Department Information Engineering - DII
Foundations of Cybersecurity

Online Secure Chat

Mohammed Ridha Mohammed Mohammed
Noushin Najafiragheb

Project Guidelines and Requirements

1. General Requirements

- Users are already registered on the server through public keys. Users authenticate themselves through the said public key.
- When the client application starts, the server and the client must authenticate. The server must authenticate with a public key certified by a certification authority, while the client must authenticate with the public key pre-installed on the server. The corresponding private key is protected with a password on the client.
- After authentication, the client and the server must negotiate a symmetric session key.
- After the log-in, a user can see other available users logged to the server.
- 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, we need to avoid this happening.

2. Chat Requirements

- A 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 chat through the server using an end-to-end encrypted and authenticated communication.
- After a "request to talk" is accepted, the server sends both clients the public key of the other client. Before starting the chat, the clients must negotiate a symmetric session key.
- When a chat starts, the clients cannot start another chat (1 chat active at a time).
- When a client wants to stop chatting, it shall log-off from the server.
- The server acts as an intermediary for the message exchanged.
- The symmetric keys negotiations must provide Perfect Forward Secrecy.
- All session messages must be encrypted, authenticated and protected against replay attacks.

Functional Requirements

1. 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.

2. Server:

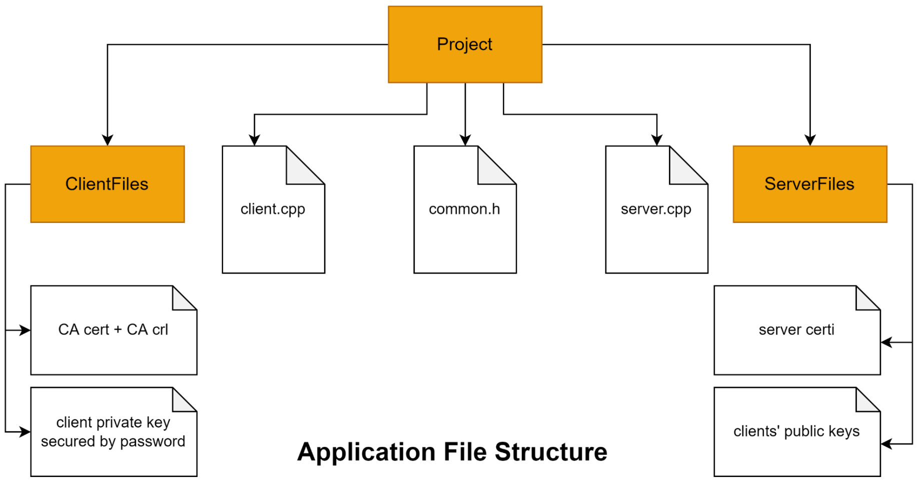
- *Multiple users to one server*: The server shall be able to accept a connection by a client.
- *Message interchange:* The server should be able to submit the client's requests.
- *Manage Public Keys*: The server shall be able to correctly send the client's corresponding public key.

Implementation Choice

- After the connection establishment with the server, the list of the available users and each recipient or sender usernames are encrypted to address privacy concerns of the users.
- The *CMDCODE* are authenticated to verify the user intentions, and it's not necessary to encrypt the operation type.
- Also, the counters are authenticated because we need to verify that the message source is the one we are expecting, but it's not considered sensitive data.
- Being public the keys are not encrypted, but only authenticated.
- The messages exchanged between two clients are always encrypted.

Skeleton of the Project

The project is divided into different files described below for completeness following the directory hierarchy.



