UNIVERSITÀ DI PISA

Computer Engineering

Foundations of Cybersecurity

secureCom

Group Project Report

TEAM MEMBERS:
Francesco Iemma
Yuri Mazzuoli
Olgerti Xhanej

Academic Year: 2020/2021

Contents

Chapter 1

Specifications

The project consists on an application for secure communication between 2 clients through an intermediate server.
The server has to:

- authenticate clients when they connect to the server (with pre-shared public key)
- authenticate itself with a certificate
- provide the list of online clients
- relay messages from one client to another, together with chat requests and response
- provide to a client the public key of another client, in order to permit a secure communication between them

A client has to:

- authenticate the server (with the certificate)
- authenticate himself with its public key

A client can:

- print the list of online clients
- authenticate another client via its public key obtained from the server
- request to chat with another client
- answer to a chat request (if not already involved in another chat)
- when in a chat, exchange text messages with another client or close the chat

Chapter 2

Design choices

2.1 Client Server Handshake

In order to authenticate themselves and establish a session key to securely communicate, a client and the server have to exchange handshake messages. We implement this protocol to provide perfect forward secrecy, starting from the pre-shared cryptographic quantities (public keys and certificates):

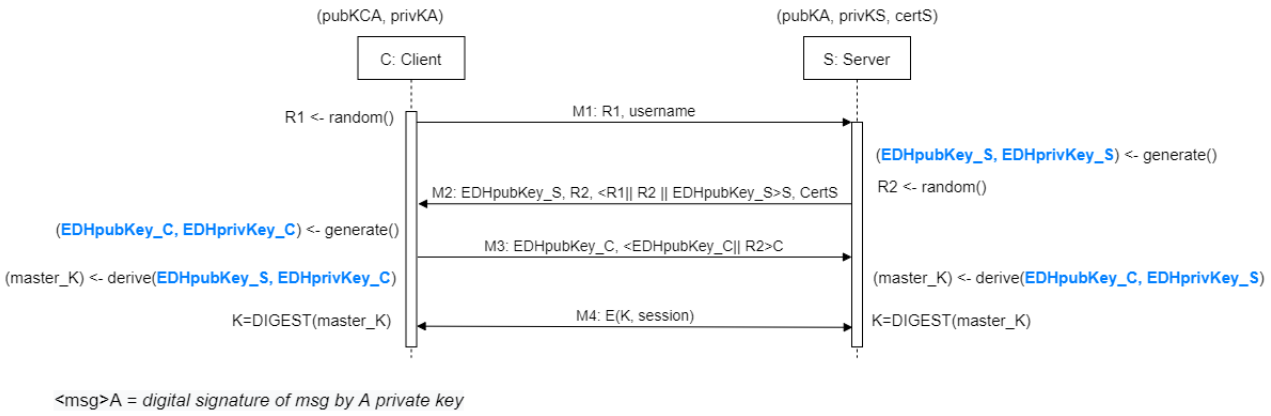

Figure 2.1: Client Server Handshake Protocol Schema

This handshake is a custom implementation of an ephemeral Diffie-Hellman Key Exchange, in which we ensure protection against the man in the middle attack with random nuances (R1 and R2). The client is able to authenticate the server via it's certificate, signed by a trusted certification authority (the client is distributed along with CA's self-signed certificate); the server has a built-in list of all client's public keys. DH's private keys are deleted after the handshake and the session key is generated by a digest of the shared secret: in this way we provide security against a possible future disclosure of one of the long term private keys.
To better understand this document is important to underline that in the following we will refer to Ephemeral Diffie-Hellman public/private key also as Diffie-Hellman public/private key implying that they are ephemeral.

