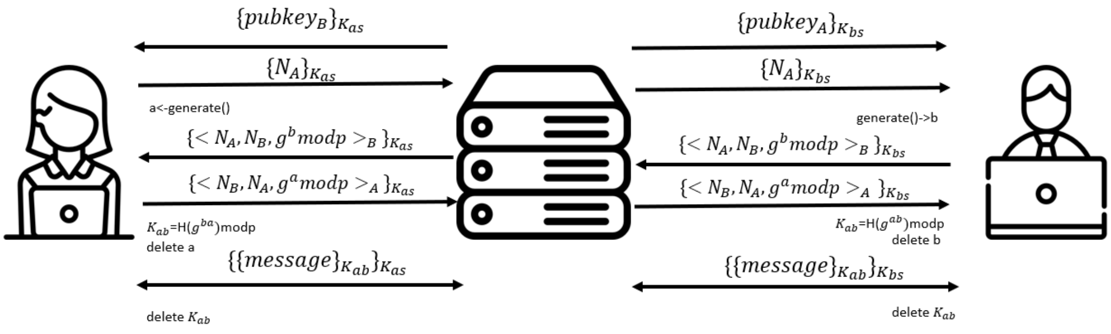


Figure 1: EDHKE between client and server

When a client wants to start another communication with another client (s)he have to send a request to another user using the server as a *trusted third party* and wait until the other user accepts his(her) request. Let's suppose that Alice wants to start a communication with Bob. The communication between Alice and Bob will be like the one presented in the picture below.



$\{pubkey_B\}_{K_{as}}$
$\{N_A\}_{K_{as}}$
a<-generate()
$\{< N_A, N_B, g^b modp >_B\}_{K_{as}}$
$\{< N_B, N_A, g^a modp >_A\}_{K_{as}}$
K_{ab}=H(g^{ba})modp
delete a
$\{\{message\}_{K_{ab}}\}_{K_{as}}$
delete K_{ab}
$\{pubkey_A\}_{K_{bs}}$
$\{N_A\}_{K_{bs}}$
generate()->b
$\{< N_A, N_B, g^b modp >_B\}_{K_{bs}}$
$\{< N_B, N_A, g^a modp >_A\}_{K_{bs}}$
K_{ab}=H(g^{ab})modp
delete b
$\{\{message\}_{K_{ab}}\}_{K_{bs}}$
delete K_{ab}

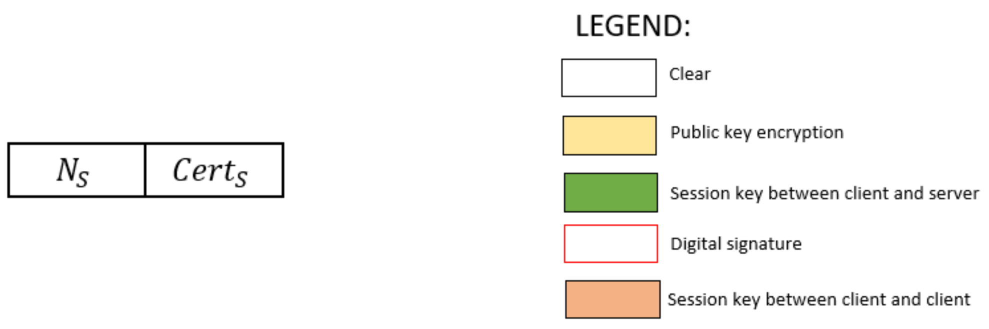
Figure 2: EDHKE between clients

The whole communication is protected against *replay attacks* using randomly generated nonces. In particular, two nonces are used. For example, in the case of the Client-Server authentication, Ns is generated from the server and assure the server about the freshness of the session while the other is generated by the client for the same reason. The same reasoning is applied in the Client-Client authentication.
The session key between the two communicating parties is established using **Ephemeral Diffie-Hellman Key Establishment protocol** to ensure the **Perfect Forward Secrecy** property.
All the messages in a session are encrypted with the *session key* and authenticated using the *authentication enryption.* 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 a counter that counts the number of messages sent by a certain party; it is useful to avoid replay attacks during a single session.

3 Messages format

3.1 Client-Server authentication

From now on we will refer to our client as Alice for simplicity. In M1 the server sends to the client its nonce and its certificate. The message is sent in clear.



N_S
$Cert_S$
LEGEND:
Clear
Public key encryption
Session key between client and server
Digital signature
Session key between client and client

Figure 3: Server sends its nonce and its certificate

After receiving M1, Alice generates her public key following *Diffie-Hellman* key generation protocol and concatenates it with her nonce and server nonce. She signs this message with her private key to prove the server she is really Alice, encrypts the concatenation between the message and the signature by means of the server's public key and sends it with her username. The username is the only part in M2 that is sent in clear (in addition to the information that are necessary for the asymmetric encryption).

