UNIVERSITÀ DI PISA

Department Information Engineering - DII

Foundations of Cybersecurity

Security Online Chat

Authors:
Luigi Gjoni
Fernando De Nitto
Marsha Gómez Gómez

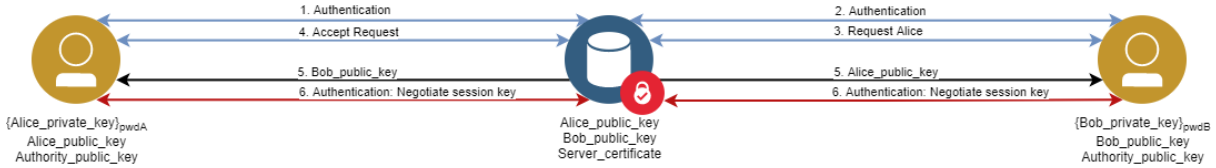
AY 2020/2021

Table of Contents

Project Guidelines and Requirements

1. Users are already registered on the server through public keys. Users authenticate themselves through said public key.
2. After the log-in, a user can see other available users logged to the server.
3. An user can send a "*request to talk*" message to another user.
4. The user who receives the "*request to talk*" can either **accept** or **refuse**.
5. If the request is **accepted**, the users proceed to chat through the server using an *end-to-end encrypted and authenticated* communication.
6. When the client application starts, Server and Client must authenticate.
7. Server must authenticate with a public key certified by a certification authority.
8. Clients must authenticate with a public key (pre-installed on server). The corresponding private key is protected with a password on each client.
9. After authentication a *symmetric session key* must be negotiated.
10. The negotiation must provide *"Perfect Forward Secrecy"*.
11. All session messages must be encrypted and every message in the session must be protected against replay attacks.
12. After a "*request to talk*" is accepted, the server sends to both clients the public key of the other client.
13. Before starting the chat a symmetric session key must be negotiated.
14. When a chat starts, the clients cannot start another chat (*1 chat active at a time*).
15. When a client wants to stop chatting, it shall log-off from the server.
16. The server is "*honest-but-curious*":
    - It will not communicate false public keys on purpose. (When the server communicates the public key of the user "*Alice*", the receiving client trusts that the server has given it Alice's public key).
    - It would try to understand the content of the communications between clients.
17. Use *C* or ***C++ language***, and ***OpenSSL*** library for crypto algorithms.
18. Key establishment protocol must establish one (or more) symmetric session key(s) with public-key crypto.
19. Then, session protocol must use session key(s) to communicate.
20. Communication must be **confidential, authenticated,** and **protected** against replay.
21. No coding vulnerabilities
22. Manage malformed messages

Basic Idea of communication



Main actors

- **Clients**: Clients will login to secure chat applications and may or may not interchange messages with another available client.
- **Server**: When a client requests for a login, server communicates the public key of the client A (*Alice*), the receiving client B (*Bob*) trusts that the server has given it Alice's public key, accepts connection, both clients will be able to reply and forward messages in a secure way.

Functional Requirements

- **Clients**:
  - *Login:* An user should be able to enter a username and password. Errors will occur if a space is left blank, the username doesn't exist, or the password doesn't match with the username. If the username and password matches, the user shall be set online and able to message anyone else online.
  - *Online Username List*: An user shall be able to get the username list of all connected clients.
  - *Request-To-Talk:* An user shall be able to send a request to another online person to start a chat talk.
  - *Accept or Deny Chat*: Before starting a chat talk, the user who receives the *Request-To-Talk* shall see accept request and reject request options.
  - *One-to-One Chat*: An user shall be able to send a message to another user who already accepted the request.
  - *One Chat only*: An user shall be able to participate in one chat per time with another client.
  - *Exit Chat*: An user shall be able to close the chat application.
- **Server**:
  - *Multiple users to one server*: The server shall be able to accept a connection by a client.
  - *Message interchange:* The server should be able to submit the client's requests.
  - *Manage Public Keys*: The server shall be able to correctly send the client's corresponding public key.

No-functional Requirements

- Communications must be confidential, authenticated and protected against replay
- Communications must use sessions key
- The negotiation of session key must provide Perfect Forward Secrecy
- All session messages must be encrypted with authenticated encryption mode
- Client must authenticate with a public key and the corresponding private key must be protected with a password
- Server must authenticate with a public key certificated by a certification authority
- Manage malformed messages

Server assumptions

- The server is "*honest-but-curious*":
    - It will not communicate false public keys on purpose. (When the server communicates the public key of the user "Alice", the receiving client trusts that the server has given it Alice's public key).
    - It would try to understand the content of the communications between clients.

Client assumptions

- Users are already registered on the server through public keys. Users authenticate themselves through said public key.
- After the log-in, a user can see other available users logged to the server.
- An user can send a "*request to talk*" message to another user.
- The user who receives the "*request to talk*" can either accept or refuse.
- If the request is accepted, the users proceed to chat through the server using an *end-to-end encrypted* and *authenticated* communication.
- When the client application starts, Server and Client authenticate:
    - *Server* authenticates with a public key certified by a certification authority.
    - *Client* authenticates with a public key (pre-installed on server). The corresponding private key is protected with a password on each client.
- After authentication, a symmetric session key is negotiated.
    - The negotiation provides *Perfect Forward Secrecy*.
    - All session messages are encrypted and authenticated.
    - Every message in the session is protected against replay attacks.
- After a "*request to talk*" is accepted, the server sends to both clients the public key of the other client.
- Before starting the chat, a symmetric session key is negotiated.
    - The negotiation must provide *Perfect Forward Secrecy*.
    - All session messages must be encrypted and authenticated, and they must be protected against a replay attack.
- When a chat starts, the clients cannot start another chat (*One chat active at a time)*.
- When a client wants to stop chatting, it shall log off from the server.