Application File Structure

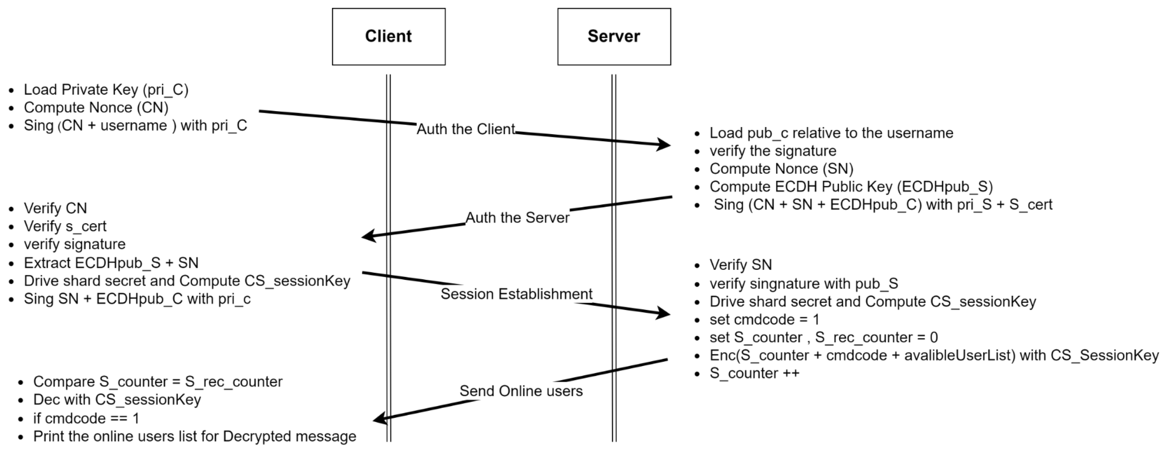
The client and server folders contain the source files of the client and the server respectively. The common file contains the shared functions used for the correct execution of server and client environments.

Authentication

When the application starts, the server and the client must authenticate each other:

- The server authenticates using a certificate released by a trusted certification authority, signing the DH public key. The client can obtain the server's public key from the certificate to verify the signature.
- The client authenticates by signing his(er) DH public key. The server verifies the signature using the client's pre-installed public key.

- The messages contain nonces to avoid replay attacks.



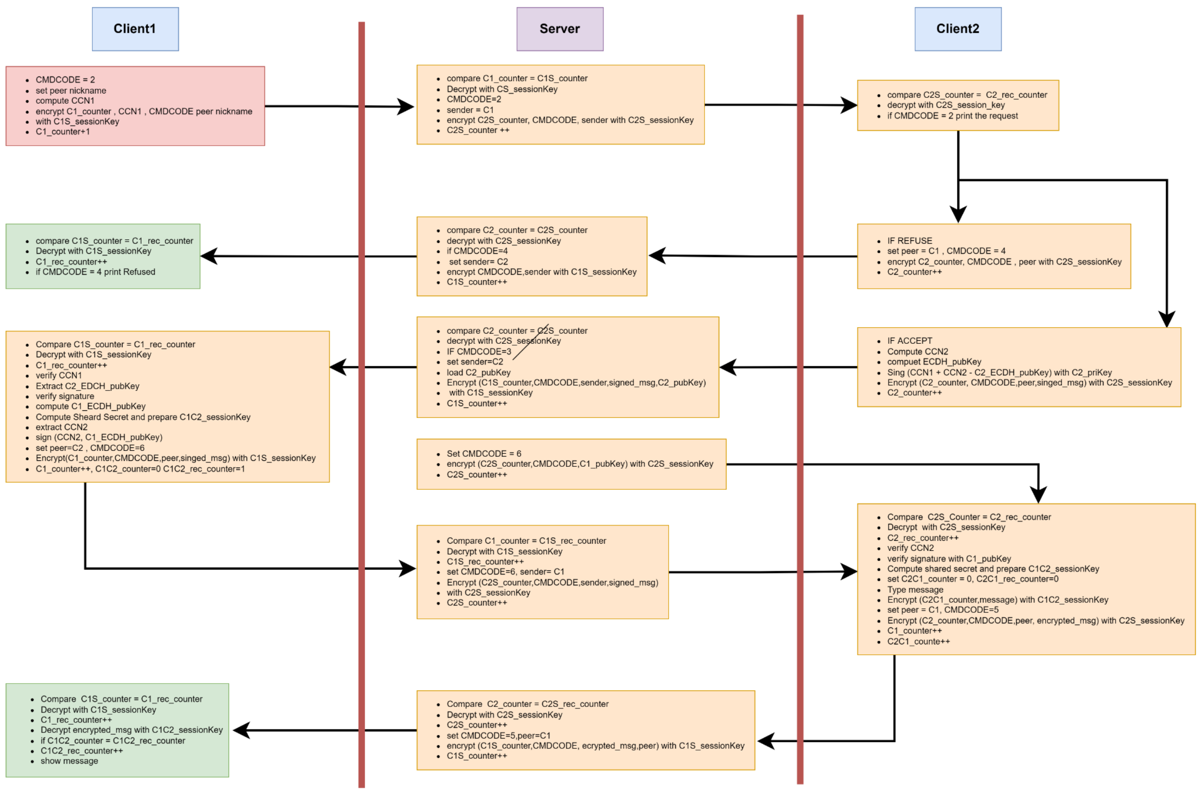
Client
Server
• Load Private Key (pri_C)
• Compute Nonce (CN)
• Sing (CN + username) with pri_C
Auth the Client
• Load pub_c relative to the username
• verify the signature
• Compute Nonce (SN)
• Compute ECDH Public Key (ECDHpub_S)
• Sing (CN + SN + ECDHpub_C) with pri_S + S_cert
Auth the Server
• Verify CN
• Verify s_cert
• verify signature
• Extract ECDHpub_S + SN
• Drive shard secret and Compute CS_sessionKey
• Sing SN + ECDHpub_C with pri_c
Session Establishment
• Verify SN
• verify singnature with pub_S
• Drive shard secret and Compute CS_sessionKey
• set cmdcode = 1
• set S_counter , S_rec_counter = 0
• Enc(S_counter + cmdcode + avalibleUserList) with CS_SessionKey
• S_counter ++
Send Online users
• Compare S_counter = S_rec_counter
• Dec with CS_sessionKey
• if cmdcode == 1
• Print the online users list for Decrypted message

