University of Pisa
Master's degree in Cybersecurity

Digital Signature Service Analysis Report

Giovanni D'Ambrosio, Umberto Melpignano

June 26, 2025

Table of Contents

1 Introduction

This document presents a comprehensive overview of the proposed Digital Signature Service (DSS), outlining the services made available to users as follows:

- **Keys Generation:** the server generates the user's public key and private key based on RSA-3072, if they do not already exist;
- **Public Key Retrieval:** the server sends the user the specified user's public key;
- **Digital Signature:** the server signs a document provided by the user with the client's private key using the RSA digital signature algorithm;
- **Key Deletion:** the server deletes the user's public key and private key. The user will need to get offline registered again.

2 System Architecture

The overall system is composed of a client application and a server daemon that communicate securely over TCP sockets using a layered protocol. The core architecture ensures:

- Mutual cryptographic key establishment using authenticated Ephemeral Diffie-Hellman (EDH);
- Secure communication through Authenticated Encryption in AES-256-GCM mode;
- User identity authentication through an encrypted channel.

3 Authentication Protocol

Key Establishment and server authentication

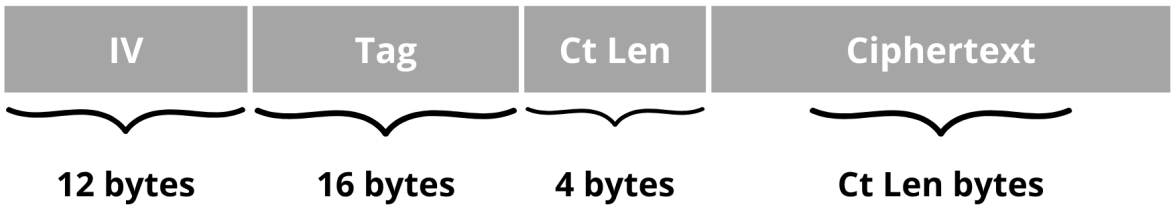
The key establishment algorithm is based on Ephemeral Diffie-Hellman built upon RFC 7919 standardized group ffdhe2048.

To achieve authentication, the server's public key, together with a nonce provided by the client to prevent replay attacks, is digitally signed through RSA-3072 using the server's private key. The signature is verifiable by the client through the public key made available to them.

The session key is then obtained after evaluating the SHA-256 digest of the shared secret computed by both the server and the client.

Session Encryption

The client and the server share messages on an encrypted channel through Authenticated Encryption in AES-256-GCM mode. Both the client and the server expect messages of the following format:



In case of a bad decryption or a tag verification failure, the receiver will terminate the session immediately.

User authentication

Through the encrypted channel, the users authenticate themselves by sending their password.

On the server side, each user is associated with a salt to store password hashes securely. Upon receival of a password, the server verifies that:

$$SHA256(password \,//\, salt_{user}) \;=\; stored\_user\_hash$$

If the verification succeeds, the user is authenticated and the authentication protocol is completed.

3.1 Client Side Operations

Upon execution, the client executes the following operations:

1. Username Submission

- Prompt the user for a username (get_username)
- Send the username to the server (send_username_to_server)

- Await server response (ACK/NCK):
    - If NCK, authentication fails and the process terminates.

2. Nonce Generation and Submission

- Generate a random nonce (generate_nonce)
- Send the nonce to the server (send_nonce)

3. Diffie-Hellman Key Exchange

- Receive the server's Diffie-Hellman (DH) public key and the signature of the key with the provided client nonce
- Verify the server's signature using the server's public key (verify_signature)
- Send the client's DH public key to the server
- Derive the shared secret and hash it to obtain the session key
- All subsequent operations (e.g., password provisioning, signing, key generation) are securely encrypted using the session key

4. Password Submission

- Prompt the user for their password (get_password)
- Encrypt the password using the session key
- Send the encrypted password to the server (send_password_to_server)
- Await server response (ACK/NCK/CPW):
    - If NCK, authentication fails
    - If CPW, prompt for a new password and repeat the encrypted submission