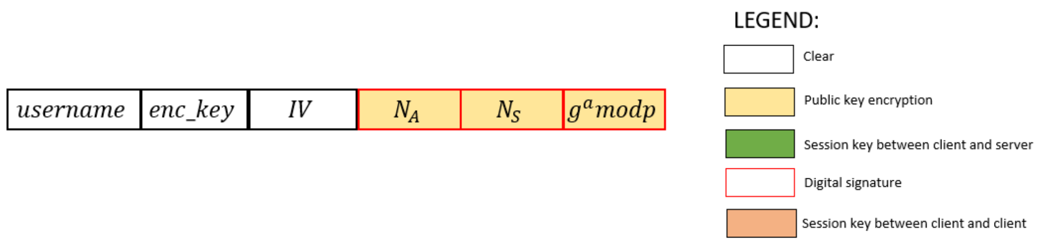


Figure 4: Alice's authentication and DH key generation message

The server generates its *Diffie-Hellman* public key and follows the same path Alice has followed in the previous message. It takes Alice's nonce, its nonce and its public key, signs the whole message and encrypts it with Alice's public key. Then the server sends it to Alice. The difference with the previous message M2 is that the only information that are sent in clear are the ones that are necessary for the asymmetric encryption

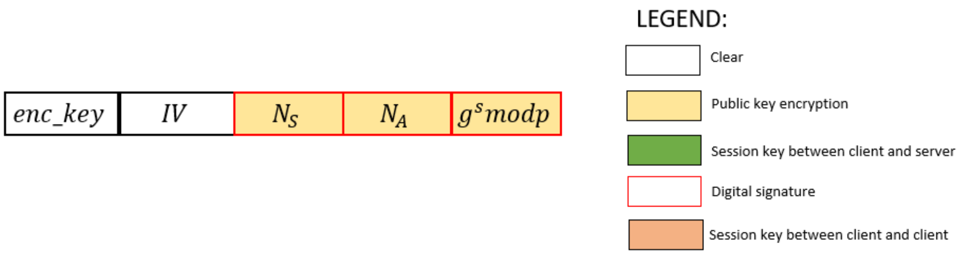
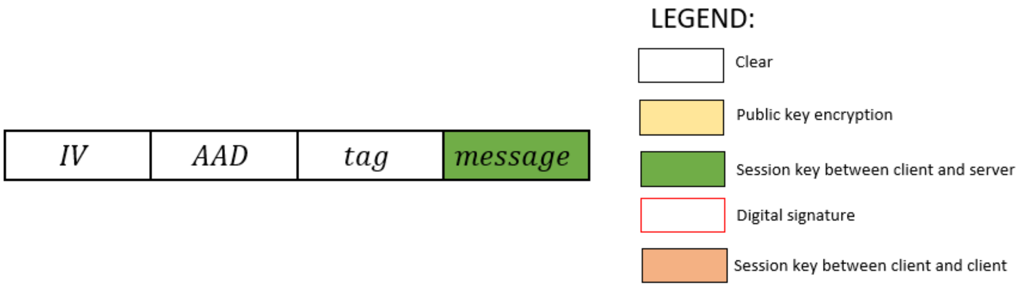


Figure 5: Server's authentication and DH key generation message

After receiving message M2 and M3 server and Alice are both able to generate the session key using the other party public key and their private key following *Diffie-Hellman* protocol. The shared key is obtained first by deriving the shared secret using the two *Diffie-Hellman* keys and then it is hashed by using SHA-256 algorithm in order to obtain the symmetric key. From now on they will use the session key to encrypt all their future messages and they will use *authenticated encryption* to assure the integrity and authenticity of their session messages.



IV
AAD
tag
$message$
LEGEND:
Clear
Public key encryption
Session key between client and server
Digital signature
Session key between client and client

Figure 6: Session message

We will now see some examples of session messages between Alice and the server.

3.1.1 List of online users

Let's suppose that Alice wants to see the list of active users which she could start a new conversation with. Alice asks for the entire list of online users to the server using the following message:

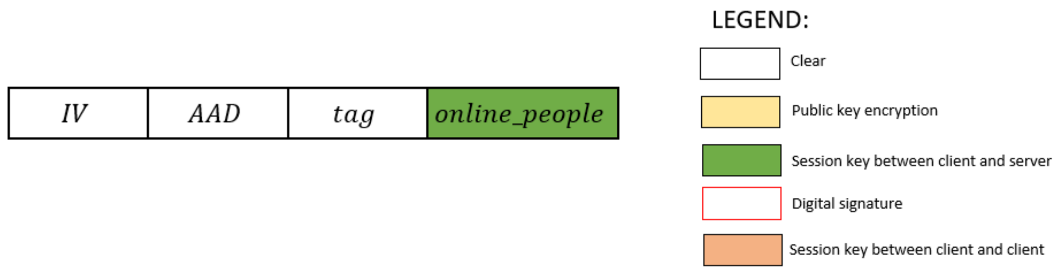

IV
AAD
tag
online_people
LEGEND:
Clear
Public key encryption
Session key between client and server
Digital signature
Session key between client and client

Figure 7: Alice's message format to see the list of active users

The server answers back to Alice sending her the list of active users or, if she is the only active user at that moment, she tells her that there are not other online users.