Chatting

- The client can send a request to talk with an available online user (--r <peer's username>).

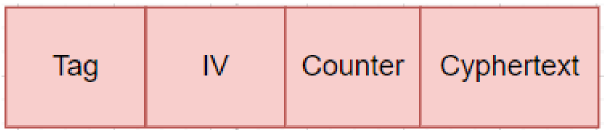
- The server receives the request and sends a message to the other client with (CMDCODE = 2).
- The peer can accept (--y) or refuse (--n) the request
- If the peer accept the request an ECDH key pair will be created to establish as CC_session_Key

- When the CC_session_key established the peers can exchange messages and set CMDCODE=5


Client1
Server
Client2
CMDCODE = 2
set peer nickname
compute CCN1
encrypt C1_counter , CCN1 , CMDCODE peer nickname
with C1S_sessionKey
C1_counter+1
compare C1_counter = C1S_counter
Decrypt with CS_sessionKey
CMDCODE=2
sender = C1
encrypt C2S_counter, CMDCODE, sender with C2S_sessionKey
C2S_counter ++
compare C2S_counter = C2_rec_counter
decrypt with C2S_session_key
if CMDCODE = 2 print the request
compare C1S_counter = C1_rec_counter
Decrypt with C1S_sessionKey
C1_rec_counter++
if CMDCODE = 4 print Refused
compare C2_counter = C2S_counter
decrypt with C2S_sessionKey
if CMDCODE=4
set sender= C2
encrypt CMDCODE,sender with C1S_sessionKey
C1S_counter++
IF REFUSE
set peer = C1 , CMDCODE = 4
encrypt C2_counter, CMDCODE , peer with C2S_sessionKey
C2_counter++
Compare C1S_counter = C1_rec_counter
Decrypt with C1S_sessionKey
C1_rec_counter++
verify CCN1
Extract C2_EDCH_pubKey
verify signature
compute C1_ECDH_pubKey
Compute Sheard Secret and prepare C1C2_sessionKey
extract CCN2
sign (CCN2, C1_ECDH_pubKey)
set peer=C2 , CMDCODE=6
Encrypt(C1_counter,CMDCODE,peer,singed_msg) with C1S_sessionKey
C1_counter++, C1C2_counter=0 C1C2_rec_counter=1
compare C2_counter = C2S_counter
decrypt with C2S_sessionKey
IF CMDCODE=3
set sender=C2
load C2_pubKey
Encrypt (C1S_counter,CMDCODE,sender,signed_msg,C2_pubKey)
with C1S_sessionKey
C1S_counter++
IF ACCEPT
Compute CCN2
compuet ECDH_pubKey
Sing (CCN1 + CCN2 - C2_ECDH_pubKey) with C2_priKey
Encrypt (C2_counter, CMDCODE,peer,singed_msg) with C2S_sessionKey
C2_counter++
Set CMDCODE = 6
encrypt (C2S_counter,CMDCODE,C1_pubKey) with C2S_sessionKey
C2S_counter++
Compare C2S_Counter = C2_rec_counter
Decrypt with C2S_sessionKey
C2_rec_counter++
verify CCN2
verify signature with C1_pubKey
Compute shared secret and prepare C1C2_sessionKey
set C2C1_counter = 0, C2C1_rec_counter=0
Type message
Encrypt (C2C1_counter,message) with C1C2_sessionKey
set peer = C1, CMDCODE=5
Encrypt (C2_counter,CMDCODE,peer, encrypted_msg) with C2S_sessionKey
C1_counter++
C2C1_counte++
Compare C1_counter = C1S_rec_counter
Decrypt with C1S_sessionKey
C1S_rec_counter++
set CMDCODE=6, sender= C1
Encrypt (C2S_counter,CMDCODE,sender,signed_msg)
with C2S_sessionKey
C2S_counter++
Compare C1S_counter = C1_rec_counter
Decrypt with C1S_sessionKey
C1_rec_counter++
Decrypt encrypted_msg with C1C2_sessionKey
if C1C2_counter = C1C2_rec_counter
C1C2_rec_counter++
show message
Compare C2_counter = C2S_rec_counter
Decrypt with C2S_sessionKey
C2S_counter++
set CMDCODE=5,peer=C1
encrypt (C1S_counter,CMDCODE, ecrypted_msg,peer) with C1S_sessionKey
C1S_counter++