General Guideline

- Use **C** or **C++** language, and **OpenSSL** library for crypto algorithms.
- Key establishment protocol establishes one (or more) symmetric session key(s) with public-key crypto.
- Then, session protocol uses session key(s) to communicate.
- Communication must be confidential, authenticated, and protected against replay.
    - No coding vulnerabilities (Secure coding principles)
- Manage malformed messages

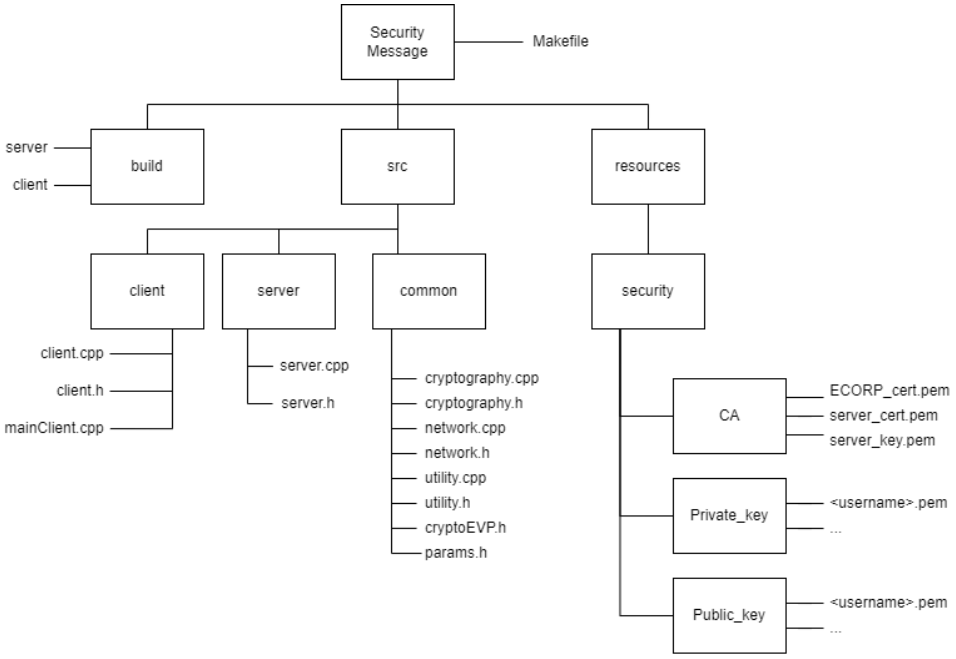
Implementation Choices

The main purpose of this application is to develop a secure online messaging app. Secure messaging can involve a lot of elements, so we have chosen the most important factors to be a secure messaging app.

The implementation of the project was performed using the **C ++** language and using the **OpenSSL v1.1.1** library (requirement 17). The code has been made as modular as possible using an **OOP** approach when possible especially for modules shared by both the client and the server. Communications between client and server have been implemented and managed with ***multi-threaded sockets***.

Skeleton of the Project

The project attached to this document is divided into different folders and files described below for completeness following the directory hierarchy.



- The **makefile** contains shell commands for the clean, build and compilation of the both environments: ***client*** and ***server***.
- The **client** and **server** folders contain the source files of the client and the server respectively.
- The **common** folder contains the shared functions regarding network and cryptography used for the correct execution of server and client environments.
- The **resources** folder contains all the files regarding the Certification Authority and the keys needed for the asymmetric encryptions and decryptions and for the digital signatures operations.

Compilation

- Go to the project's root folder: `Security-Chat/`
- Compile the project with the binaries: `make all`

Project Execution

To execute the project it is necessary go to build directory the following instructions:

- **For the server**
  - ./server [OPTIONS]
- **For the client**
  - ./client [OPTIONS]

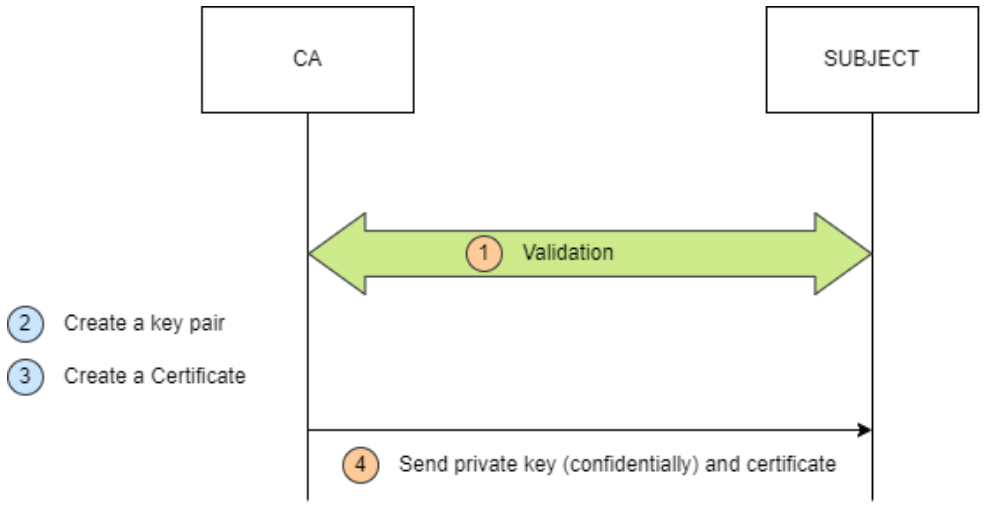
Options

Name	Shortcut	Default	Description	Server	Client
Port	-p	8080	Connect to custom port	✓	✓
Host	-h	127.0.0.1	Custom host-to-IP mapping		✓

CA and Certificates

Software and Methodology

Simple Authority software was used to generate the certificates. A certification authority was first created and then user certificates were generated along with their private keys following one of the two schemes available in the literature.



For each private key the correspondent public key was extracted by the command:

```
openssl rsa -pubout -in rsa_privkey.pem -out rsa_pubkey.pem
```

A Certificate Revocation List (CRL) was also generated, stored and verified by the application (in the client module).

