UNIVERSITY OF PISA

Master in Computer Engineering
Project for Foundation of cybersecurity

Secure Messaging Application

Author:
Tommaso Burlon

Academic year
2020/2021

Summary

1 Specification

The aim of the project is to develop an application for secure messaging. The application follows a client-server architecture. In the application every client is already registered with the server this means that the server already has the public key of every client.
When a client connect to the server an handshake protocol must be used to guarantee the authentication of both parts and the confidentiality.
When the client and the server are authenticated, the client can performs three actions: ask to the server the list of online users, ask to the server to start a new conversation with one of the active user or to log off.
When the client request to the server a new chat the server forward the request to the user and wait for the response if the other user accept the request the two clients start an handshake protocol to guarantee an end-to-end encryption, an user can have only one chat at the time and to start a new chat has to logoff from the server and start over.
Every message must be encrypted, authenticated, must guarantee perfect forward secrecy and must be protected from replay attack.

2 Handshake Protocols

The application support four different handshake protocol: the client to server, the client request to chat, the client to client and the log-off.

2.1 Client to server

This handshake starts from the client that send a message containing the ECDH (Elliptic curve Diffie-Hellman) public key and a random nonce, the server responds with his certificate signed by a Certification Authority a random nonce and the other ECDH public key, this message and the nonce of the client are signed using the private key of the server. The client verify the server certificate using the public key of the CA, extract the public key of the server, check the signature of the message and generate the shared secret so he obtains the symmetric key.
The last message comes from the client an it contains only two fields: the name of the user and a random nonce, this message is encrypted using the

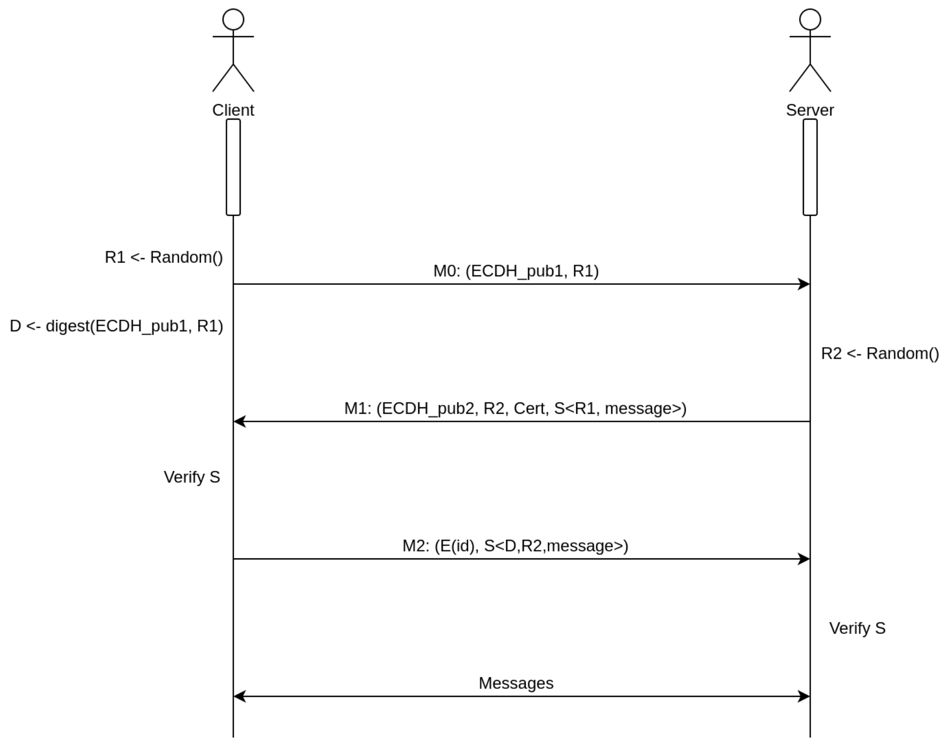
Client
Server
R1 <- Random()
M0: (ECDH_pub1, R1)
D <- digest(ECDH_pub1, R1)
R2 <- Random()
M1: (ECDH_pub2, R2, Cert, S<R1, message>)
Verify S
M2: (E(id), S<D,R2,message>)
Verify S
Messages

Figure 1: client-server handshake

symmetric key and it is signed by the client signing also the first message and the server nonce.
The only message that is not protected it is the first, but the signature of the final message guarantee the integrity of the first message.
The protocol is strong against replay attack thanks to the first two nonces, and guarantee perfect forward security.
After the handshake every message from the client to the server and from the server to the client is encrypted with the symmetric key using two counters as initialization vector, one for the messages coming from the server and one for the messages coming from the client, each message is authenticated using an HMAC.