2.2 Chat Request

With the client-server handshake we build a secure tunnel between each client and the server. Using this tunnel every client can execute command on the server in a secure way; the most important (and complex) command is a chat request, of which we provide a scheme:

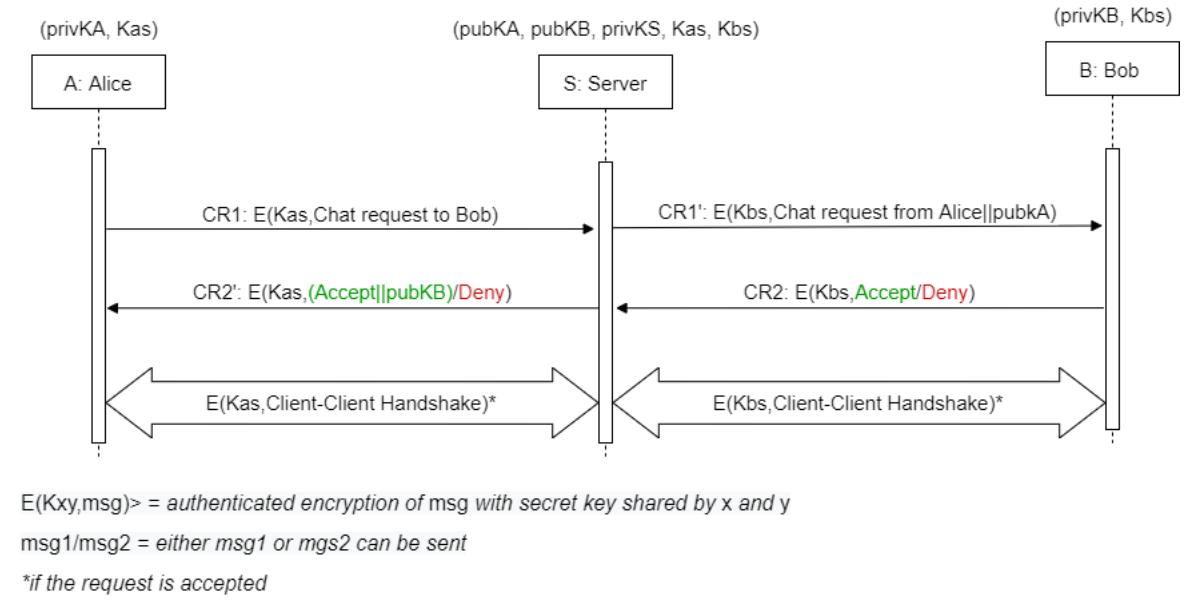

Figure 2.2: Chat Request Protocol Schema

In this case the security is based on the assumption that the server is obliged to communicate correct public keys. This assumption is fulfilled since the server is considered honest-but-curious.

2.3 Client to Client handshake

In order to guarantee a secure communication of the clients against the server, we perform an ephemeral Diffie-Hellman Key Exchange before starting the chat. In this case the two parties already know each other public keys, because the server provided them.

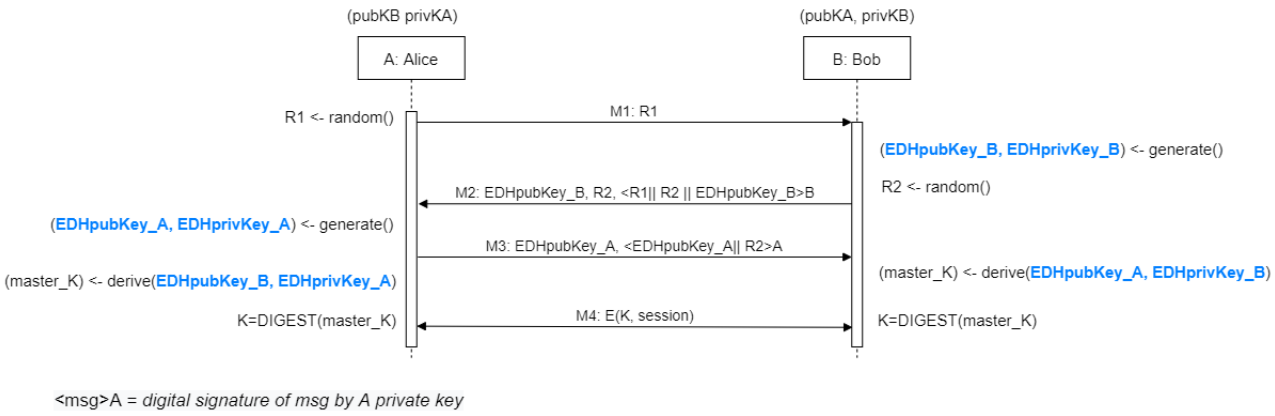

Figure 2.3: Client Client Handshake Protocol Schema

The server in not represented because it only re-transmit messages from a client to the other without changing anything; if the server try to implement a "man in the middle" attack he will only obtain a denial of service because the protocol is protected by private key signatures. Also

in this case, DH keys are discarded after the handshake, and future messages are numbered against reply attacks.