Certification Authority

The fictional Certification Authority is called EVIL CORP (inspired by the Mr.Robot TV Series) and an example of certificate issued by the CA is shown in the following picture.

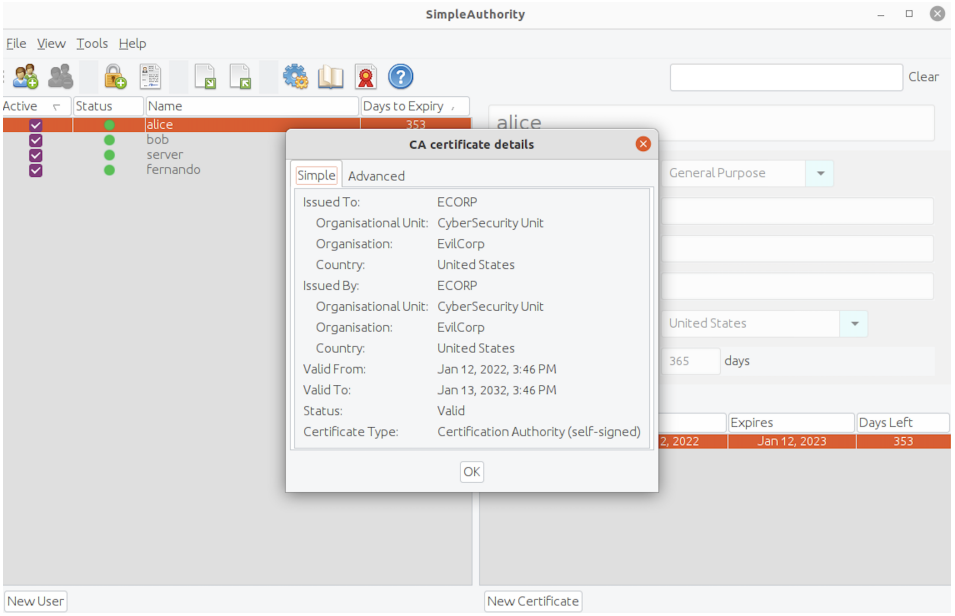


Figure 1: Self-Signed Certificate of CA

Certified Entities and credentials

Certified entities are listed in the following list along with their private key passwords. The same passwords are used in the application for user login. This last operation is done only on the client side in order to verify the presence of the user in the system.

User	Password
server	server
alice	alice
bob	bobo[1]
marsha	marsha
fernando	fernando
luigi	luigi
fsociety	mrrobot

Each of the entities described in the above table has a certificate signed by a CA with a 365 days validity starting from January 12, 2022.

[1] The password of bob is "*bobo*" since SimpleAuthority doesn't allow a password lower than 4 characters for security reasons.

Modules Descriptions

In this section we'll present a brief introduction on all the main modules of the application. For each module It will be shown a general description of the module itself, the main features of the module and its dependencies.

Server

This module represents the messaging application server and it is implemented using a socket-based communication model by multi-threading.

Main Features of the module

- It verifies the identity of the clients
- It provides to the clients the list of online users
- It forwards the messages from a client to another
- It manages the structure that contains the user list of the chat application

Library Dependencies (linked libraries in compilation)

- -pthread: for the multi-threading
- -lcrypto: for the in OpenSSL Library Functions and Structures

Module Dependencies (module included in the source file)

- params.h: common params shared by the application
- cryptography.h: suite of security functions
- utility.h: collection of utility functions

Client

This module represents the messaging application client and it is implemented using a socket-based communication model by multi-threading.

Main Features of the module

- It verifies the certificate of the server querying a Certification Authority

- It ask to the server to talk with another clients
- It sends/receives message to/from another client through the server

Library Dependencies (linked libraries in compilation)

- -pthread: for the multi-threading
- -lcrypto: for the in OpenSSL Library Functions and Structures

Module Dependencies (module included in the source file)

- params.h: common params shared by the application
- cryptography.h: suite of security functions
- network.h: suite of networking functions
- utility.h: collection of utility functions

Cryptography

This module represents the core of the security mechanisms in the application. It is developed with an OOP Approach and it is used both by the client and the server module that could call the security functions creating an instance of the class Cryptography. Each instance of the Cryptography class contains a data structure named **CryptoEVP** where the parameters useful for the end-to-end communication are stored. The data structure is created during the creation of an instance of the class and it is deallocated when the class destructor is called (or automatically at the end of the function when the variable becomes out of scope).

The CryptoEVP structure (file crypto_evp.h) contains mainly:

- Counters (sent and received)
- Server Session Key
- Client Session Key
- Peer Username
- State of the client
- Message Type
- Socket File Descriptor of the Server

Main Features of the module

- It manages all the quantities passed in the communication in a secure coding way (increment of nonces, counters etc.)
- It implements a function to generate Random Number (it encapsulates the OpenSSL PRG functions)
- It implements symmetric encryption/decryption functions
- It implements asymmetric encryption/decryption functions
- It implements all the functions that manage che Certificates and the CRL

- It implements all the functions that de/serialize quantities in the OpenSSL environment (keys, certificates etc. using BIOs)

Library Dependencies (linked libraries in compilation)

- -lcrypto: for the in OpenSSL Library Functions and Structures

Module Dependencies (module included in the source file)

- utility.h
- cryptography.h: header file

Utility

This module contains few utility functions don't directly linked with the implementation of the security functions in the project but useful for the logic of the communication.

Params

This module is a header file containing all the parameters shared by the parts of the communication. Most of them are numbers and directories re-defined to write a more human-readable code.

It includes:

- The directories of the certificate, CA, CRL and public/private keys
- Parameters for the Server
- Parameters for the Clients
- Parameters for the security functions

Security Design

In this section we'll describe the techniques used in the implementation to answer the requirements of the project. All the techniques described are implemented in the module "*cryptography.cpp*"following the secure coding methodology. The server and the client module use these functions to talk to each other. The way the modules use them are illustrated in detail at the end of this report in the sequence diagrams.