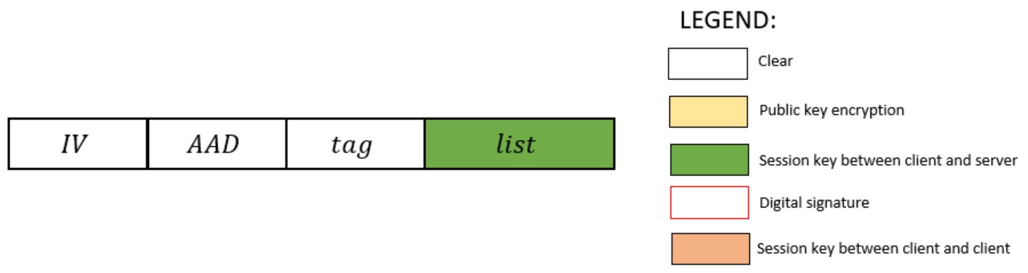

IV
AAD
tag
list
LEGEND:
Clear
Public key encryption
Session key between client and server
Digital signature
Session key between client and client

Figure 8: Server answers sending the list of active users

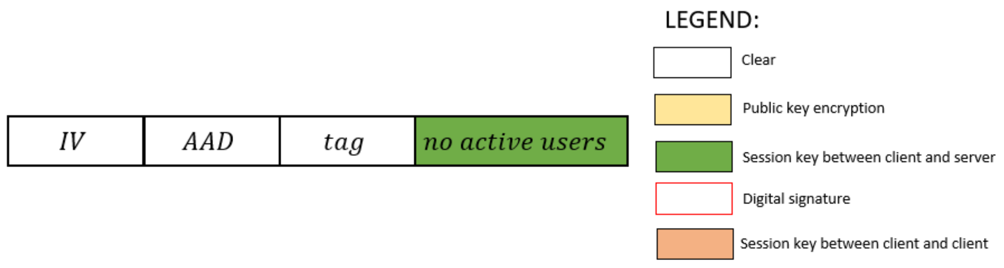
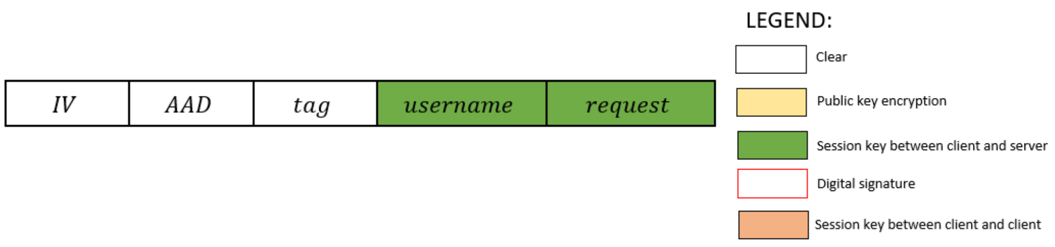

IV
AAD
tag
no active users
LEGEND:
Clear
Public key encryption
Session key between client and server
Digital signature
Session key between client and client

Figure 9: If there are not other active users the server notifies it to Alice

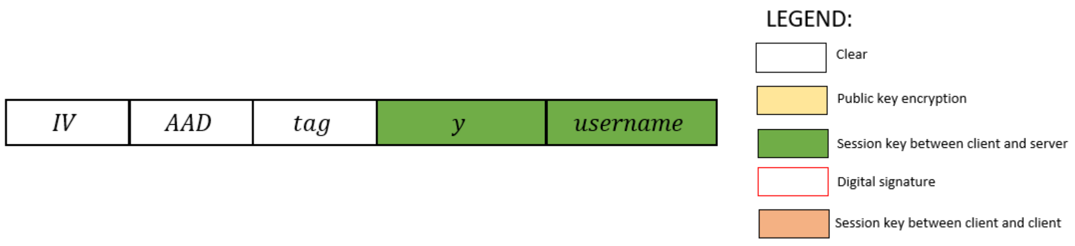
3.1.2 Chatting Requests

Clients can send a chatting request in order to start a communication. The sender client chooses a receiver client from the list and, using the server as a *trusted-third-party*, sends him a request. Let's suppose that Alice wants to start a communication with Bob. Alice encrypts her request to talk with Bob using the session key she has previously established with the server and sends the request to the server itself. The server receives it and makes some checks. It controls that Bob is online and not already busy in another communication and checks the correct format of the request message. If all the checks are successful, it encrypts the request with the session key that shares with Bob and forwards the message to Bob. After receiving the message, Bob decides if he wants to accept the request and start chatting with Alice or if he does not want to communicate with her. Bob encrypts his response with the session key he shares with the server and sends the message to it. The server does the same procedure he have done before with the requests. Checks the response, encrypts it with the session key it shares with Alice and forwards it to her. Here we have just reported the request message and the affirmative response for simplicity, but the negative response have the same format of the affirmative response, while the error message will just contain the error type but not the username.



