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 consist on an application for secure communication between 2 clients through an intermediate server.
The server have to:

- authenticate clients on connecting to it (with pre-shared public key)
- authenticate ifself with a certificate
- provide the list of online clients
- relay messsages 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 have to:

- authenticate the server on connecting to it (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 choises

2.1 Client Server Handshake

In order to authenticate themselvse and enstablish 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):

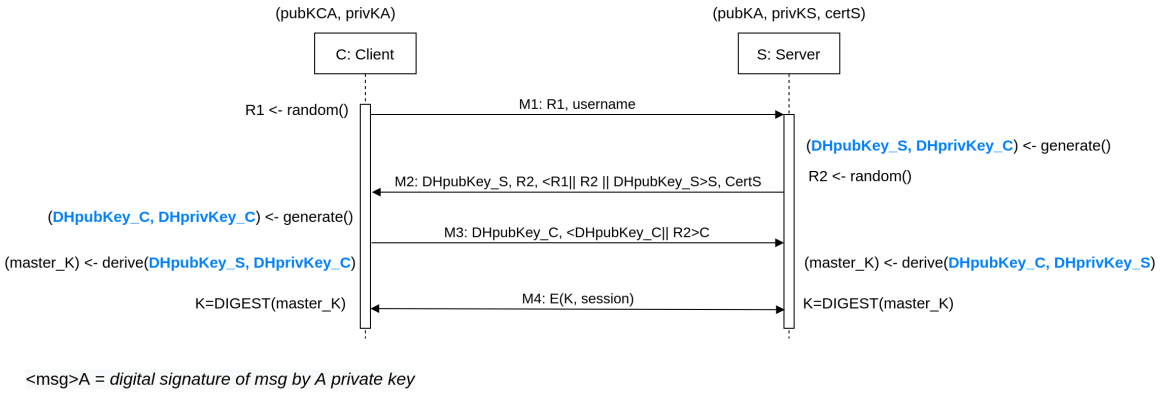

Figure 2.1: Client Server Handshake Protocol Schema

This handshake is a custom implementation of an ephimeral 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 have a built-in list of all client's public keys. DH's private keys are deleted after the handshake and the key is generated by a digest of the shared seceret: in this way we provide security against a future disclousure of one of the long term private keys.

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:

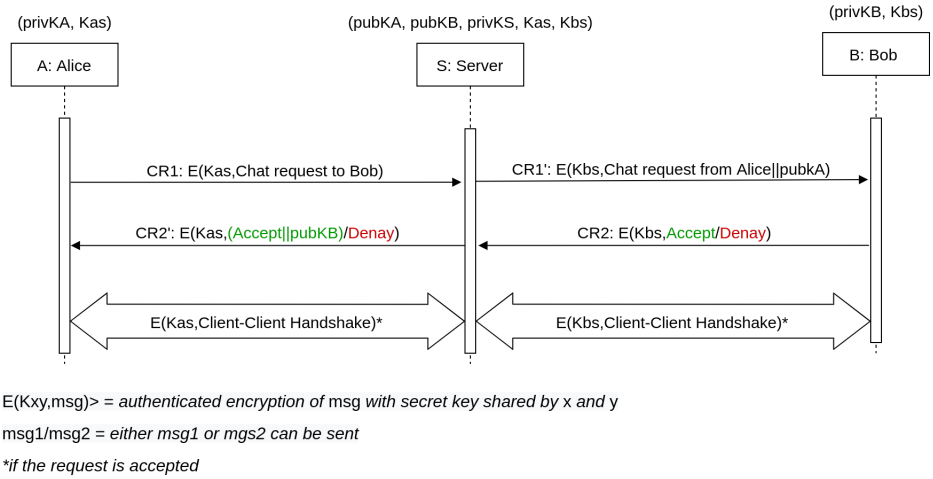

Figure 2.2: Chat Request Protocol Schema

The server is obliged to communicate correct public keys.