3.2 Server Side Operations

Upon connection with a client, the server executes the following operations:

1. Username Reception and Validation

- Receive the username from the client (get_user_hello)
- Validate the username (e.g., existence and format)
- Send ACK or NCK to the client (send_ACK / send_NACK)
    - If NCK, terminate the authentication process

2. Nonce Reception
    - Receive the nonce from the client

3. Diffie-Hellman Key Exchange
    - Generate the server's DH key pair (send_signed_dh_params)
    - Sign the tuple (server_public_key, client_nonce) (sign)
    - Send the server's DH public key and the signature to the client (send_signed_dh_params)
    - Receive the client's DH public key
    - Derive the shared secret and hash it to obtain the session key
    - All subsequent operations (e.g., password provisioning, signing, key generation) are securely encrypted using the session key

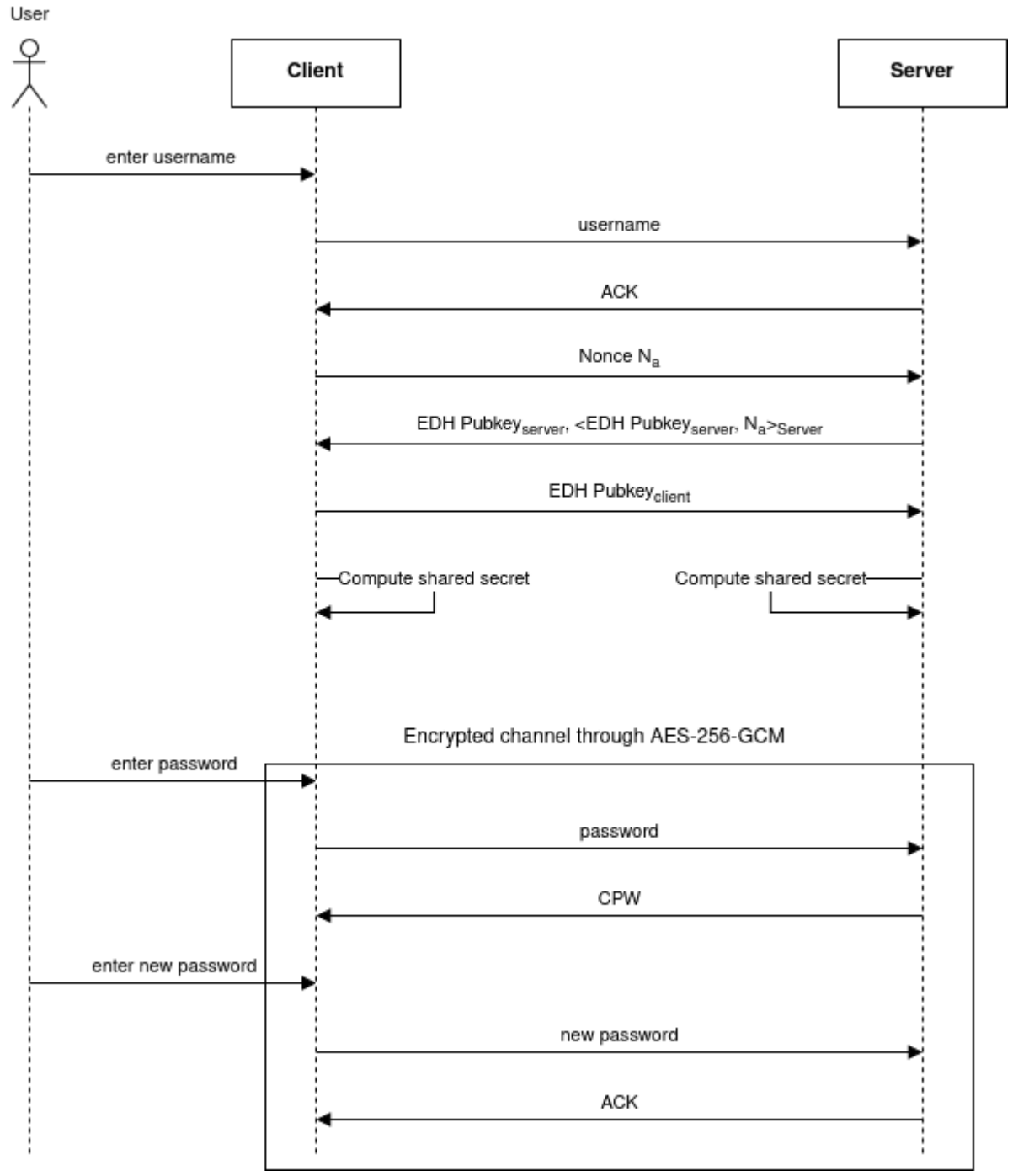
4. Password Reception and Validation
    - Receive the encrypted password from the client (get_encrypted_message)
    - Decrypt the password using the session key
    - Validate the password (is_password_valid)
        - If invalid, send NCK
        - If a password change is required, send CPW and wait for the new password