LEGEND:
IV
AAD
tag
username
request
Clear
Public key encryption
Session key between client and server
Digital signature
Session key between client and client

Figure 10: Chatting request format



LEGEND:
IV
AAD
tag
y
username
Clear
Public key encryption
Session key between client and server
Digital signature
Session key between client and client

Figure 11: Affirmative response format

3.2 Client-Client authentication

In the previous chapter we introduced the possibility of two clients starting a new communication. Let's now look at what happen when the receiver client accepts the sender request. Let's recall the previous example. In the previous example Alice has sent a request to Bob. Let's suppose that Bob accepts that request. The server sends to both clients the public key of the other client so that they can start their authentication. Alice generates a fresh nonce, encrypts it by means of the session key she shares with the server and sends it to the server that will then forwards it to Bob encrypting it by mean of the session key that shares with Bob. Bob generates his private and public key using *Diffie-Hellman* protocol. After receiving Alice's nonce he concatenates it with a fresh nonce he has generated and his DH public key. He signs it with his private key and encrypt the result with the session key he shares with the server.

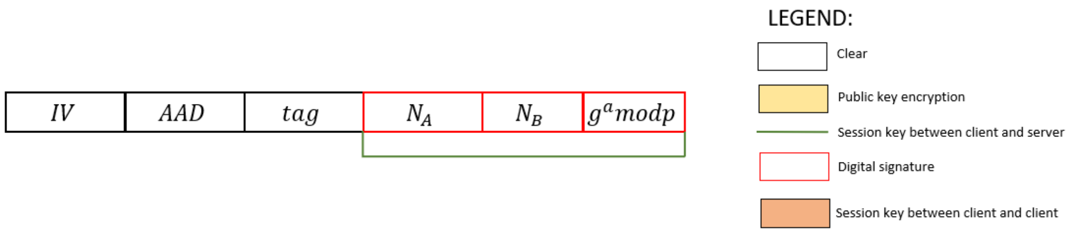
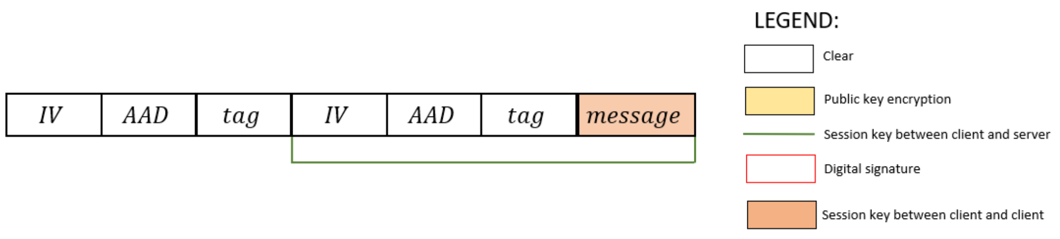


Figure 12: Client's authentication and DH key generation

Alice generates her private and public key using *Diffie-Hellman* key generation method and waits for Bob's message. After receiving it through the server she verifies Bob's signature by mean of Bob's public key and checks her nonce with the nonce Bob has sent her back to avoid replay attack and then she does the same procedure Bob has done in the message above. She concatenates Bob's nonce, her nonce and her DH public key together, she signs by means of her private key first and then she encrypts by mean of the session key she shares with the server. Then she can compute the session key shared with Bob. After receiving Alice's message, Bob verifies Alice's signature and checks the validity of his nonce and if both checks are successful he computes the session key as well. From now on Alice and Bob can communicate in a secure way by exploiting the shared session key. Alice will also encrypt the encrypted messages by mean of the session key she shares with the server because the server is still the *trusted-third-party* they use to communicate with each other and Bob will do the same. Anyway, the server will not be able to understand their communication because it does not know the session key between Alice and Bob. It can just see the messages go back and forth but it cannot understand their meanings.



IV
AAD
tag
IV
AAD
tag
message
LEGEND:
Clear
Public key encryption
Session key between client and server
Digital signature
Session key between client and client