2.2 Client request to chat

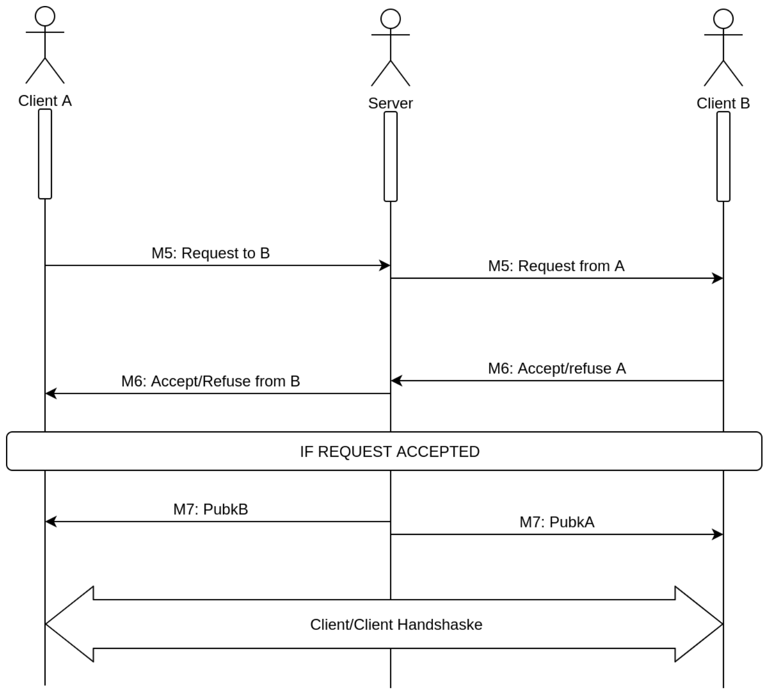

Figure 2: client-server handshake

This handshake is pretty simple: when a client A want to communicate with another client B he sends a message to the server with the username of the client B, the server forwards the message to the client B who can refuse, accept or ignore. A client can send requests to more user but when a user accept the request the communication starts and the other user cannot join. If the client B accept the request, he send a message to the server which forward the message to client A and then send to A and B the public key of the other client, after this messages the client should start a protocol to establish an end-to-end encryption and the server works only as an encrypted switch.