4 Sequence Diagrams Interaction

The following is a high level representation through sequence diagrams of the protocol describing the interactions between client and server.

4.1 Authentication protocol

Successful authentication (first login)



Message format

C -> S: [16 bytes] username

S -> C: [3 bytes] response = “ACK”

C -> S: [16 bytes] nonce

S -> C: [1620 bytes] $Pubkey_{server}$, $<Pubkey_{server} || nonce>_{Server}$

C -> S: [804 bytes] $Pubkey_{client}$

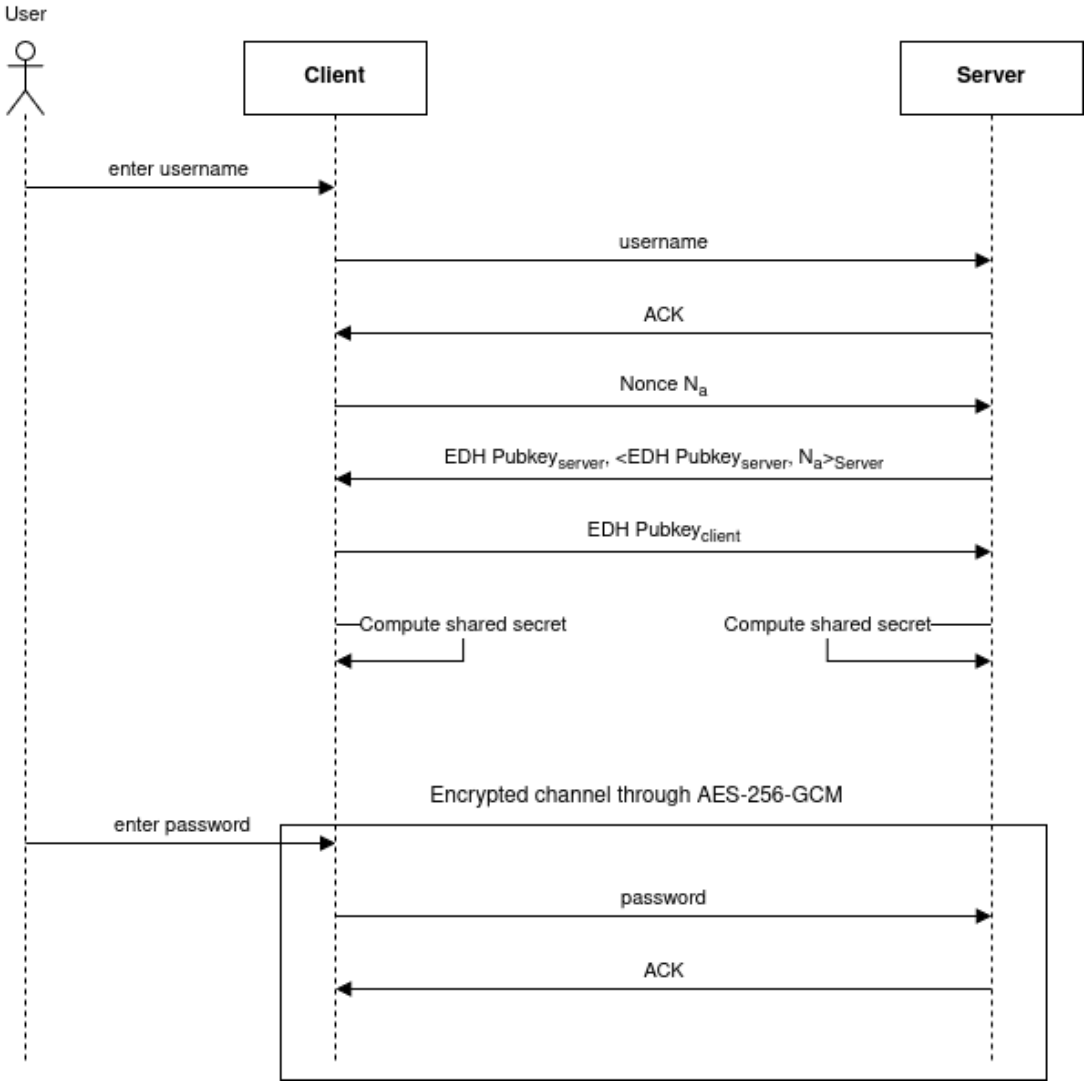
C -> S: [12 bytes][16 bytes][4 bytes][16 bytes] IV_1, tag_1, ct_{len}, E_k(password)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“CPW”)