Figure 13: Message between two clients format

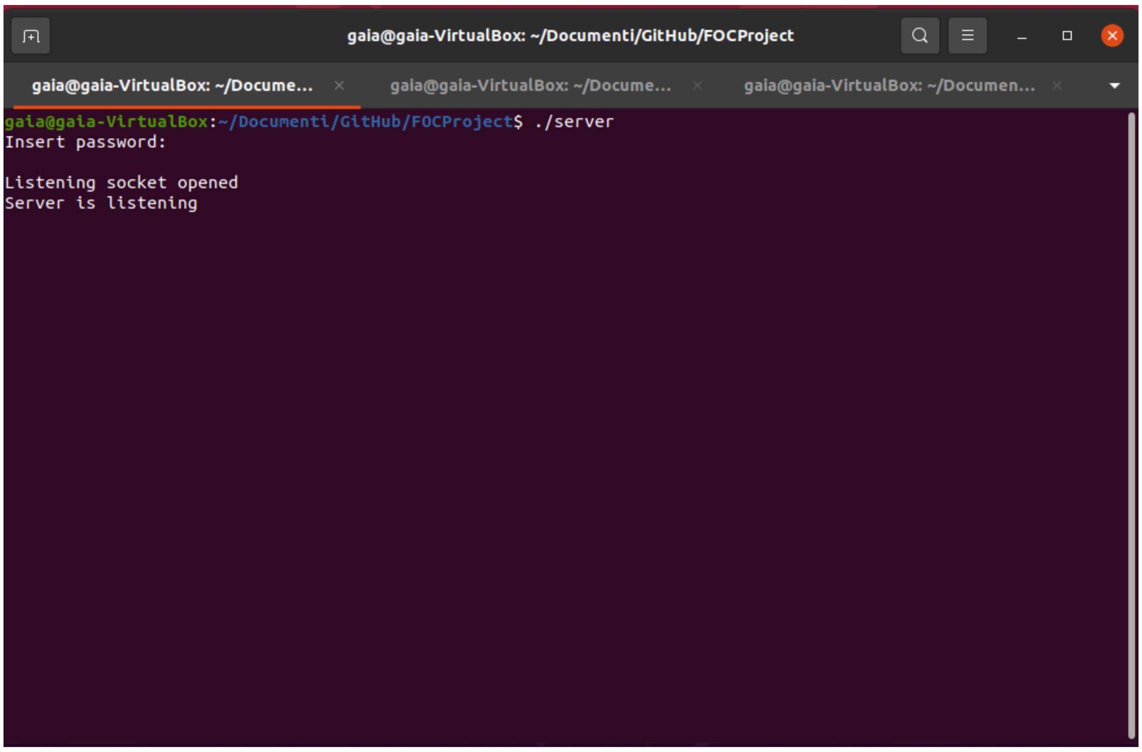
4 User Manual

Before starting executing the server and the client, the source files have to be compiled. To simplify this task, a makefile has been created:

- **make server**: to compile the server and some other support files included if not already compiled
- **make client**: to compile the client and some other support files included if not already compiled
- **make clean**: to remove server and client executable files and other object files created before

4.1 Starting the server

The server can be run using the command **./server**. After starting, the server ask you to insert a password. The password here requested is the one used to protect the server's private key. If the password entered is wrong the server will not be able to access its private key and the execution will stop with an error.


gaia@gaia-VirtualBox: ~/Documenti/GitHub/FOCProject
gaia@gaia-VirtualBox: ~/Docume...
gaia@gaia-VirtualBox: ~/Docume...
gaia@gaia-VirtualBox: ~/Documen...

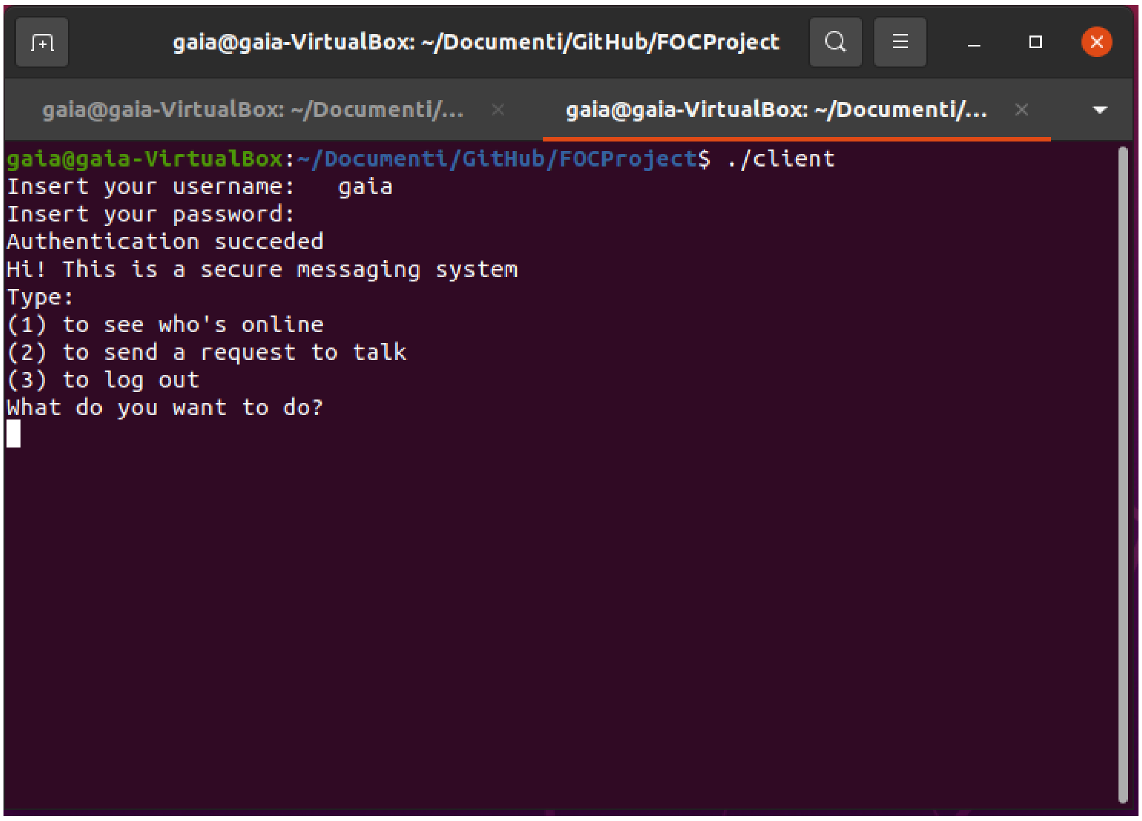
```
gaia@gaia-VirtualBox:~/Documenti/GitHub/FOCProject$ ./server
Insert password:

Listening socket opened
Server is listening
```

Figure 14: Server starts

The password entered is not shown on the screen for security reasons.