2.3 Client to Client handshake

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

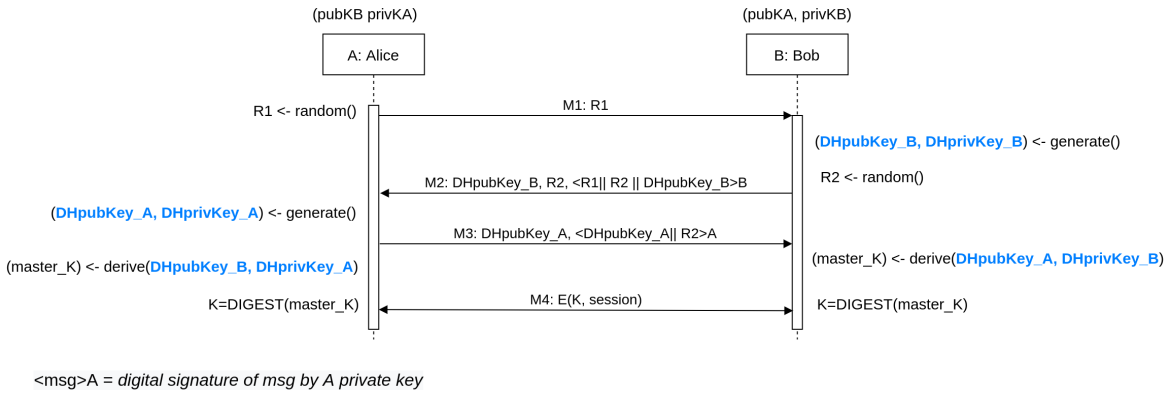

Figure 2.3: Client Client Handshake Protocol Schema

The server in not represented because it only retransmit 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 discared after the handshake, and future messages are numbered againts reply attacks.

Chapter 3

Implemetation

3.1 Software Architecture

From an implementation prospective, the client program has to communicate with user and server; in the server program instead, messages have to pass from one process to another in order to be delivered to the recipient client.

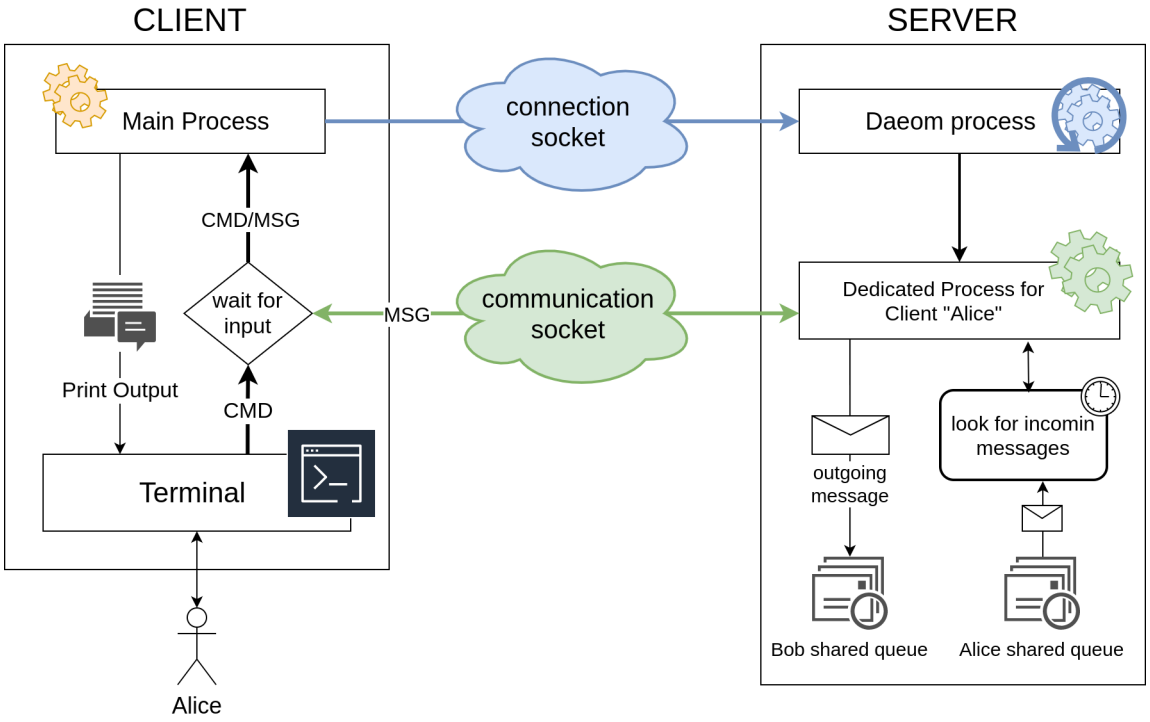

Figure 3.1: Implementation scheme

At the beginning the client will connect to a standard server port which is in listening state; than 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 incomin queue and it will forward them to the client. When this behaveour is unwanted, the relative interrupt is disabled. Using the system call `select()` the client is able to listen from multiple source of input, in this case 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.