Authenticated Symmetric Encryption

For the symmetric encryption we adopt an **AES-GCM on 128 block size** since it implements both authentication and encryption in a single round. Of course the cons of this approach is that if the key is compromised both the authentication and the confidentiality are compromised. With the AES+HMAC alternative that is not true but we avoid the cost of managing two different keys choosing a simpler implementation alternative that provides a really good security strength.
In the application all the communications use an authenticated symmetric encryption to forward messages. The key used in the communication is a session key computed by a handshake from the two parts of the communication.

Asymmetric Encryption and their use

In this project both RSA and Diffie Hellman are used. RSA was used to implement the digital signature technique (public keys in the certificate are in the RSA 2048 bit that provide a security strength equivalent to AES 128 bit). The client verifies the certificate of the server and then both sign a fresh quantity plus other information to try to the other part their identity and the knowledge of the ephemeral keys (Direct Authentication and Perfect Forward Secrecy) used to compute the session key. The ephemeral keys are generated using standard parameters with Elliptic Curves DH on 256 bit (128 bit security strength).

Fresh and Unpredictable quantities

To avoid replay attacks we implement some simple functions that generate fresh and unpredictable quantities exchanged by all the communication parts.

Secure Coding Approach

All the quantities managed by functions follow a secure coding approach: all the increments on quantities are checked to avoid overflow and the size of messages passed into the function are checked before starting to manage them. No formatted strings are used (we use C++ strings and convert them into C strings when needed). Also the buffers and operations with the buffers were checked. We use the FlawFinder[2] tool to analyze all the modules we developed. This tool provides all the potential vulnerabilities in the file listed in the CWE List. This list includes both software and hardware weakness types listed by the Common Weakness Enumeration[3] (CWE™) Community.

[2] https://dwheeler.com/flawfinder/
[3] https://cwe.mitre.org/about/index.html

Messages

In this section we will describe which kind of messages are exchanged during all the steps in the communication and their format.

Type of Message

Each message is responsible for a specific purpose and the latter could be identified by the type of the message. We will briefly describe the type.

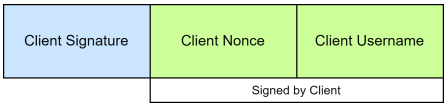
Message Type	Integer Value	Description
EXITED	0	The client shutdown the application
ONLINE_USERS	1	The client request the list of the online users
REQUEST_TO_TALK	2	The client send a request to another client
REQUEST_ACCEPTED	3	The peer accept the request to talk
REQUEST_REFUSED	4	The peer refuses the chat
TEXT_MESSAGE	5	Text Message exchanged during the chat
CHAT_SESSION	6	Send the ephemeral key to the peer and establish a chat session
UNKNOWN_USER	7	The server notify to the client that the peer is busy or unknown
CHAT_CLOSED	8	The client's peer closed the communication

Format of Message Exchanged

In this section we will describe all the formats of messages exchanged during the communication in order to see which parts are authenticated and which one is encrypted. The number of the message corresponds with the number in the sequence diagrams in the next paragraph.

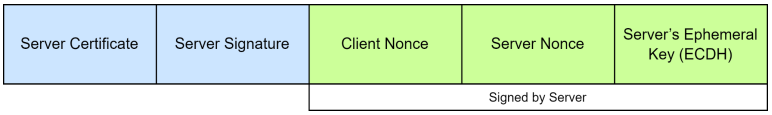
Client Server Authentication

MESSAGE A.5 (Client->Server)



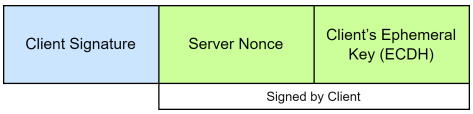
This message is sent from client to the server to authenticate itself by means of its digital signature.

MESSAGE A.11 (Server->Client)



This message is sent from server to client to authenticate itself with a certificate and its digital signature. The server also has its ephemeral key to the client.

MESSAGE A.21 (Client->Server)



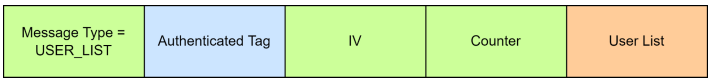
This message is sent from client to the server to provide its ephemeral key

User List Request

MESSAGE B.5 (Client->Server)

This message is sent from client to the server to request the list of online users in the system.

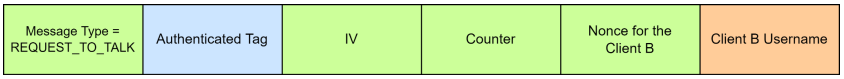
MESSAGE B.10 (Server->Client)



This message is sent from server to the client to request the list of online users in the system.

Chat Request to talk

MESSAGE C.7 (Client A->Server)



This message is sent from Client A to the server to forward a request to talk to Client B.

MESSAGE C.12 (Server->Client B)

This message is sent from server to Client B to forward a request to talk from Client A.

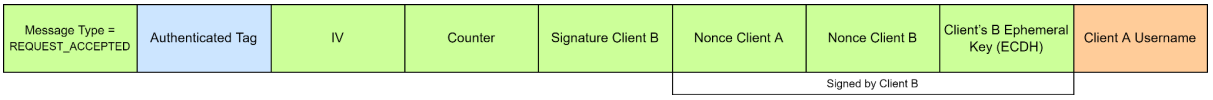
Chat Request to talk Refused

MESSAGE D.13 (Client B->Server->Client A)

This message is sent from Client B to the server to forward a refused request to talk to Client A.