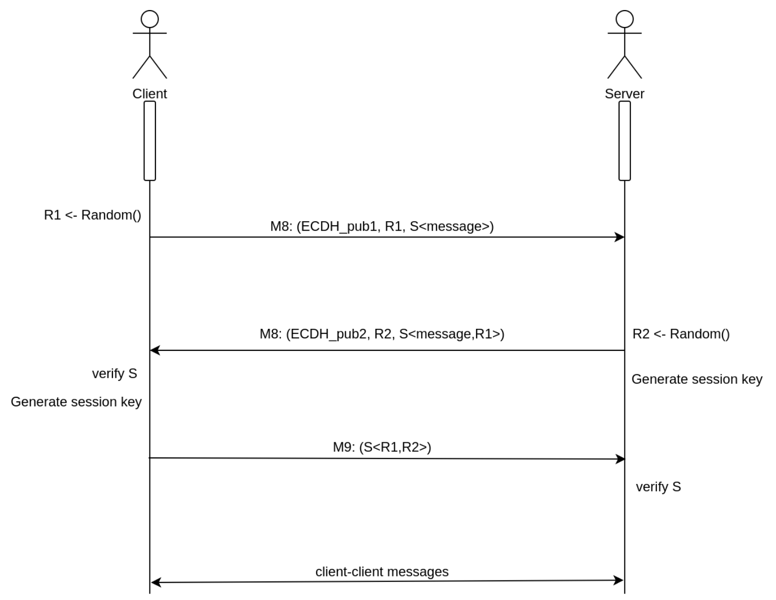

Figure 3: client-client handshake

2.3 Client to client

This handshake starts when the previous handshake end. The client A (who receive the message with the field isFirst set) send a message to the client B which contain an ECDH public key a random nonce and the signature of the client. The client B send the same kind of message signing also the nonce from the client A, the client A then answer with a simple informative message with a signature of the nonce from A and from B.

3 Software implementation

3.1 Architecture

The application follows a server-client architecture.
the server is multithreaded with 1 daemon and 1 thread for each client, the server start by loading every public key of the clients and his certificate, then start a socket to listen new connection from the clients.
when a client connect the server start a new thread to handle the requests of

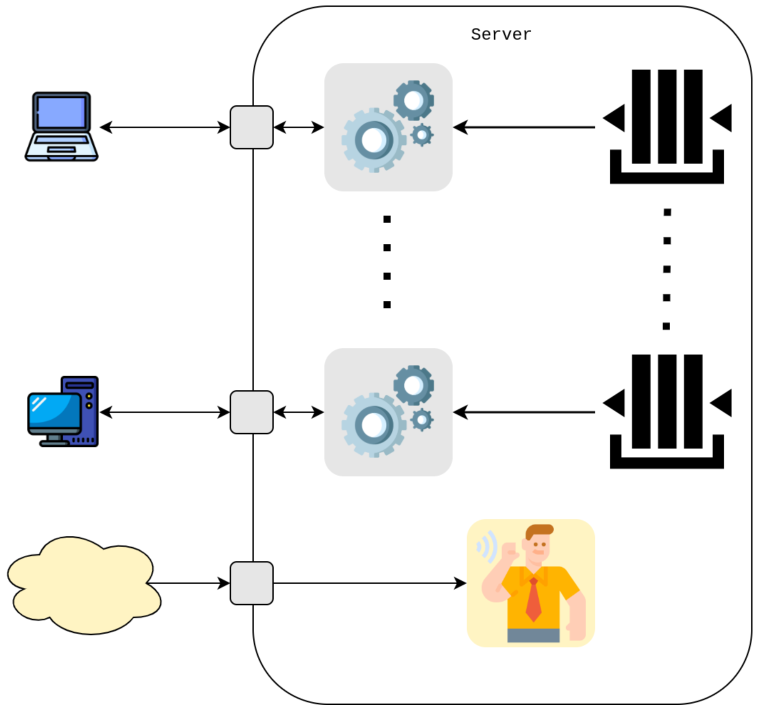

Figure 4: server architecture

the client this worker thread have also a shared queue where other threads can push messages coming from the system or from other workers.
The client is single threaded that use a poll to polling two file descriptor one for the stdin where the commands a written and one for the socket.

3.2 Cipher and Hash

The ECDH protocol for key generation use the curve `NID_X91_62_prime256v1`, the hash used to calculate the HMAC and the signature is the `sha256`, the certificate key is a `RSA 4096 bit` key, and the symmetric block cipher used is the `aes_256_ctr`. The private and public key for the user use the `brainpoolP512r1` elliptic curve.

4 Messages

4.1 client-server protocol:

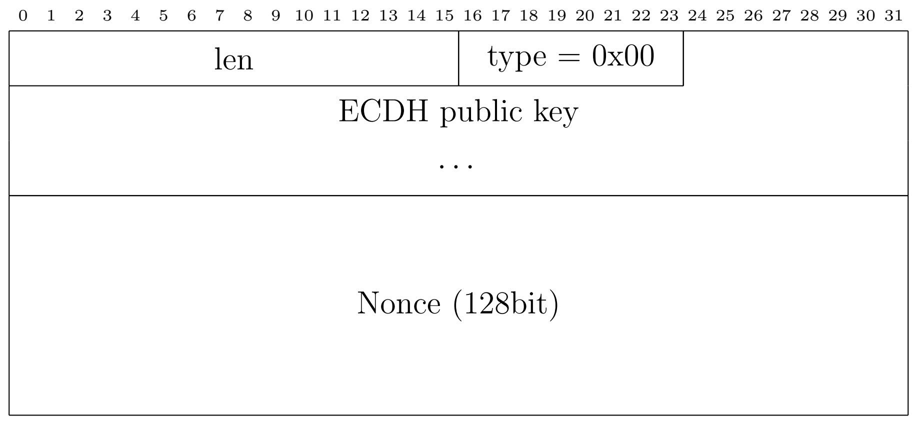

Figure 5: first message sent by the client no authentication

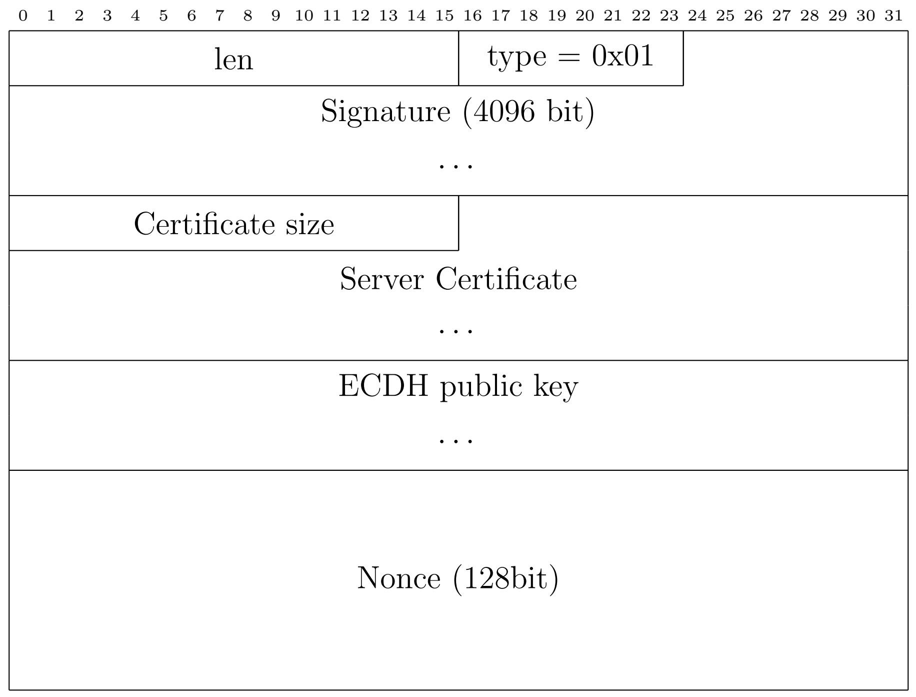

Figure 6: server certificate signed message the signature contains the client nonce

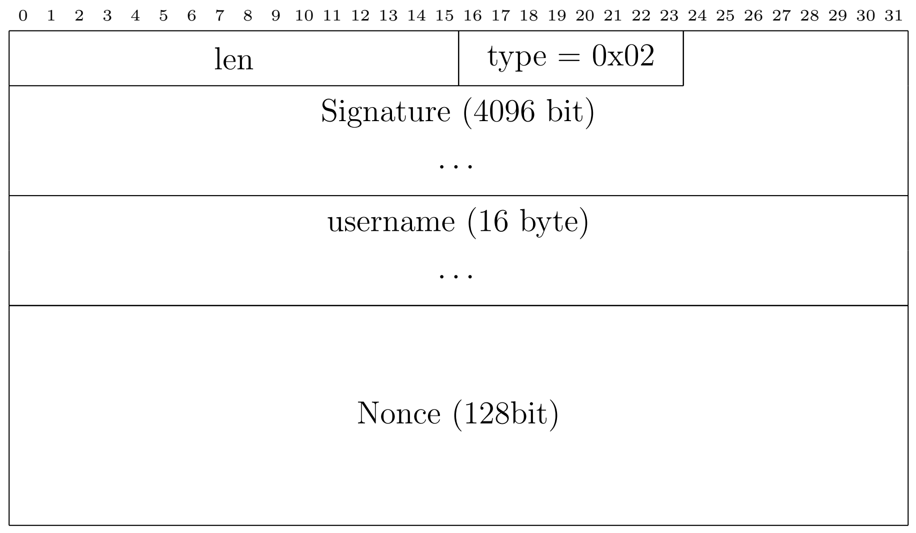

Figure 7: client authentication message, the signature authenticate also the first packet