3.2 Alghoritms and Protocols

We briefly describe the cypher suite we choose to use in order to guarantee security specifications. Cyphers are the same for Cient-Server and Cient-Cient communication, specific message formats are reported In Capter 4.

3.2.1 Public Key Authentication

Long Term Keys The Cerification Authority (CA) have a `Publick Key`, included in a self-signed certificate; the corresponding private key is embedded in the program for certificate generation, SimpleAuthority [1]. CA's cerficiate and Revocation List (CRL) are exported and distributed with client executable. The server cerfificate, which is signed by CA, contains the `Publick 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 Ephimeral Diffie-Hellman Key Exchange. We choose to use the Elliptic Curve implenetation because is very efficent (in term of performance vs security strenght); we use `NID_X9_62_prime256v1` standard parameters that ensure 128-bits secuiry strenght 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 `16 bytes nonces`.

3.2.2 Authenticated Encryption

From the handshake we obtain a 256-bits simmetric key which is used in `AES-256 GCM` authenticated encryprion protocol. We use a `16 bytes` tag for authentication of cyphertext and clear fields in the header. GCM scheme ensure a cyphertext size equal to the plaintext size; this makes programming easier, mantaining 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 my an attacker. The session is automatically closed when a message with number `0xFFFFFFFF` is recived. Clients mantains 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 behaveour in sessions with clients.

Note on Chat message encryprion

Chat message are encrypted twice, one time with the and Cient-Cient session key and the second one with Cient-Server session key; this will not double the BIT strenght of the cypher against a brute force attack, because of the `meet in the middle` attack. At the same time, this fact makes the system more secure agaist a password recover attack; in case, for whatever reason, a session key between clients is discover by an attacker, this will not be enaugh 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