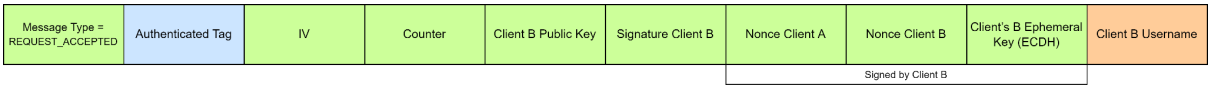
Chat Request to talk Accepted

MESSAGE E.12 (Client B->Server)



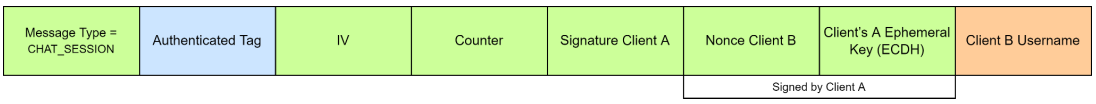
This message is sent from Client B to the server asking to forward its signed ephemeral key to Client A.

MESSAGE E.17 (Server->Client A)



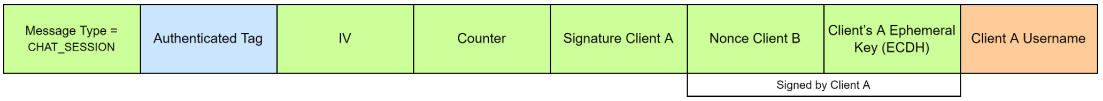
This message is sent from server to Client A to forward the signed ephemeral key of Client B.

MESSAGE E.32 (Client A->Server)



This message is sent from Client A to the server asking to forward its signed ephemeral key to Client B.

MESSAGE E.37 (Server->Client B)



This message is sent from server to Client B to forward the signed ephemeral key of Client A.

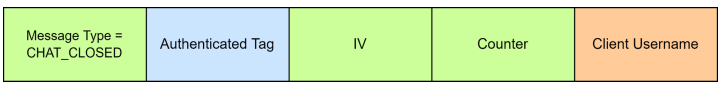
Text Message

MESSAGE F.18 (Client->Server->Client)

This message is the text message exchanged between two clients during the chat through the server.

Chat Closed

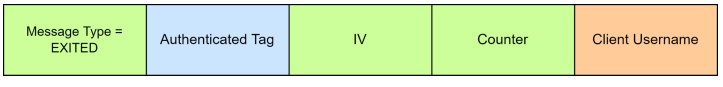
MESSAGE CHAT CLOSE (Client->Server->Client)



This message is sent from a client (who exited from the chat) to the Server. If the client was talking with another client the server forward the message to the peer that returns available.

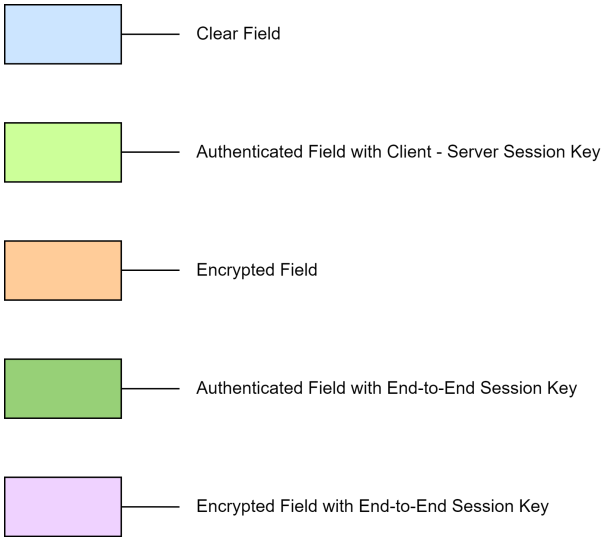
Application Exited

MESSAGE APP EXITED (Client->Server->Client)

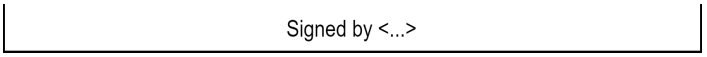


This message is sent from a client (who exited from the application) to the Server. If the client was talking with another client the server forward the message to the peer that returns available.

Legend:



If multiple fields are signed by use of a private key a white bar



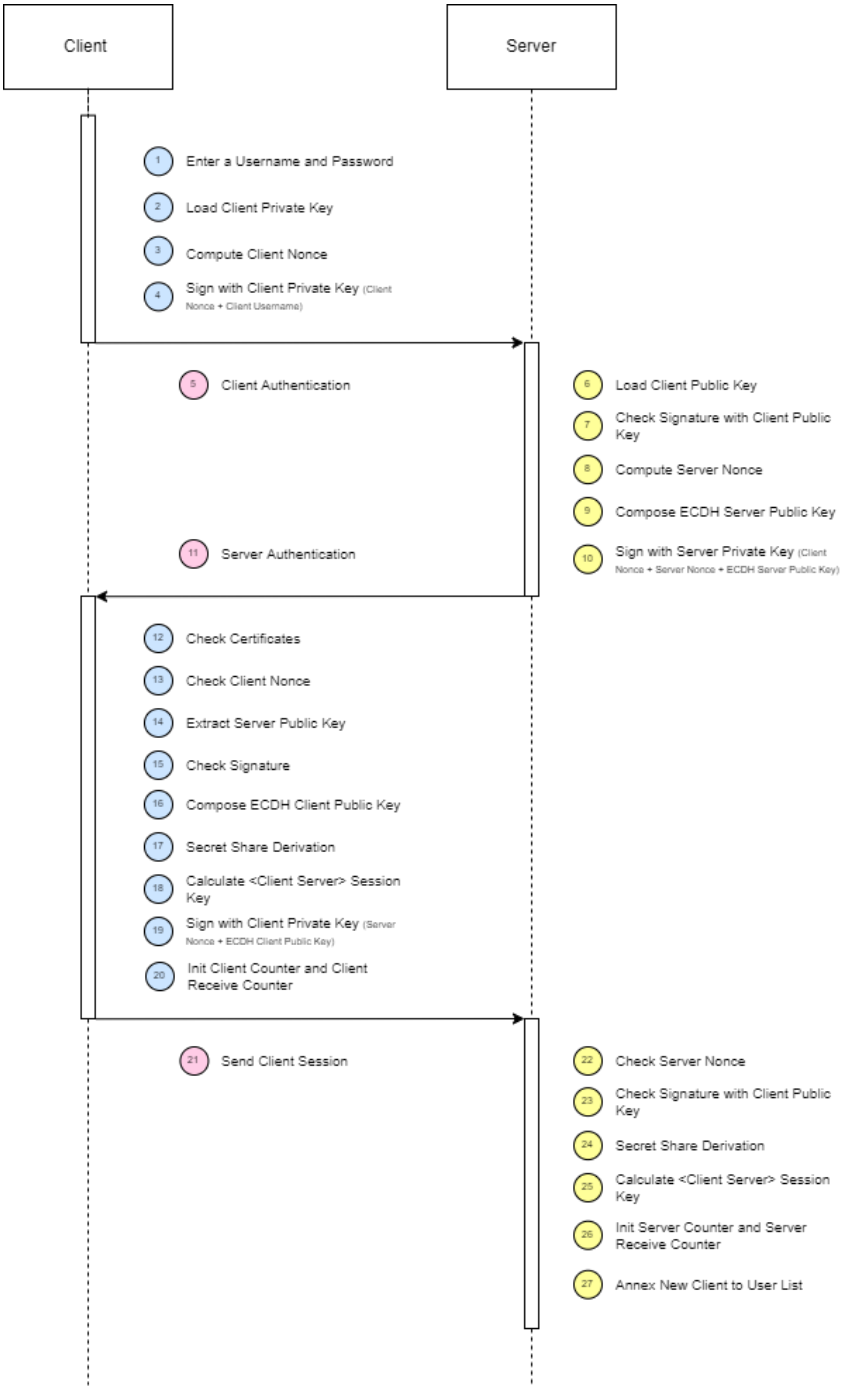
will cover the space below the interested signed fields.