Authenticated encryption

The messages in a session are encrypted, authenticated to be protected against replay attacks. Hence, AES-GSM-128 is used to encrypt messages. 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 the two parts of the communicatio After encryption, each message has the following structure:



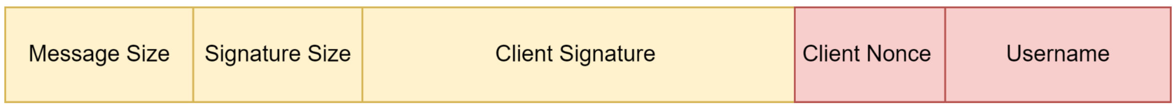
The Initialization Vector length is 12bytes, and it is randomly generated before each encryption. The counter is initialised to zero when a session starts, and it is incremented by 1 at each encrypted message. Both IV and counter are only authenticated. The counter approves the freshness of the message. A message is fresh if the counter received is equal to the actual counter +1.

Messaging

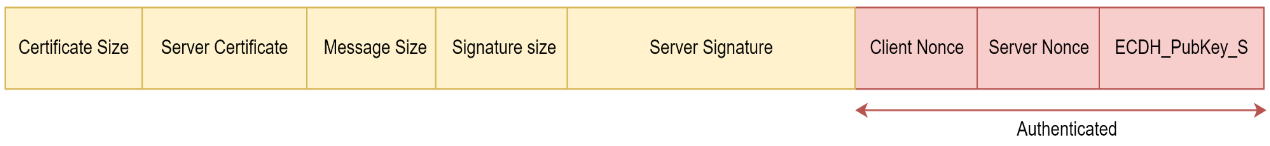
Each message has a different structure. The “cmdcode” is defined to identify the different structure of the messages.

Message	Command	cmdcode	Description
Exit	--ex	0	The user exits the application
List	--l	1	The user requests the list of the updated online users
RTT	--r	2	The user requests to talk with the specific user
Accept	--y	3	RTT request is accepted
Refuse	--n	4	RTT request is rejected
Message	<Any Message>	5	The chat messages
Session Establishment	<no command>	6	send ECDH_PUB_KEY and PUB_KEY to user who accepted the chat request
USER_NOT_FOUND	<no command>	7	RTT to unavailable/unknown client
Chat Closed	<no command>	8	The client’s peer closed the communication