4.2 get_list command

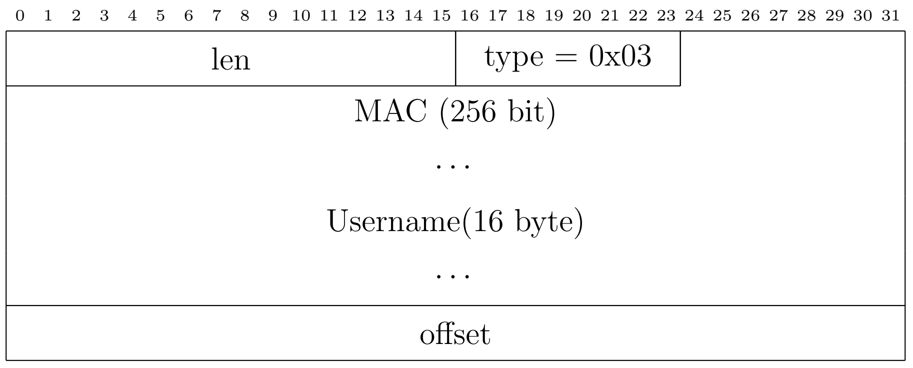

Figure 8: message sent by the client to obtain the list, the message is authenticate and encrypted

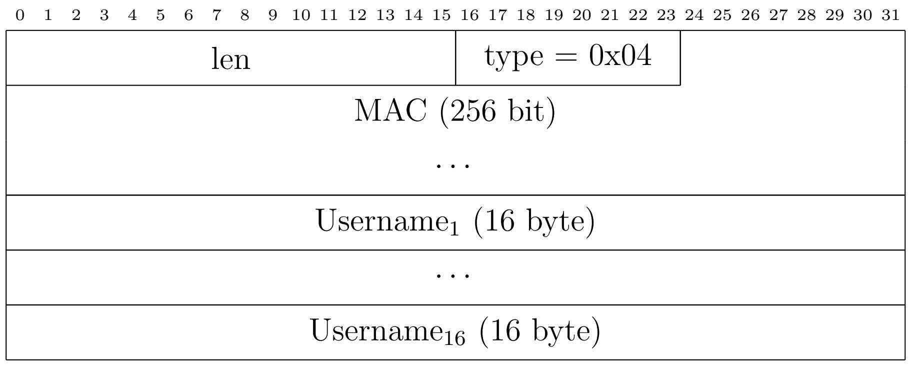

Figure 9: response of 16 online users, the message is encrypted and authenticated