Note: Since communication is based on sockets, a first message is always exchanged between the parties to communicate the message size and allocate the correct amount of buffers. All other parameters used follow the dimensions of the standards and can be automatically deducted as the system is known.

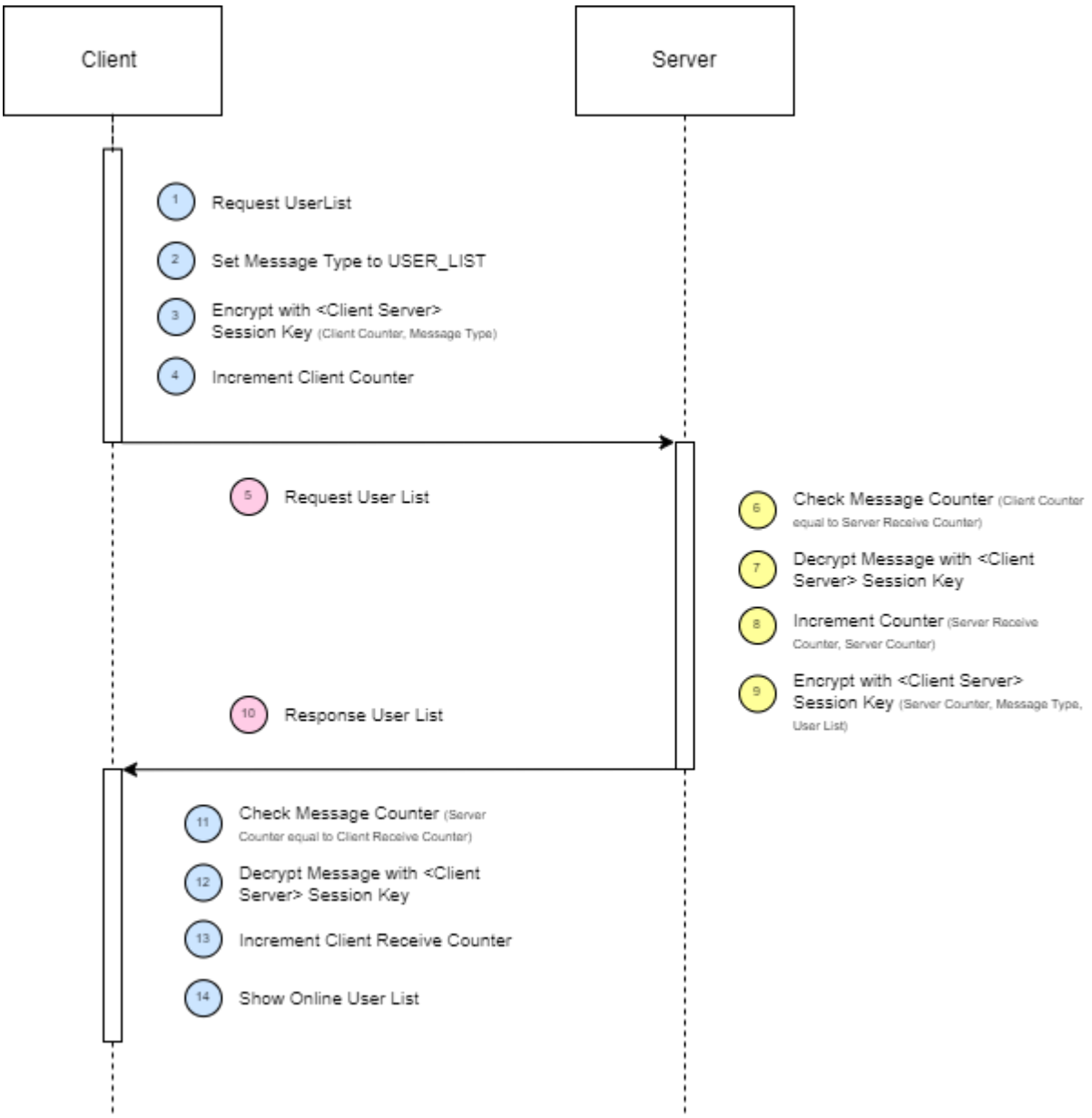
Sequence Diagrams

In this paragraph we describe the sequence diagram about all the communications established in the application by the parts.