2.4 Session Ending

In order to end a chat session, one of the two users may send a `STOP CHAT` command to the server; this command is forwarded to the other client, who will close the chat session by his side; in order to disconnect from the server, the user have to send the `EXIT` command. In case one client stops without sending the `EXIT` command, the server will notice it and will proceed to close the session and (if necessary) relative chat session.

Chapter 3

Implementation

3.1 Software Architecture

From an implementation viewpoint, the client program has to communicate with the user and the server; in the server program instead, messages have to pass from one process to another in order to be delivered to the recipient client. In the figure 3.1 we can see the implementation scheme.

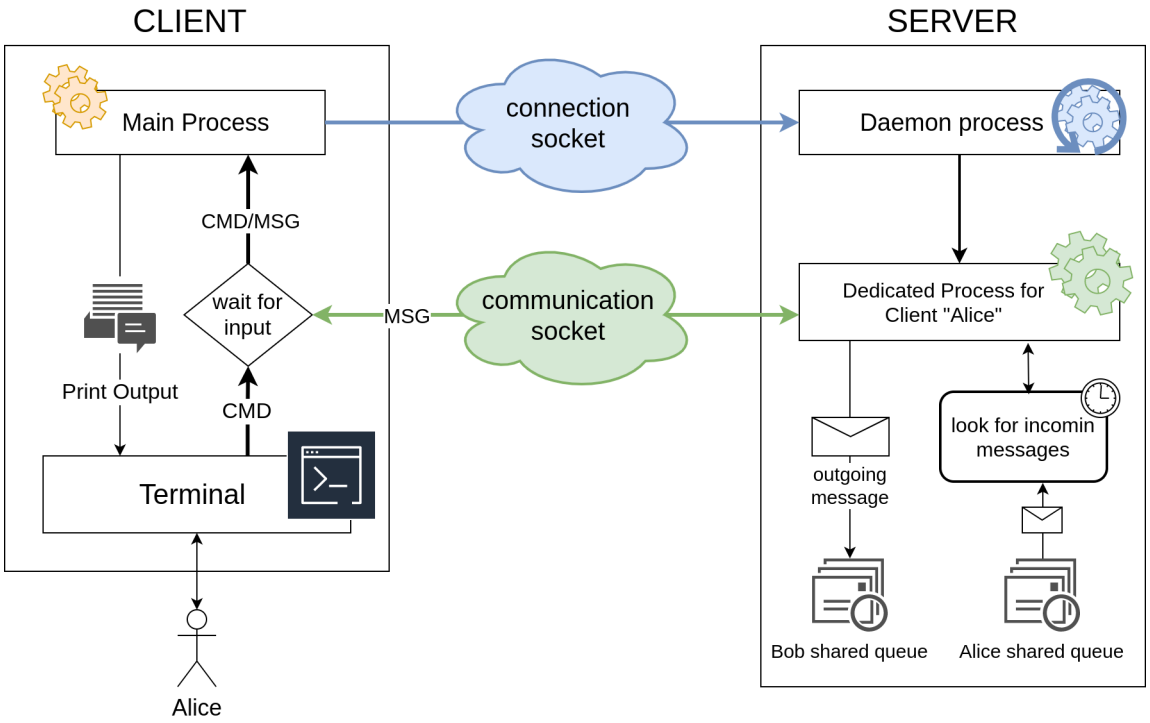

Figure 3.1: Implementation scheme

At the beginning the client will connect to a standard server port which is in listening state; then the communication is commuted to another port, with a dedicated server process. At the end of the secure handshake, the process will be dedicated to the authenticated user who is connected. This server process is able to read from his dedicated queue of incoming messages, and relay messages to others client via their queue; at constant intervals a timer will read all the messages from the incoming queue and it will forward them to the client. When this behavior is unwanted, the relative interrupt is disabled.
Using the system call `select()` the client is able to listen from multiple sources of input, in this case the sources are: communication socket and terminal. In this way the client program

is able, for example, to automatically refuse a chat request when the user is already chatting or it is in the authentication phase with another client.