4.3 Client request handshake

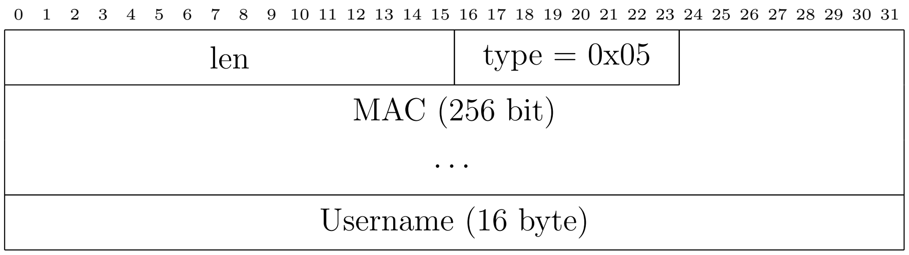

Figure 10: message to request a chat, the message is encrypted and authenticated

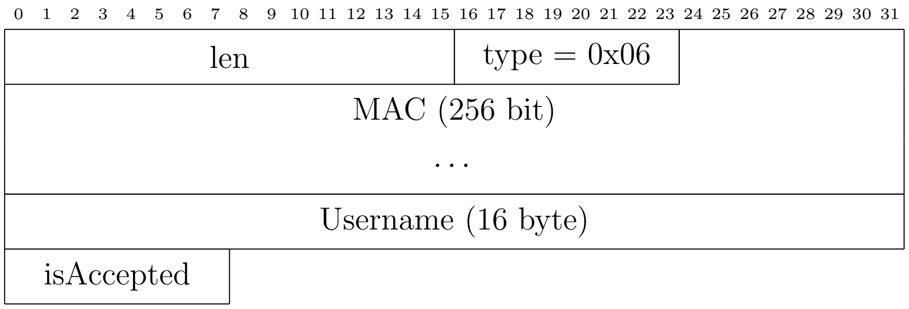

Figure 11: response of the client, the message is encrypted and authenticated

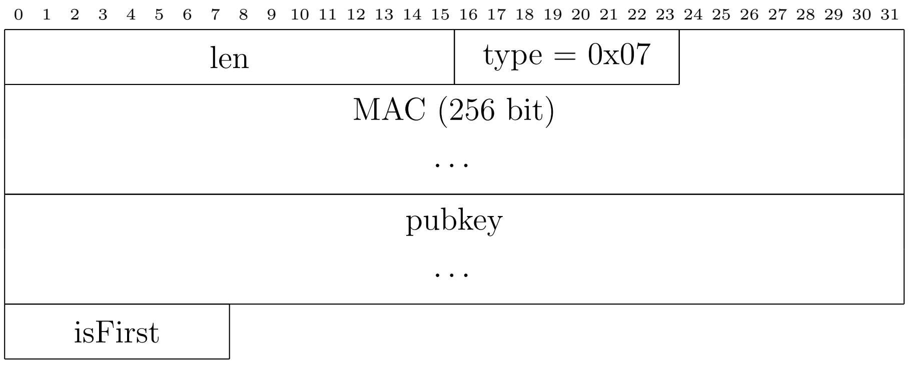

Figure 12: response of the server with the public key of the other client, the message is encrypted and authenticated

4.4 client-client handshake

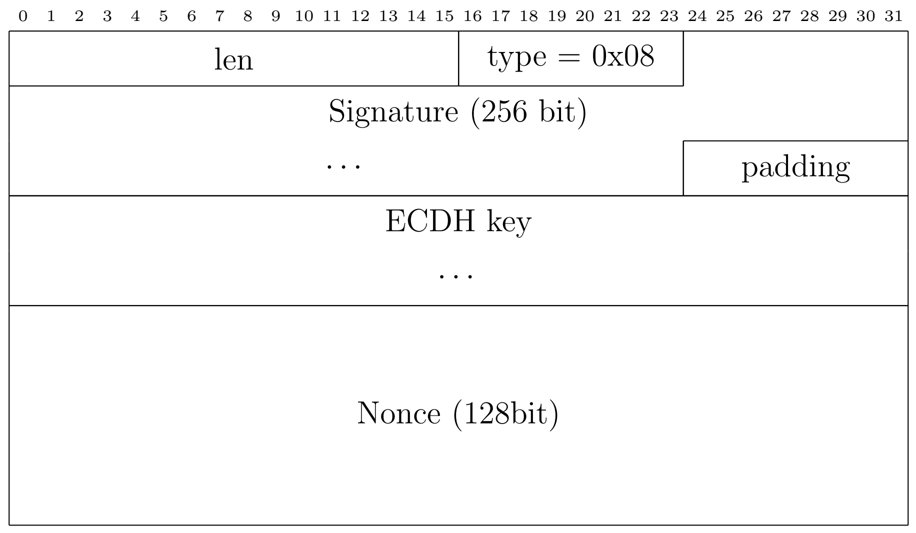

Figure 13: message sending the ECDH key, the message is encrypted by the server and authenticated by the client.