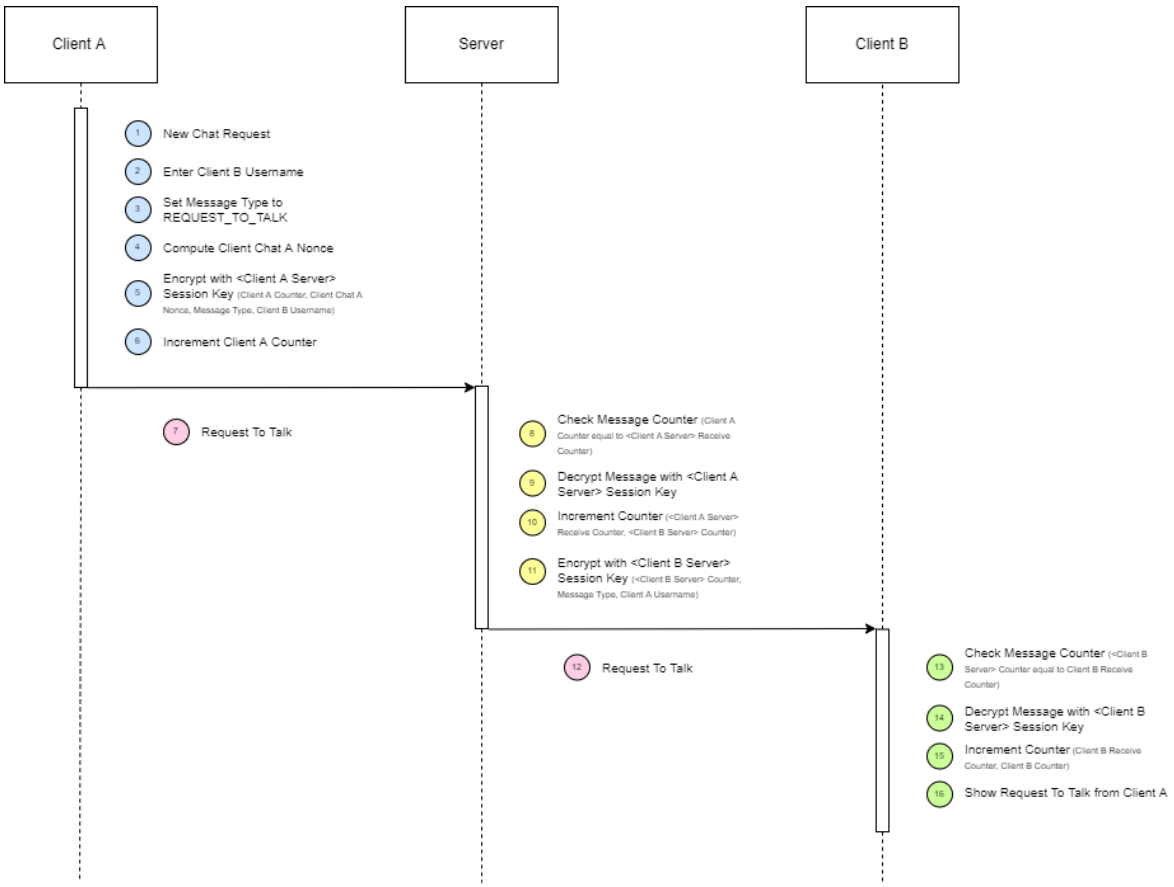
A. Client-Server Authentication



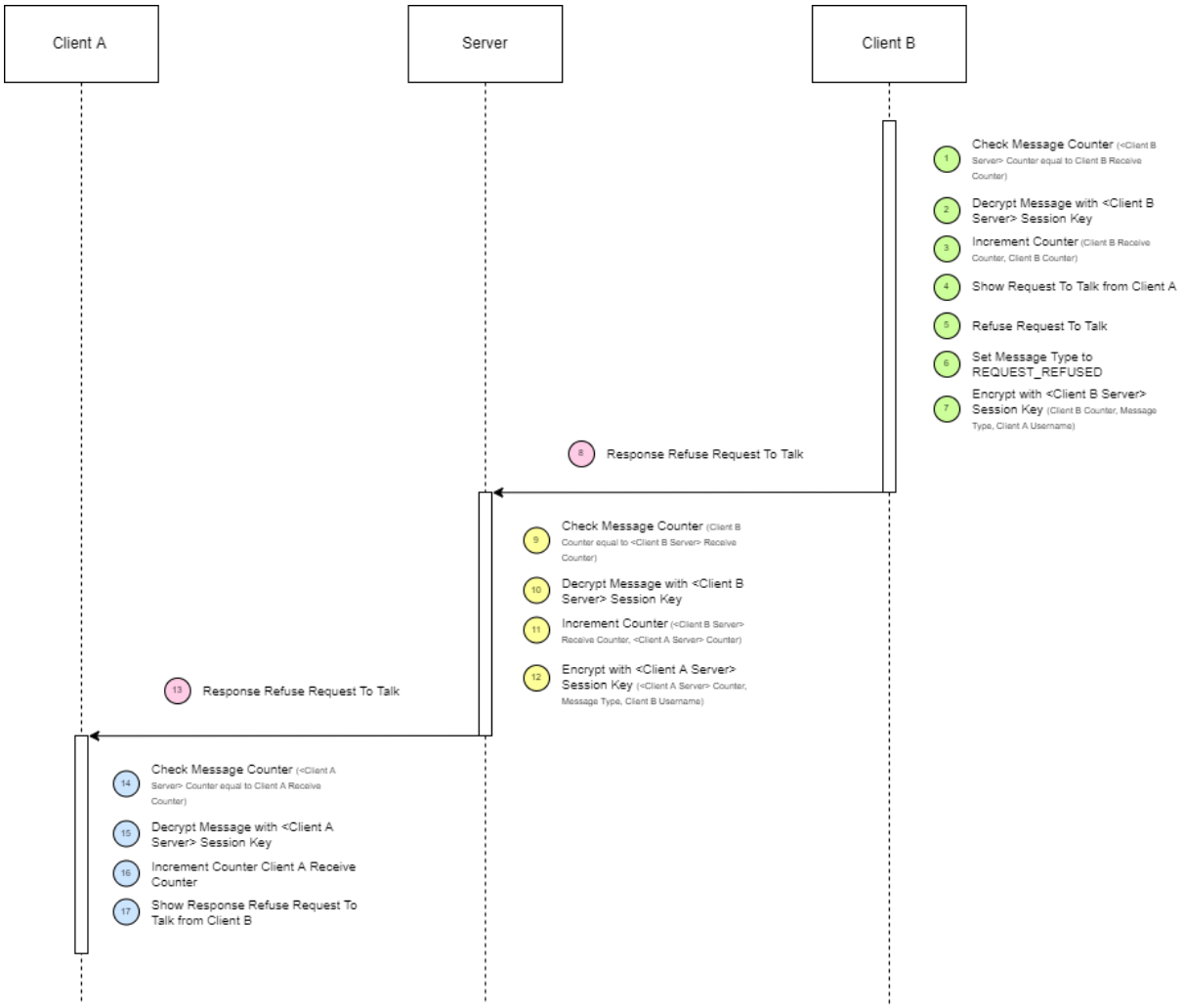
B. User List


Client
Server
1 Request UserList
2 Set Message Type to USER_LIST
3 Encrypt with <Client Server> Session Key (Client Counter, Message Type)
4 Increment Client Counter
5 Request User List
6 Check Message Counter (Client Counter equal to Server Receive Counter)
7 Decrypt Message with <Client Server> Session Key