4.2 Starting the client

The client can be run using the command ./**client**. After starting, the user is asked to insert his/her password. The password here requested is the one used to protect the user's private key. If the password entered is wrong the client will not be able to access his/her private key and the execution will stop with an error.


gaia@gaia-VirtualBox: ~/Documenti/GitHub/FOCProject
gaia@gaia-VirtualBox: ~/Documenti/...
gaia@gaia-VirtualBox: ~/Documenti/...

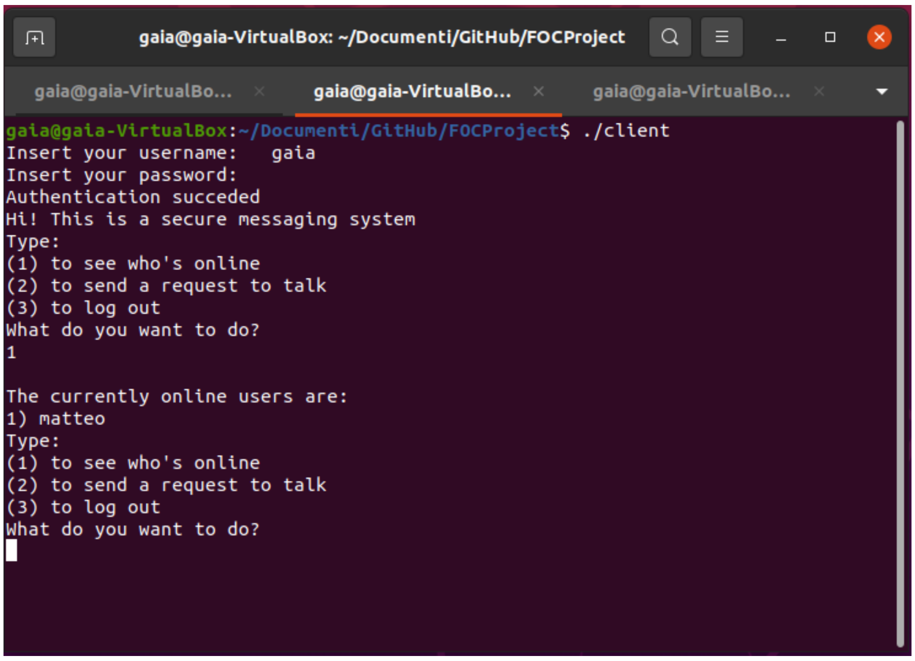
```
gaia@gaia-VirtualBox:~/Documenti/GitHub/FOCProject$ ./client
Insert your username:   gaia
Insert your password:
Authentication succeded
Hi! This is a secure messaging system
Type:
(1) to see who's online
(2) to send a request to talk
(3) to log out
What do you want to do?
```

Figure 15: Client starts

Even for the client the password entered is not shown on the screen for security reasons.

4.3 Request of the list of active users

When the user enter the input **1** it means he/she wants to see the list of the currently online users.

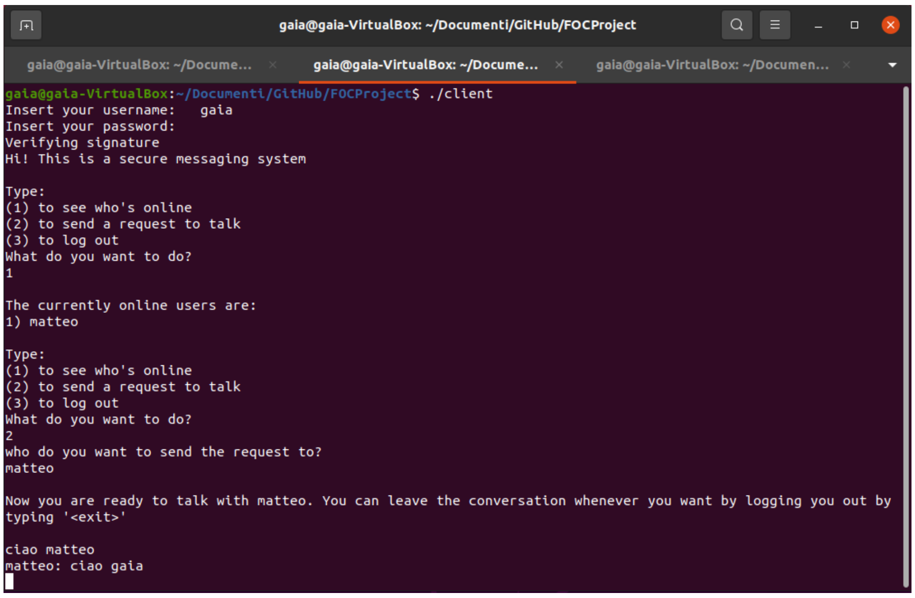


```
gaia@gaia-VirtualBox:~/Documenti/GitHub/FOCProject$ ./client
Insert your username:   gaia
Insert your password:
Authentication succeded
Hi! This is a secure messaging system
Type:
(1) to see who's online
(2) to send a request to talk
(3) to log out
What do you want to do?
1

The currently online users are:
1) matteo
Type:
(1) to see who's online
(2) to send a request to talk
(3) to log out
What do you want to do?
```