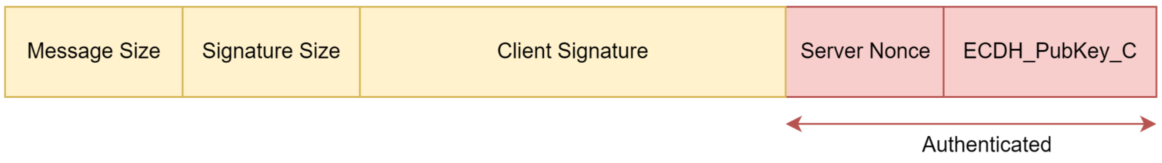
Client Authentication Message:
Client → Server



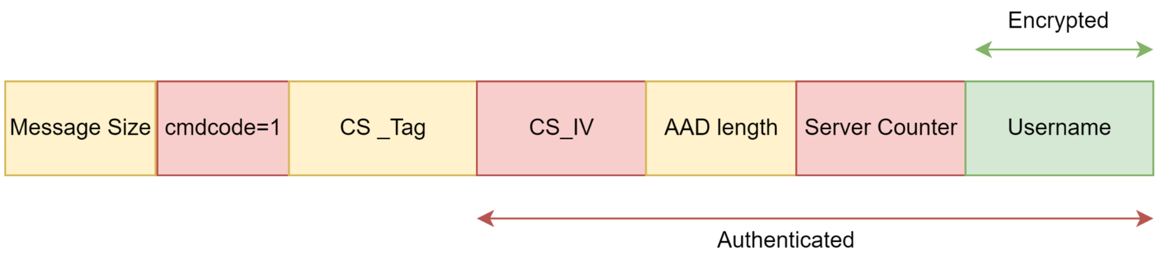
Server Authentication Message:
Server→ Client