8 Increment Counter (Server Receive Counter, Server Counter)
9 Encrypt with <Client Server> Session Key (Server Counter, Message Type, User List)
10 Response User List
11 Check Message Counter (Server Counter equal to Client Receive Counter)
12 Decrypt Message with <Client Server> Session Key
13 Increment Client Receive Counter
14 Show Online User List

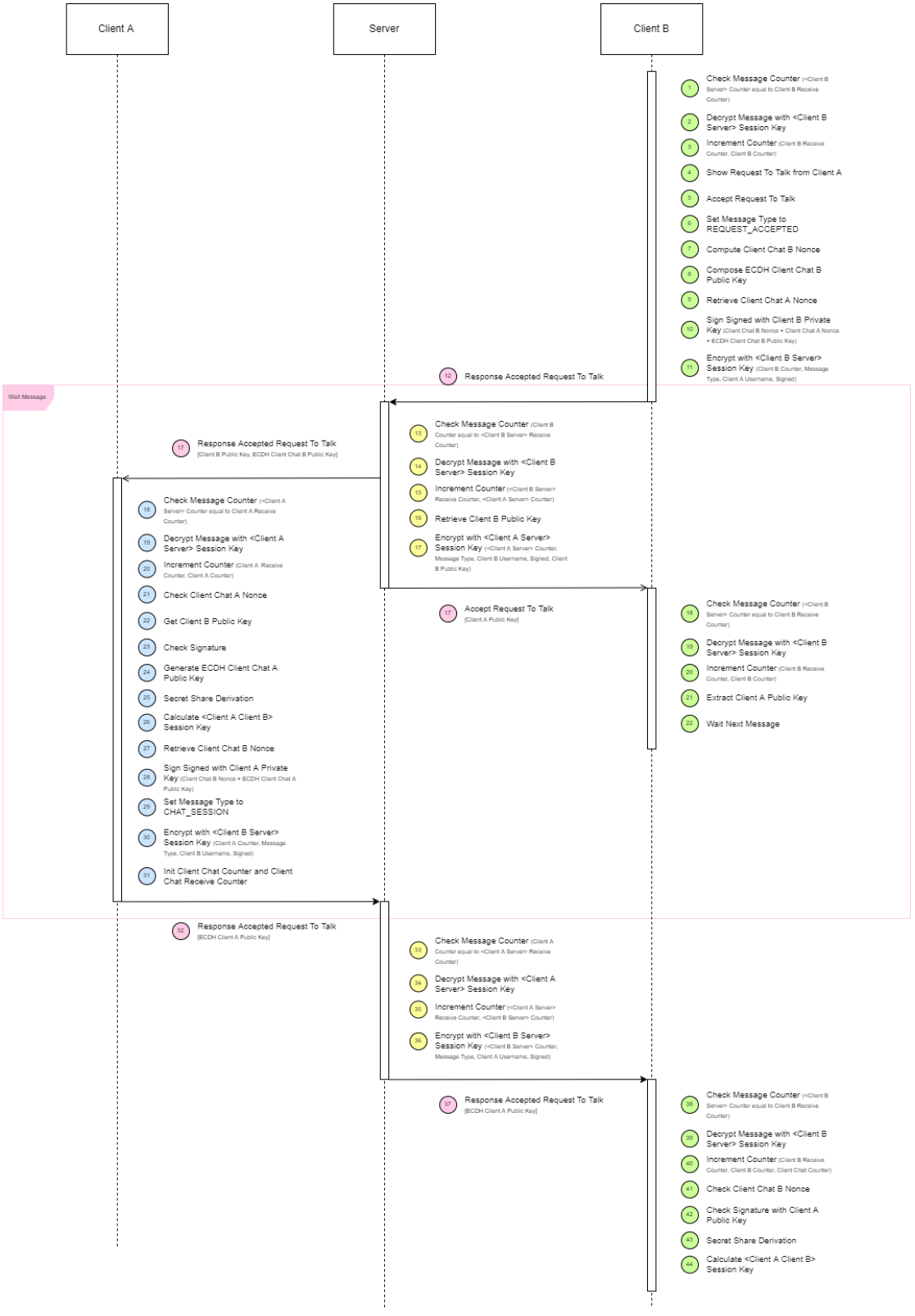
C. Chat Request to Talk



D. Chat Refuse



E. Chat Accepted



F. Secure Chatting