Figure 16: Client requests the list of active users

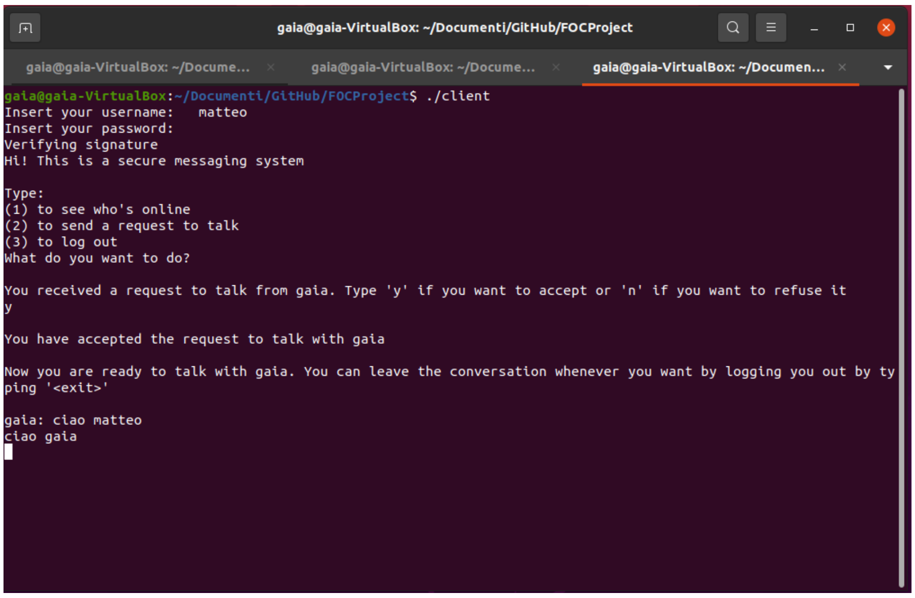
4.4 Communication between clients

When a client wants to start a new communication he types **2** and it will be then asked to specify the name of the user he wants to communicate with. When the receiver client will accept the request they can start chatting together.



```
gaia@gaia-VirtualBox:~/Documenti/GitHub/FOCProject$ ./client
Insert your username:   gaia
Insert your password:
Verifying signature
Hi! This is a secure messaging system

Type:
(1) to see who's online
(2) to send a request to talk
(3) to log out
What do you want to do?
1

The currently online users are:
1) matteo

Type:
(1) to see who's online
(2) to send a request to talk
(3) to log out
What do you want to do?
2
who do you want to send the request to?
matteo

Now you are ready to talk with matteo. You can leave the conversation whenever you want by logging you out by
typing '<exit>'

ciao matteo
matteo: ciao gaia
```