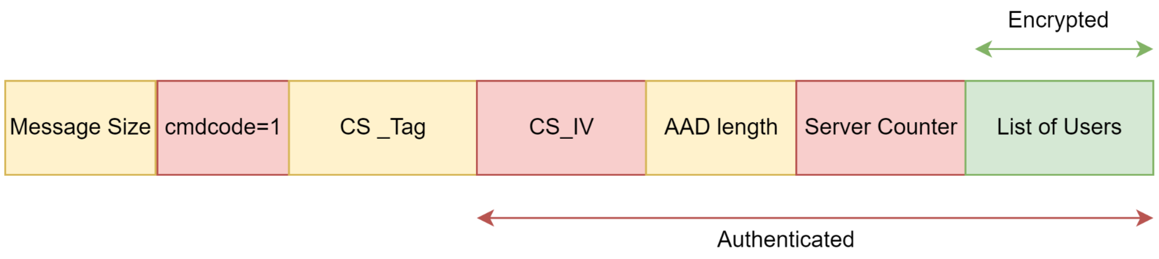
Session Establishment
Client → Server



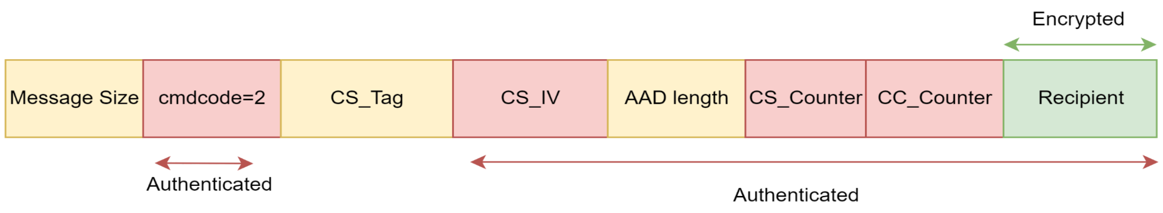
Available Users List:
Client → Server



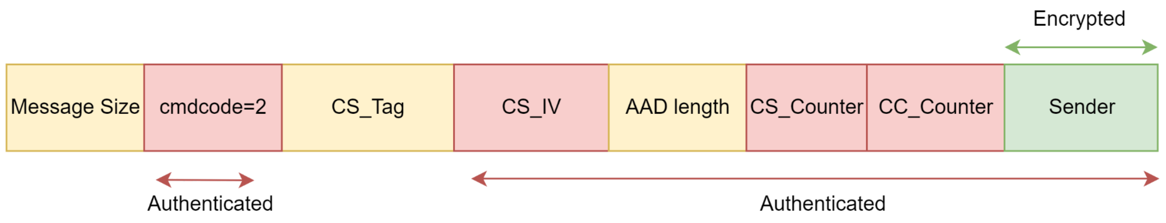
Server→ Client



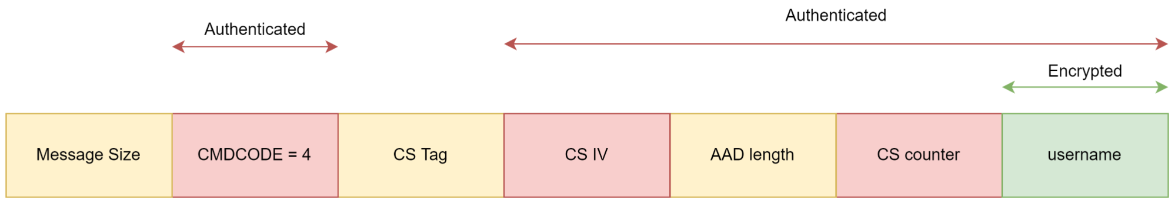
RTT from C1:
Client (C1) → Server



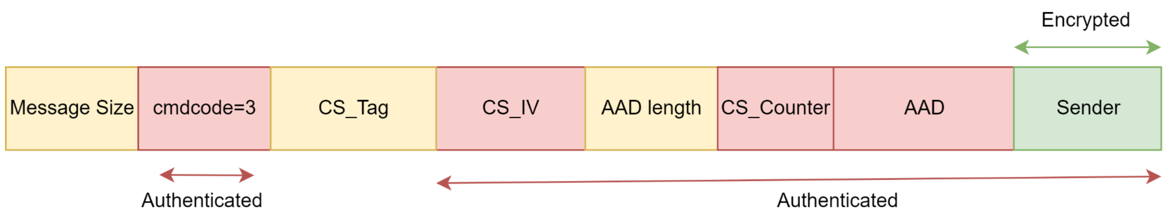
Server→ Client(C2)