3.2 Algorithms and Protocols

We briefly describe the cipher suite we choose to use in order to guarantee security requirements. Ciphers are the same for Client-Server and Client-Client communication, specific message formats are reported In Chapter 4.

3.2.1 Public Key Authentication

Long Term Keys The Certification Authority (CA) has a `Public Key`, included in a self-signed certificate; the corresponding private key is embedded in the program for certificate generation, SimpleAuthority [1]. CA's certificates and Revocation List (CRL) are exported and distributed with client executable. The server certificate, which is signed by CA, contains the `Public Key` of the server; it is stored server side, and provided to the client during the handshake. Clients Public Keys are also stored in the server, while only the client hold its Private Key. All keys are `RSA Key Pairs with 2048-bits Public Keys` and are stored in `.pem` format, as certificates and CRLs are.

Short Term Keys Handshakes protocols are performed with the Ephemeral Diffie-Hellman Key Exchange. We choose to use the Elliptic Curve implementation because is very efficient (in term of performance vs security strength); we use `NID_X9_62_prime256v1` standard parameters that ensure 128-bits security strength with a 256-bits curve. We use the `SHA-256` digest of the `shared secret` as the session key. To ensure freshness in the challenge-response scheme we use `2 bytes nonces`.

3.2.2 Authenticated Encryption

From the handshake we obtain a 256-bits symmetric key which is used in `AES-256 GCM` authenticated encryption protocol. We use a `16 bytes` tag for authentication of ciphertext and clear fields in the header. GCM scheme ensure a ciphertext size equal to the plaintext size; this makes programming easier, maintaining an optimal resistance against all known attacks. Messages exchanged in sessions are numbered, starting from 1, to a maximum of $2^{32} - 1$ (`0xFFFFFFFF`), which is the maximum integer representable on 32 bits; this will make possible to identify a message reply, done by an attacker. The session is automatically closed when a message with number `0xFFFFFFFF` is received. Clients maintains two counter for the communication with the server: one for the next message to send and another for the next message to receive; the same thing is done for the communication with another client (to prevent the server to reply a message); also the server has to implement this behavior in sessions with clients.

Note on Chat message encryption

Chat message are encrypted twice, one time with the Client-Client session key and the second one with Client-Server session key; this will not double the BIT strength of the cipher against a brute force attack, because of the `meet in the middle` attack. At the same time, this fact makes the system more secure against a password recover attack; in case, for whatever reason, a session key between clients is discovered by an attacker, this will not be enough for read private messages, because the attacker have to discover also the session key between a client and the server.