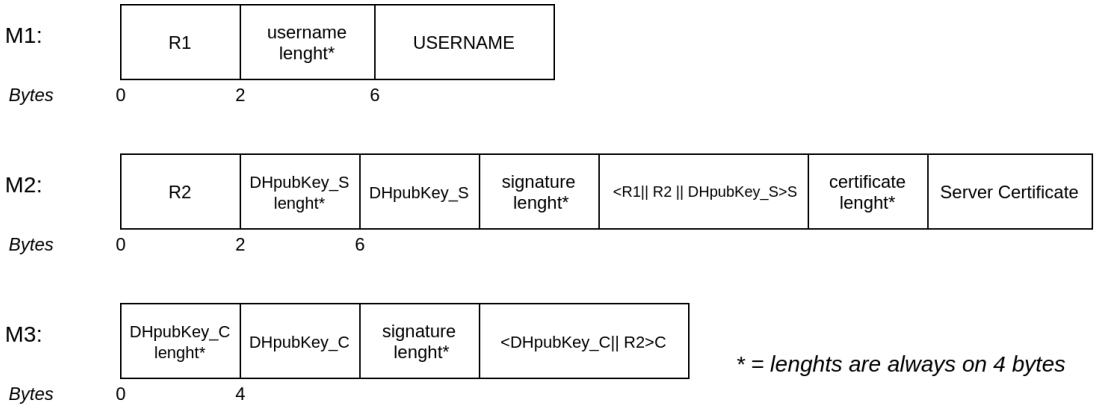

Figure 4.1: Client Server Handshake Message Format

Headers

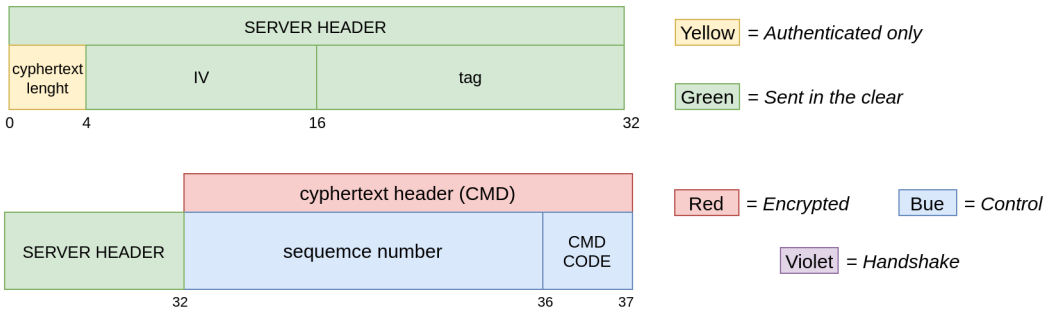

Figure 4.2: Client Server Header Format

Client Client Handshake

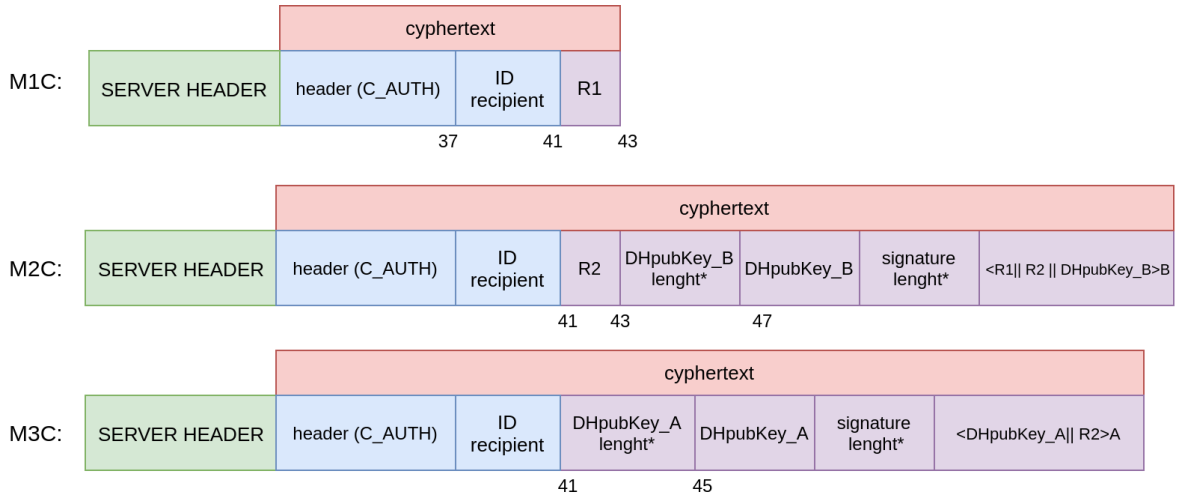

Figure 4.3: Client Client Handshake Message Format