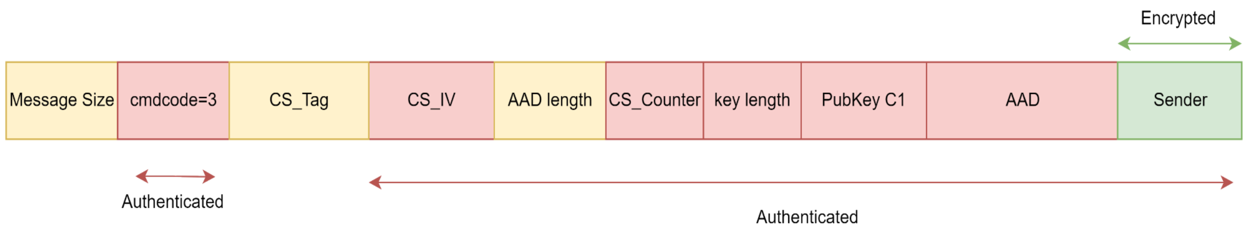
Refuse RTT:
Server → Client



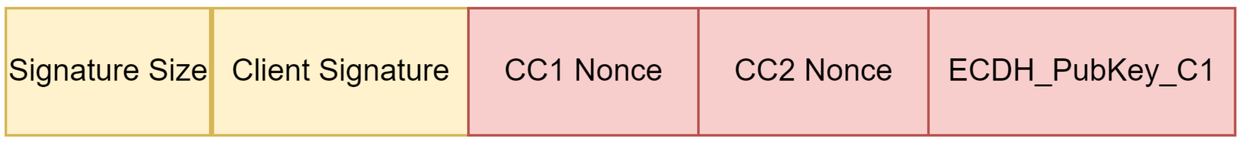
Accepting RTT:
Client(C2) → Server



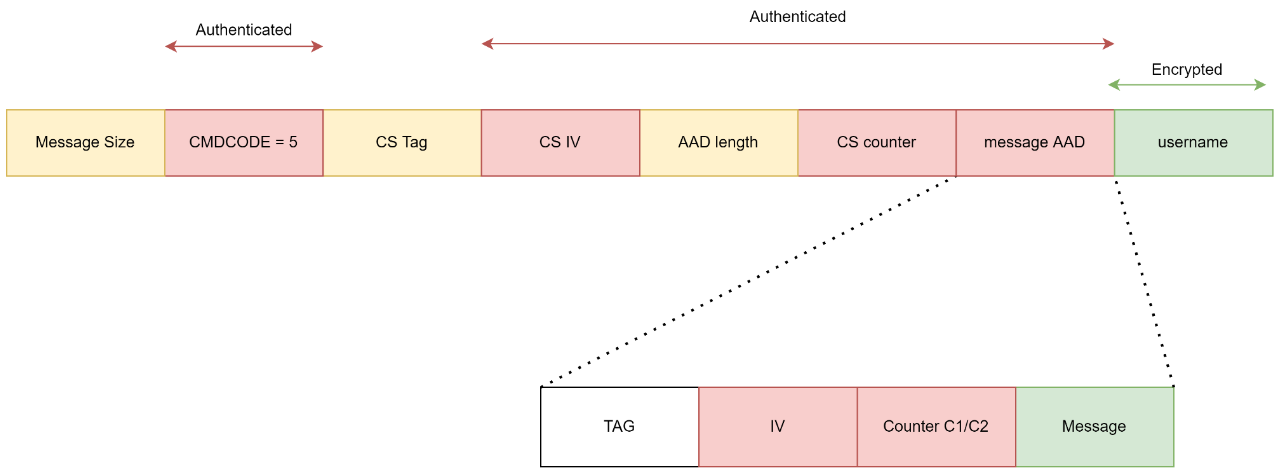
Server → Client (C1)



ADD structure:

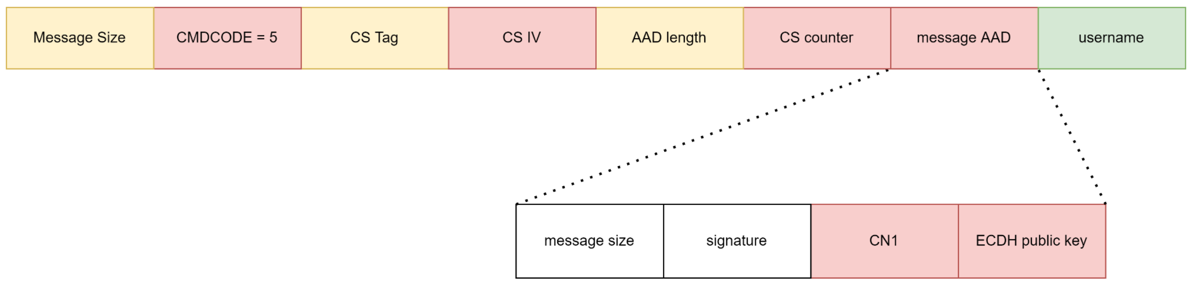


Chat Messages:

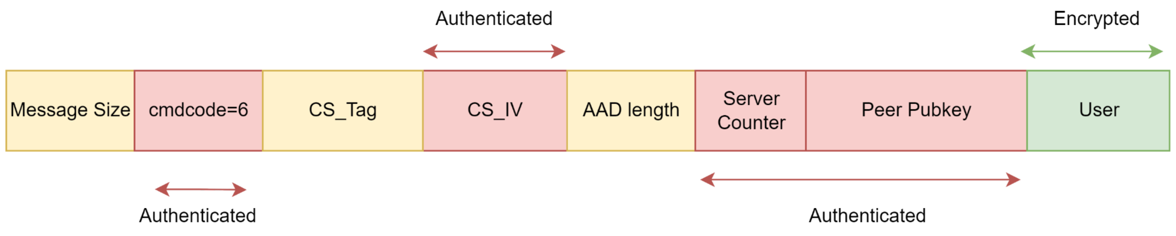


CC Session Establishment:

Client (C2) → Server



Server → Client (C1)


Authenticated
Encrypted
Message Size
cmdcode=6
CS_Tag
CS_IV
AAD length
Server Counter
Peer Pubkey
User
Authenticated
Authenticated