C -> S: [12 bytes][16 bytes][4 bytes][16 bytes] IV_3, tag_3, ct_{len}, E_k(new_password)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_3, tag_4, ct_{len}, E_k(“ACK”)

Successful authentication (after first login)



Message format

C -> S: [16 bytes] username

S -> C: [3 bytes] response = “ACK”

C -> S: [16 bytes] nonce

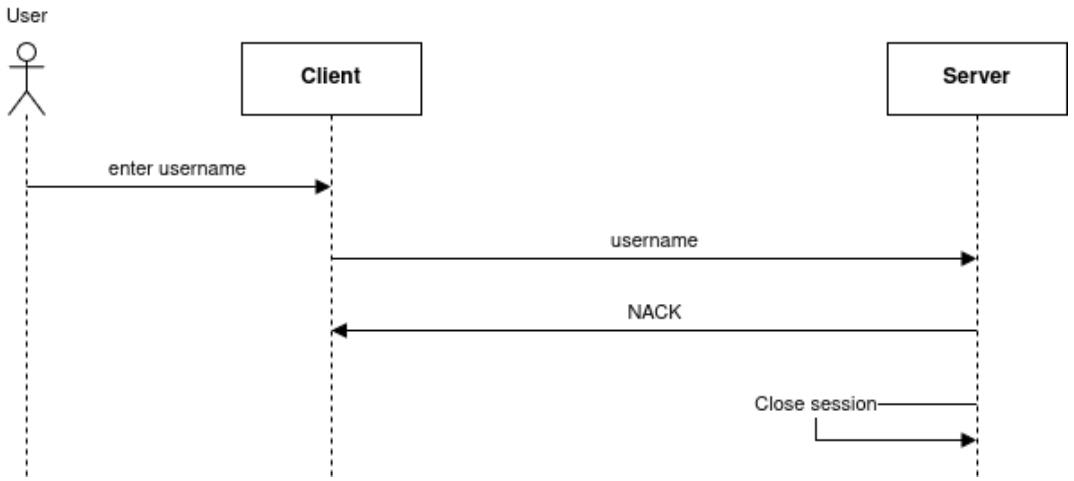
S -> C: [1620 bytes] $Pubkey_{server}$, $<Pubkey_{server} || nonce>_{Server}$