[1] https://simpleauthority.com

Chapter 4

Messages Format

4.1 Handshake

Client Server Handshake

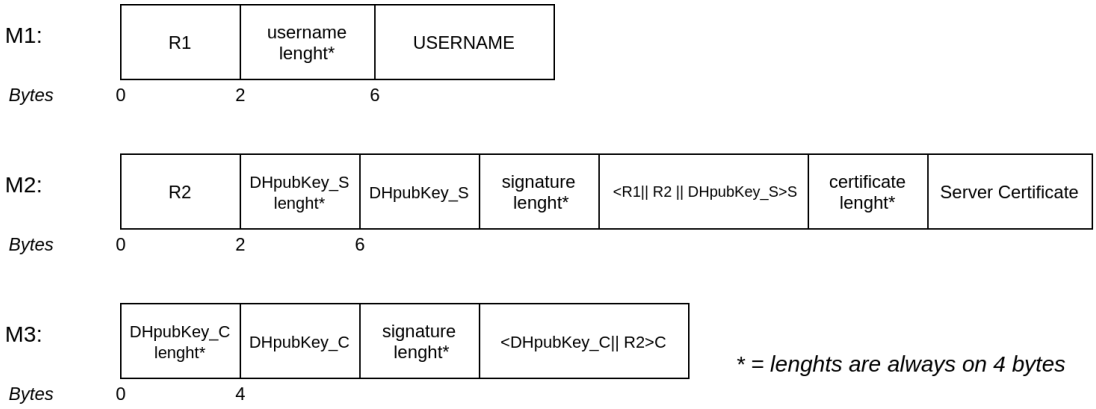

Figure 4.1: Client Server Handshake Message Format

Headers

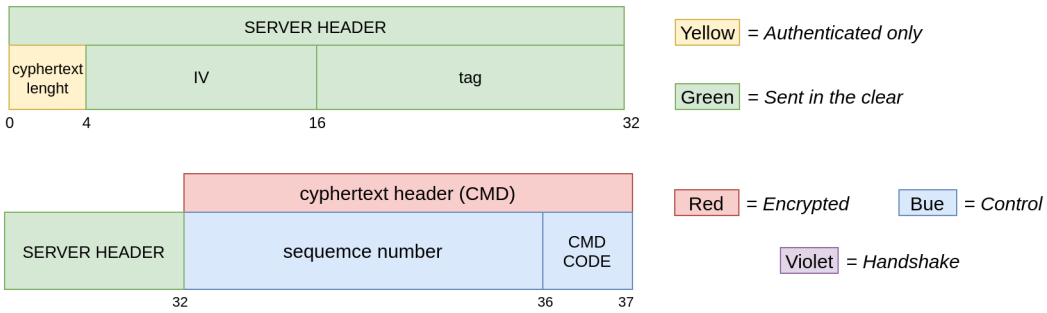

Figure 4.2: Client Server Header Format

Client Client Handshake

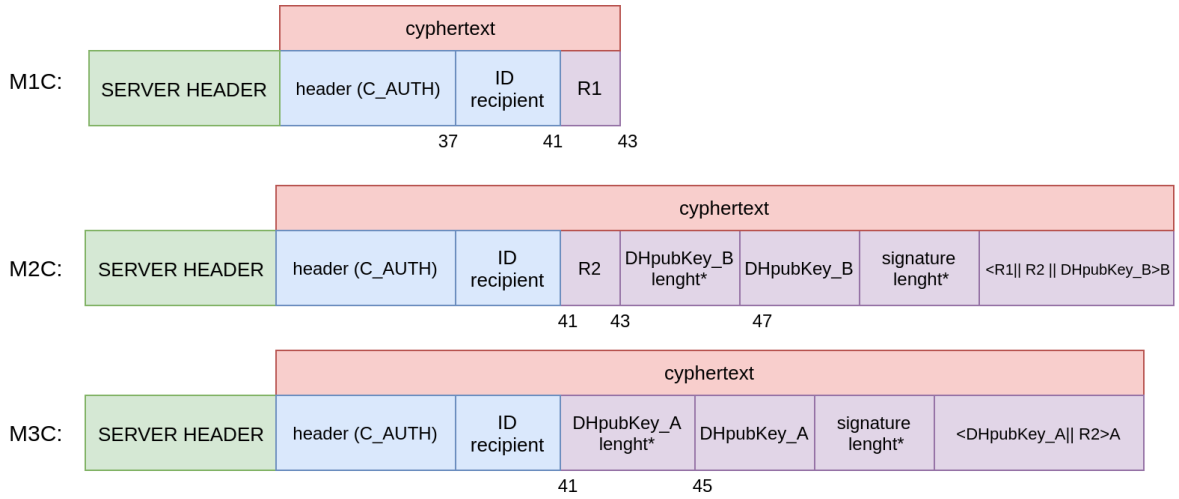

Figure 4.3: Client Client Handshake Message Format

4.2 Commands

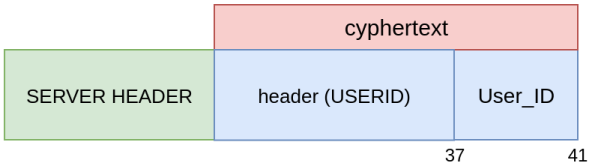

Figure 4.4: First message of every Session: server communicate client userID

Client Online List Request and Answer

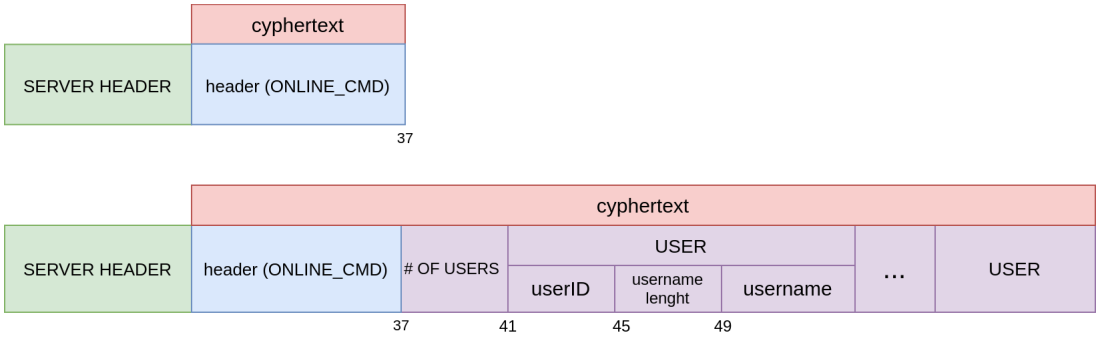

Figure 4.5: Client Online Message Format

Other commands

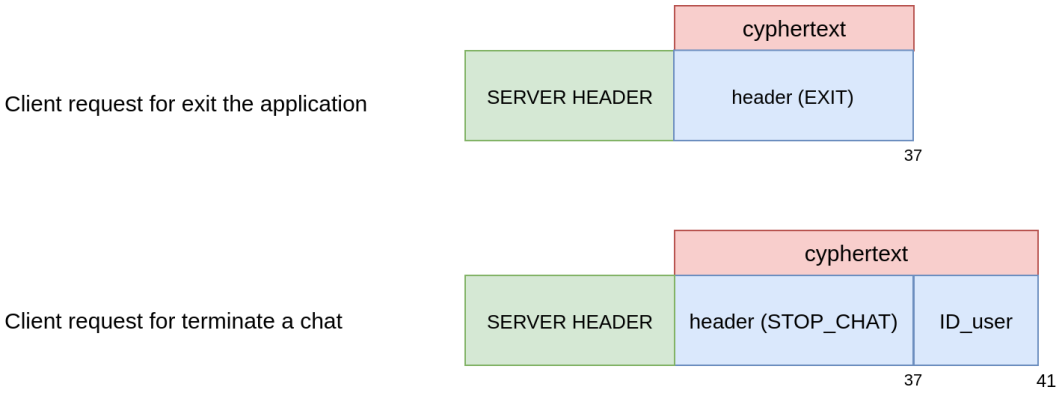

Figure 4.6: Other commands Format

Chat Request and Answers

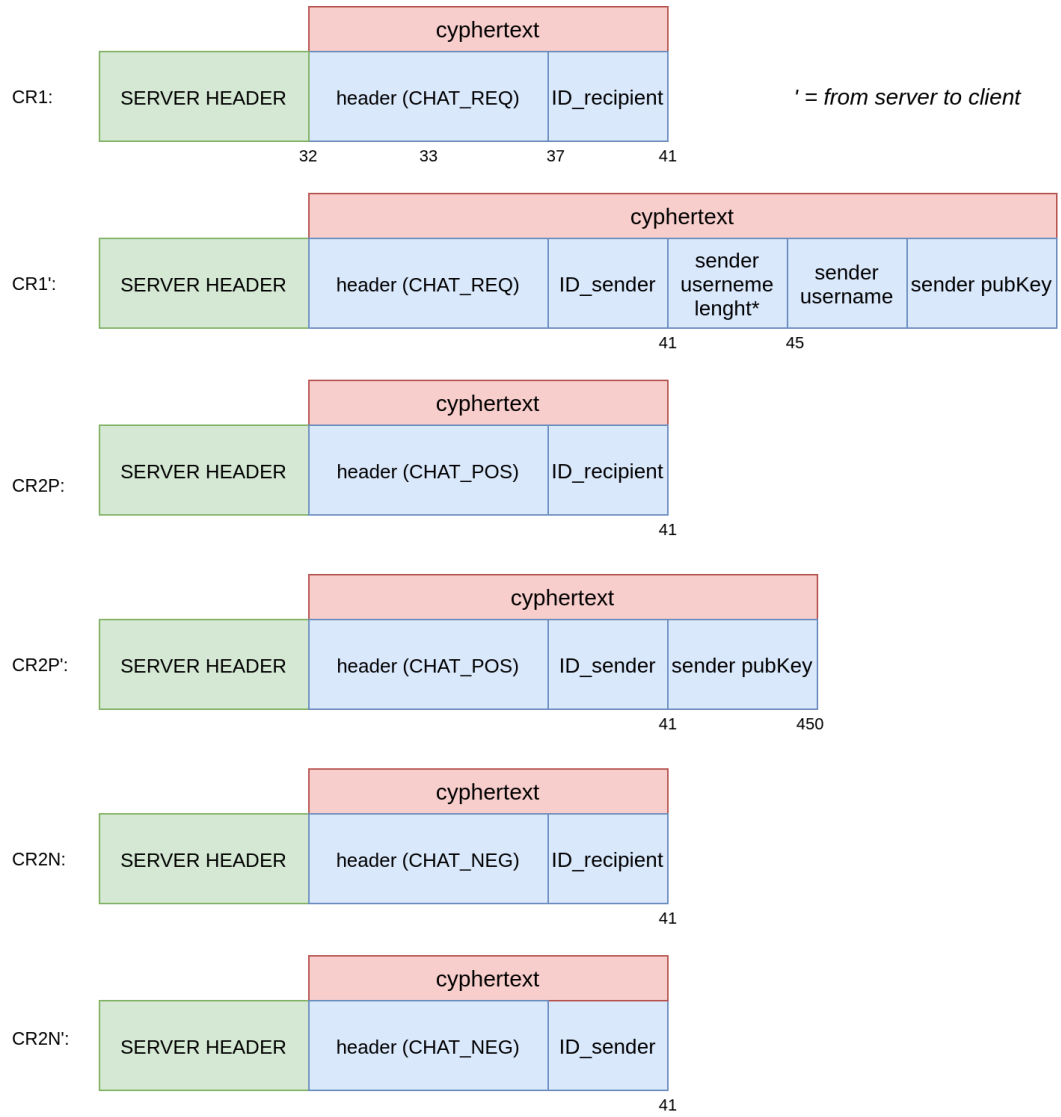

Figure 4.7: Chat Request Message Format