Figure 17: Communication from the sender client's point of view

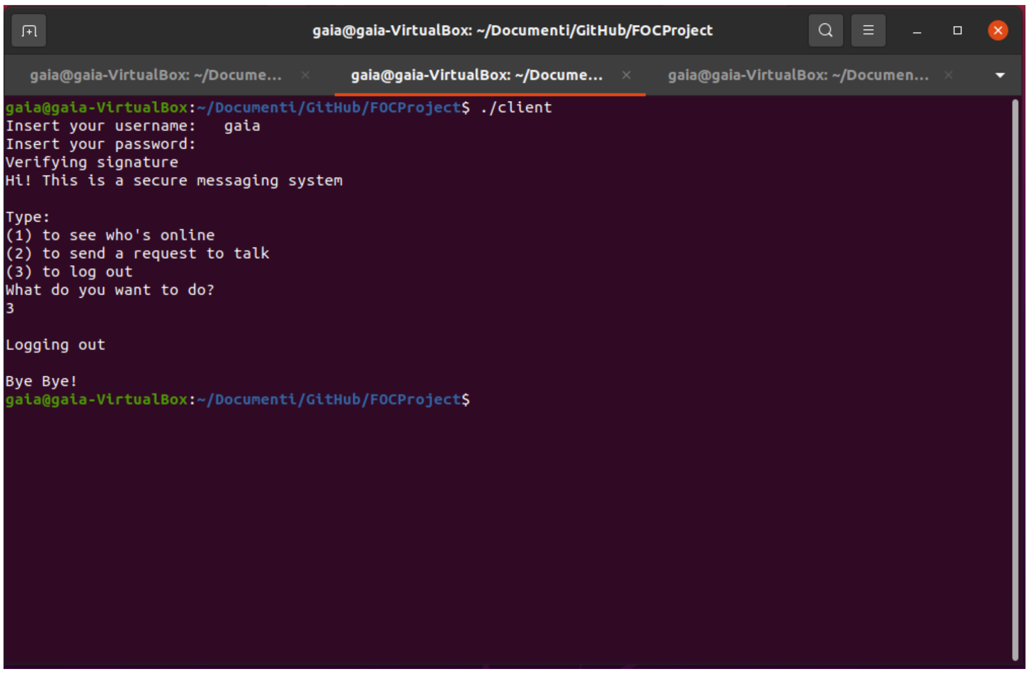

gaia@gaia-VirtualBox: ~/Documenti/GitHub/FOCProject
gaia@gaia-VirtualBox: ~/Docume...
gaia@gaia-VirtualBox: ~/Docume...
gaia@gaia-VirtualBox: ~/Documen...

```
gaia@gaia-VirtualBox:~/Documenti/GitHub/FOCProject$ ./client
Insert your username:   matteo
Insert your password:
Verifying signature
Hi! This is a secure messaging system

Type:
(1) to see who's online
(2) to send a request to talk
(3) to log out
What do you want to do?

You received a request to talk from gaia. Type 'y' if you want to accept or 'n' if you want to refuse it
y

You have accepted the request to talk with gaia

Now you are ready to talk with gaia. You can leave the conversation whenever you want by logging you out by ty
ping '<exit>'

gaia: ciao matteo
ciao gaia
```

Figure 18: Communication from the receiver client's point of view

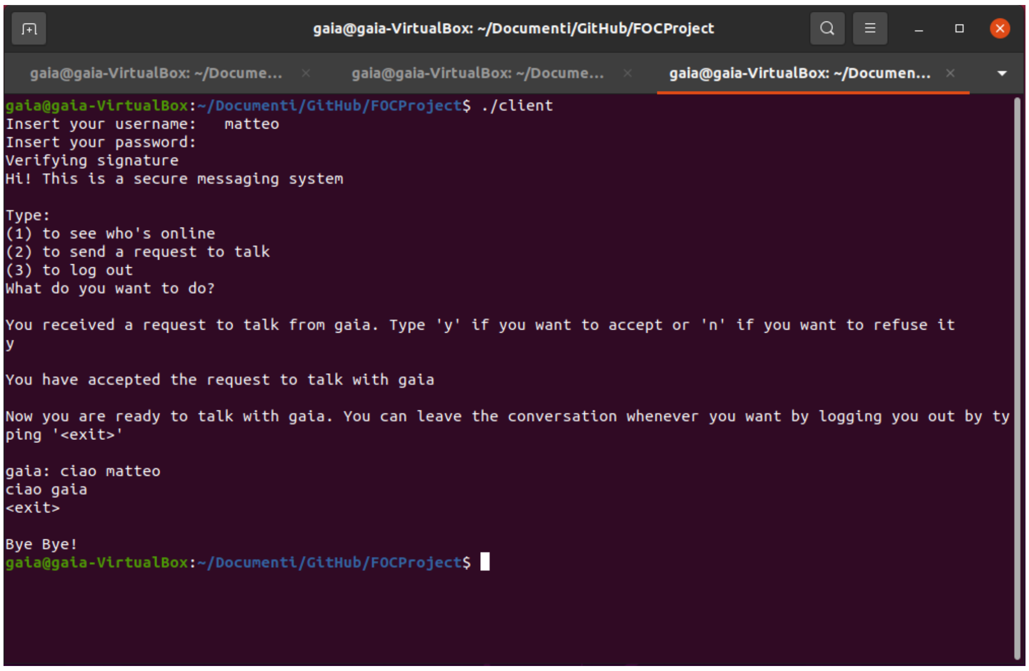
4.5 Log out

When the client wants to log out he just have to type **3** or **exit** if he/she is engaged in a communication.


gaia@gaia-VirtualBox: ~/Documenti/GitHub/FOCProject
gaia@gaia-VirtualBox: ~/Docume...
gaia@gaia-VirtualBox: ~/Docume...
gaia@gaia-VirtualBox: ~/Documen...

```
gaia@gaia-VirtualBox:~/Documenti/GitHub/FOCProject$ ./client
Insert your username:   gaia
Insert your password:
Verifying signature
Hi! This is a secure messaging system

Type:
(1) to see who's online
(2) to send a request to talk
(3) to log out
What do you want to do?
3

Logging out

Bye Bye!
gaia@gaia-VirtualBox:~/Documenti/GitHub/FOCProject$
```

Figure 19: Log out typing command **3**



gaia@gaia-VirtualBox: ~/Documenti/GitHub/FOCProject

```
gaia@gaia-VirtualBox:~/Documenti/GitHub/FOCProject$ ./client
Insert your username:   matteo
Insert your password:
Verifying signature
Hi! This is a secure messaging system

Type:
(1) to see who's online
(2) to send a request to talk
(3) to log out
What do you want to do?

You received a request to talk from gaia. Type 'y' if you want to accept or 'n' if you want to refuse it
y

You have accepted the request to talk with gaia

Now you are ready to talk with gaia. You can leave the conversation whenever you want by logging you out by ty
ping '<exit>'

gaia: ciao matteo
ciao gaia
<exit>

Bye Bye!
gaia@gaia-VirtualBox:~/Documenti/GitHub/FOCProject$ 
```

Figure 20: Log out typing **exit**