4.5 utility messages

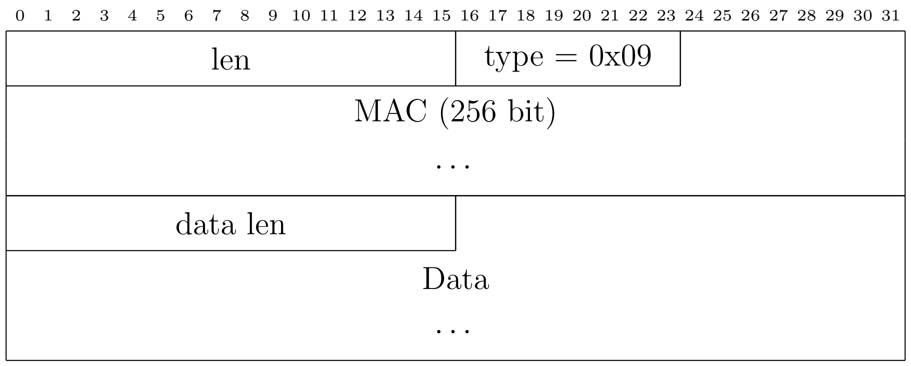

Figure 14: secure message wrapper the message is encrypted and authenticated, also the data inside the message could be encrypted and authenticated.

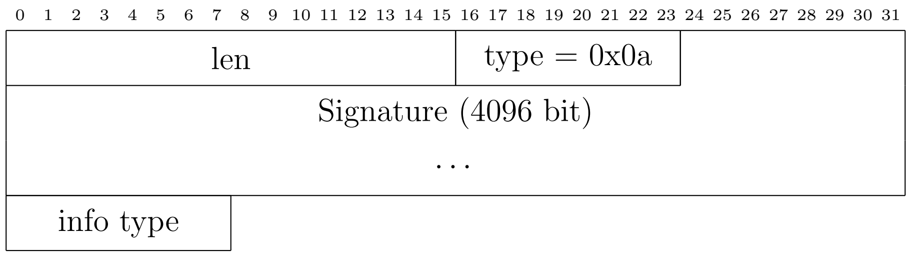

Figure 15: informative message,the message is signed but not encrypted.

5 Manual

5.1 Login

The first interface the the user see is the login form where the user has to insert his username, the password of the private key file and path to the private key.
This action has to possible outcomes: success if the key is found and password is correct, failure if the key is not found or the password is not correct. There is no control if the username and the private key correspond because only the server can associate some key with an username

```
initialize...
...Cipher and Hash function loaded
...CA cartificate loaded
login:  ALICE
password:  ******
private key path:  prvkey.pem
...  private key loaded
*************** WELCOME ALICE ***************
To start a new connection execute the command !connect
To exit execute the command !logoff
```

Figure 16: A successful login screen

5.2 Connect to server

After the login phase the user can connect to the server typing the command `!connect [ip_address] [port]` with this command it is possible to connect to a server with a specific ip address and a specific port written after the command.
The user can also log off from the application by typing `!logoff` this action is possible in every next phase of the program.

```
!connect 127.0.0.1 8080
connecting to server:  127.0.0.1:8080
...  server connected
starting server authentication...
server certificate:  /C=IT/ST=Italy/L=Pisa/O=CyberCompany
/CN=ChatServer.com
The server has been authenticated
```

Figure 17: A successful authentication with the server

5.3 Server communication

After the connection the server and the client proceed to authenticate, if the authentication is a success the user can now request to see every online members typing the command `!get_online [page_number] [user_searched]`, this command returns at most 16 online user it is possible to earch for a particular user by filling the field `user_searched`. In this phase it is also possible to request a chat to another user typing `!request [username]` this command send a request to a specific user that can reply by typing `!accept/!refuse [username]`.

5.4 Client chatting

When to users agree to start a conversation the two clients start an authentication phase and then it is possible for the two user to chat with an end-to-end encryption.
If an user want to end the conversation he has to type the `!logoff` command to logging off and the server or the command `!quit` the first command will stop the client server communication while the second one will only stop the chat.

```
new request to chat from BOB
!accept BOB
Succesful authentication, key generated.
************** CHAT WITH BOB********************
ciao bob
come va?
BOB> tutto bene alice
BOB> e tu?
```

Figure 18: example of a the client chat