4.3 Chat

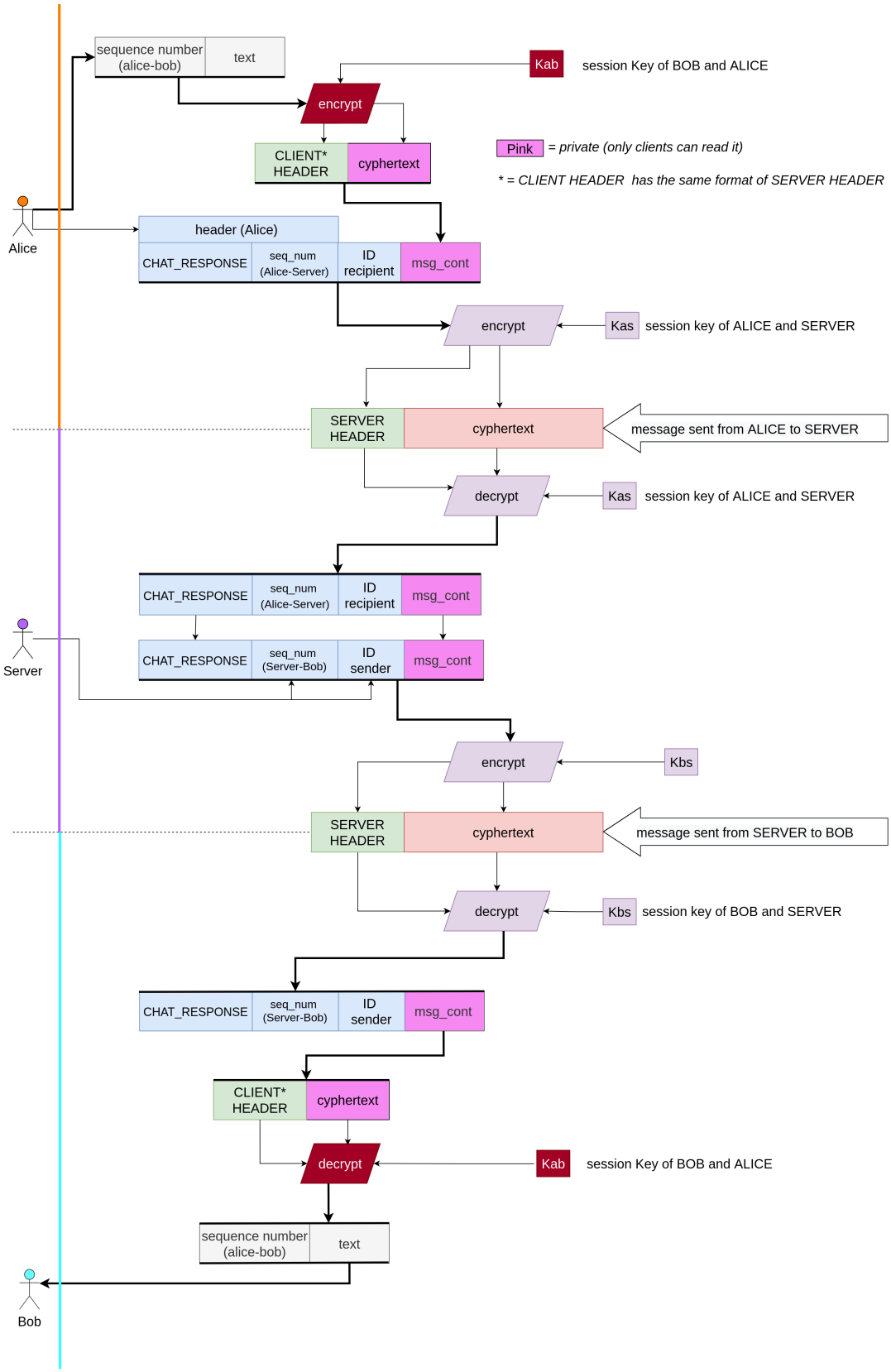

Figure 4.8: Client Client Handshake Protocol Schema

Chapter 5

User Manual

Server

The server is a daemon, so it has to be started and it will be always running. At server startup a password a `sudo` password is required in order to properly set kernel variables for the message queue; than another password is required, this time to decrypt and use the private key, which is stored in the `SecureCom_prvkey.pem` file.

5.1 Login

At client startup the user need to insert his/her username and the password to decrypt and use his private key:

```
**************************************************************************
                          SECURE COMMUNICATION
**************************************************************************
!exit       Close the application
!help       See all the possible commands