4.2 Commands

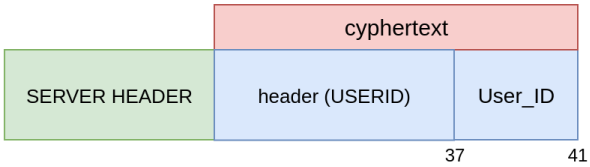

Figure 4.4: First message of every Session: server communicate cliet userID

Client Online List Request and Answer

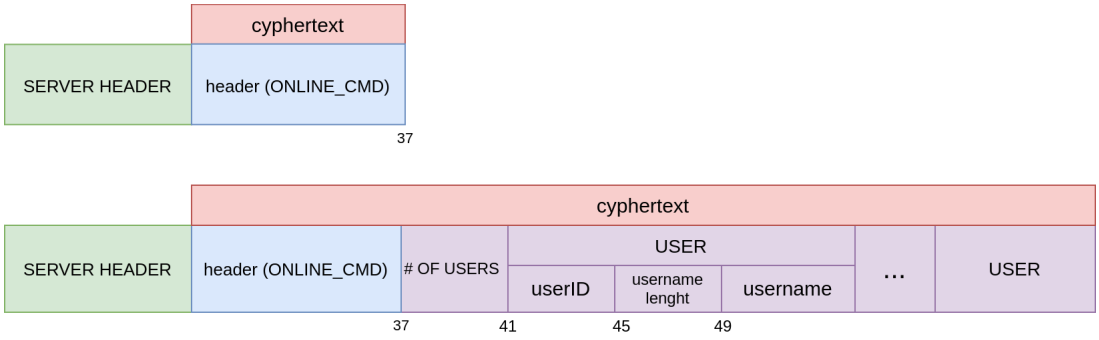

Figure 4.5: Client Online Message Format

Other commands