--------------------------------------------------------------------------
Who are you?
> alice
Enter PEM pass phrase:
--- AUTHENTICATION DONE ---
HELLO alice
```

At the end of this phase client and server have correctly authenticated each others and can now communicate using a session key.

5.2 Chat

Users list

With this command a client can retrieve the list of online users:

```
!users_online

**** USER LIST ****
  ID   Username
  0    alice
  1    bob
******************
```

Chat request

With the chat command, a client can request to chat with another one who is online:

```
!chat

**************************************************
Write the userID of the user that you want to contact
> 0
Wait for user's response and authentication ....
Enter PEM pass phrase:
AUTHENTICATION WITH alice SUCCESFULLY EXECUTED
```

Chat response

when a chat requests is received, the user is asked to accept ot deny it; in case of positive answer, handshake is performed (password is required at both sides).

```
**********************************************************
Do you want to chat with bob with user id 1 ? (y/n)
y
Wait for authentication ...
Enter PEM pass phrase:
Wait ...
AUTHENTICATION WITH bob SUCCESFULLY EXECUTED
```

Messages

In the chat, text messages can be sent and received; received messages are printed on the right.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                            CHAT
All the commands are ignored in this section except for !stop_chat
Send a message to bob
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

this is a secure message
                                     bob -> also this
!stop_chat
                     +++ Chat terminated +++
```