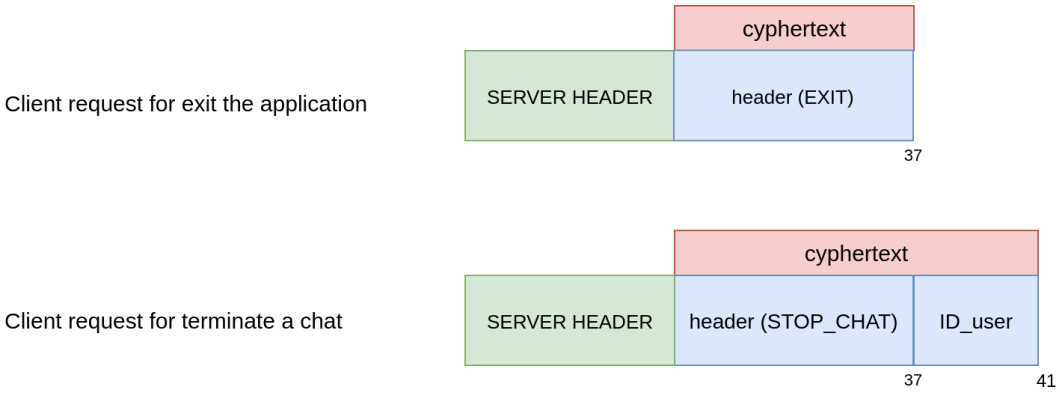

Figure 4.6: Other commands Format

Chat Request and Answers

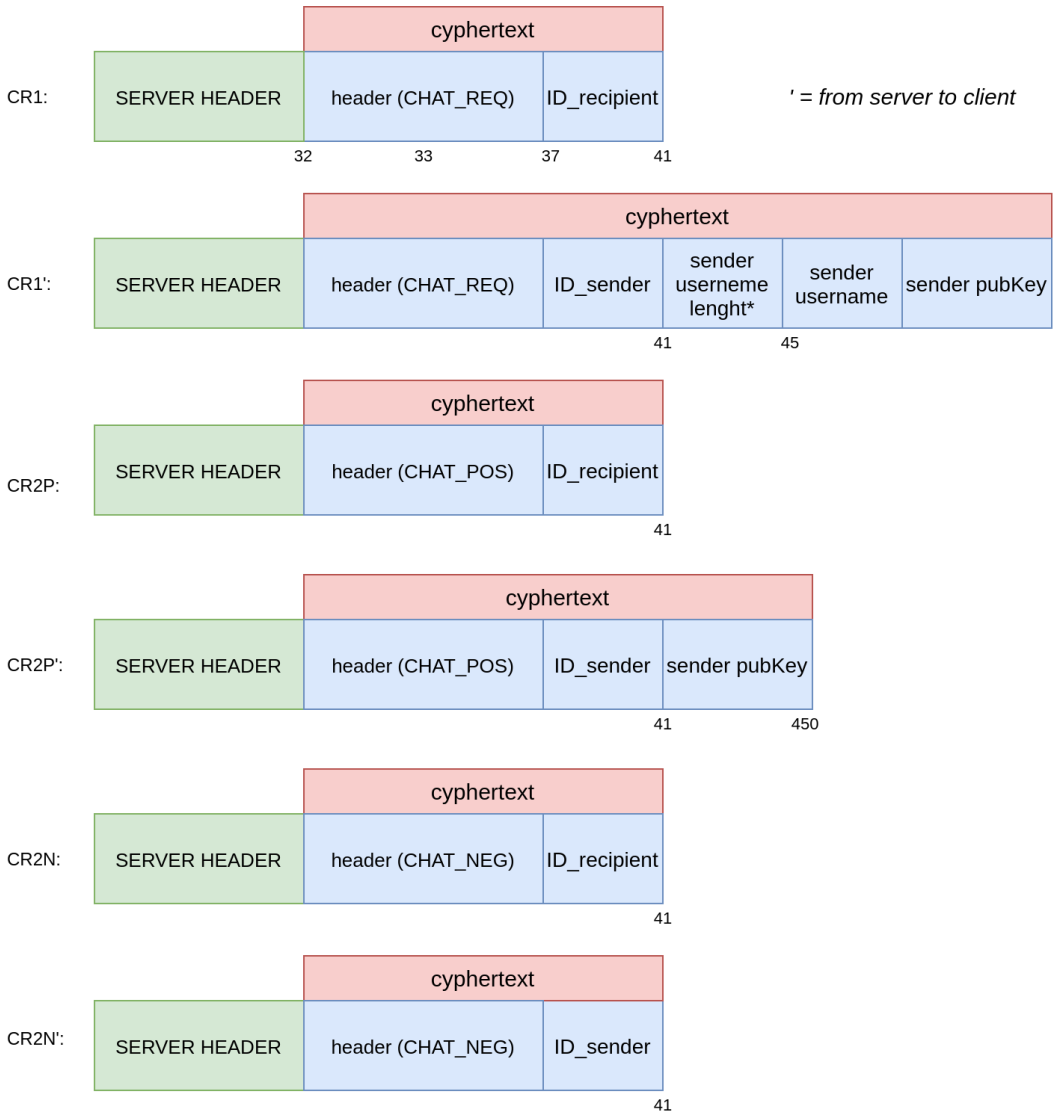

Figure 4.7: Chat Request Message Format

4.3 Chat

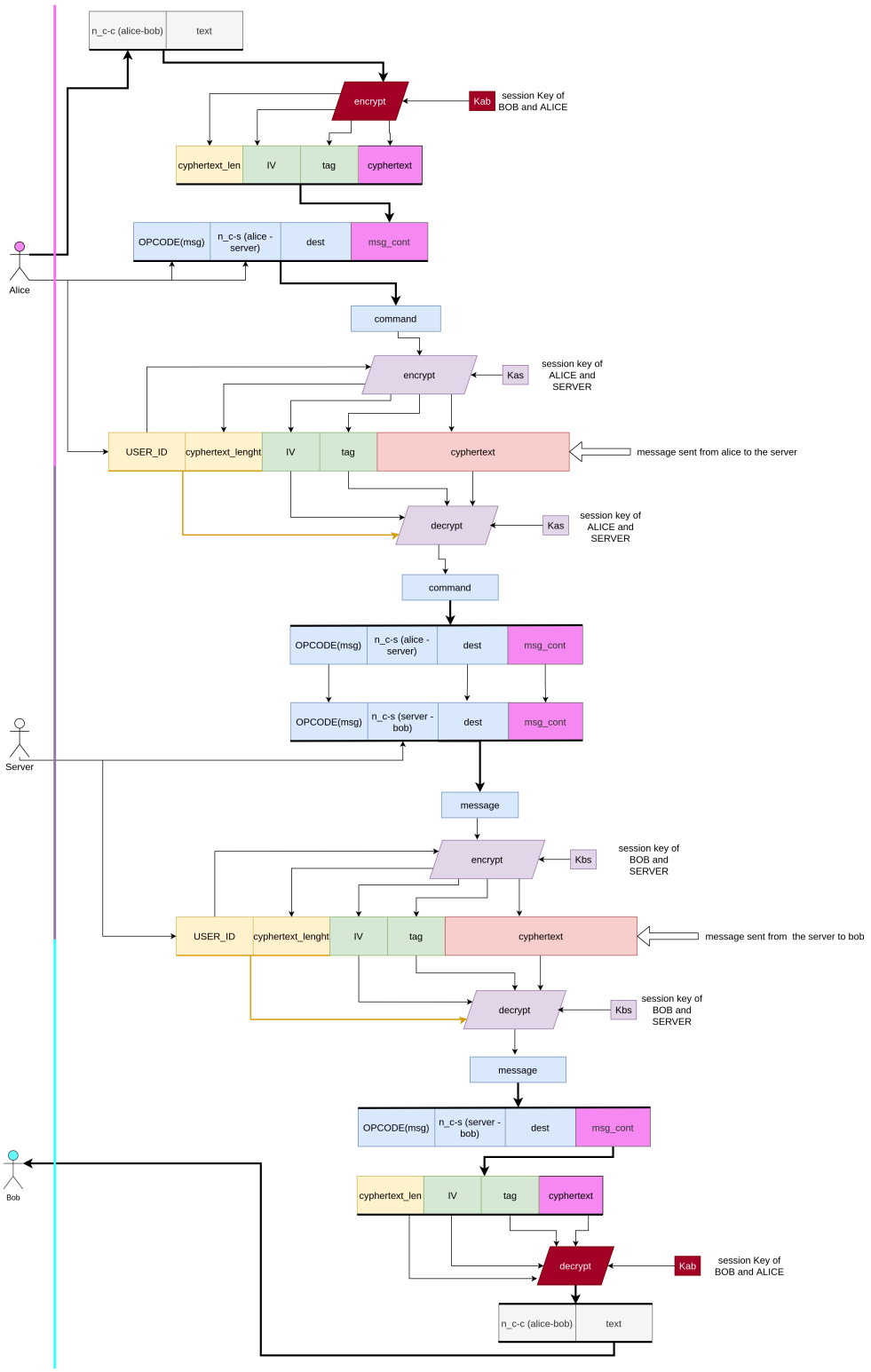

Figure 4.8: Client Client Handshake Protocol Schema