C -> S: [804 bytes] $Pubkey_{client}$

C -> S: [12 bytes][16 bytes][4 bytes][16 bytes] IV_1, tag_1, ct_{len}, E_k(password)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“ACK”)

Username not recognized

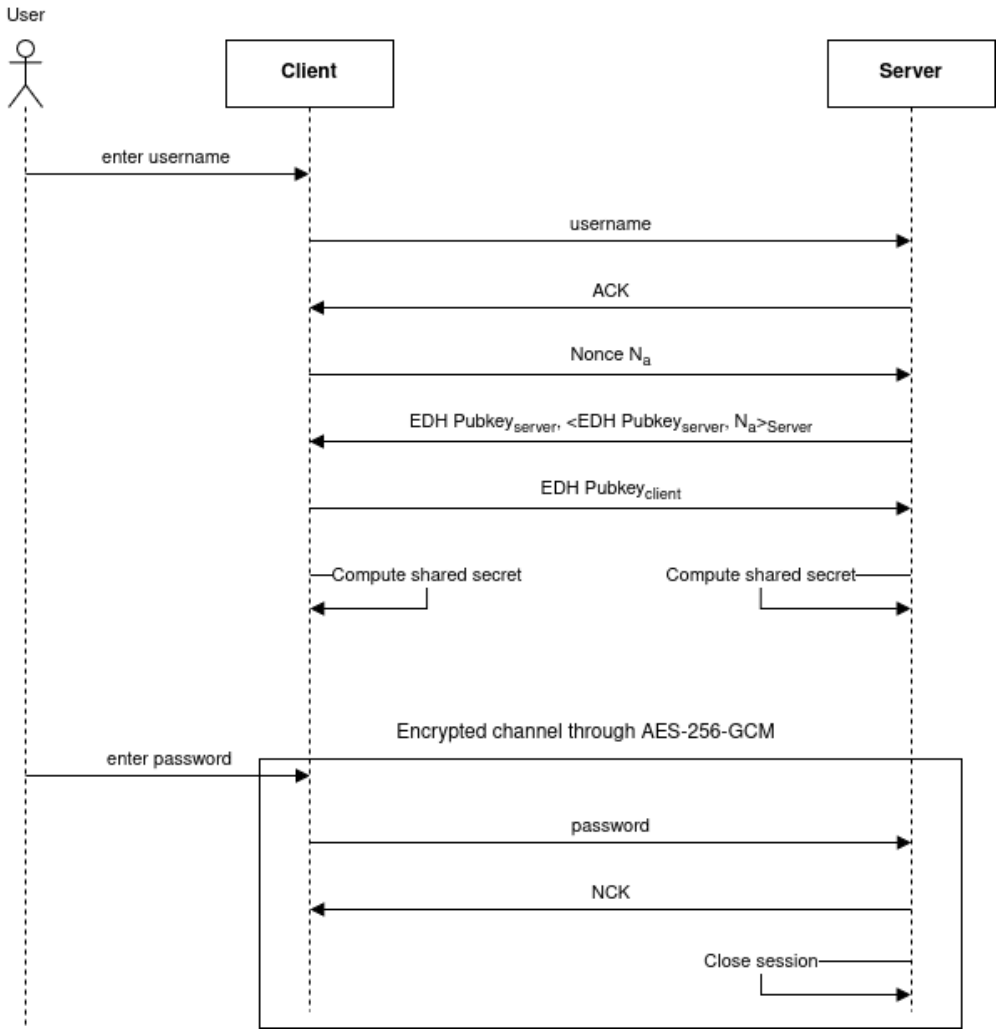


Message format

C -> S: [16 bytes] username

S -> C: [3 bytes] response = “NCK”

Wrong password



Message format

C -> S: [16 bytes] username

S -> C: [3 bytes] response = “ACK”

C -> S: [16 bytes] nonce

S -> C: [1620 bytes] $Pubkey_{server}$, <$Pubkey_{server}$ || nonce>$_{Server}$

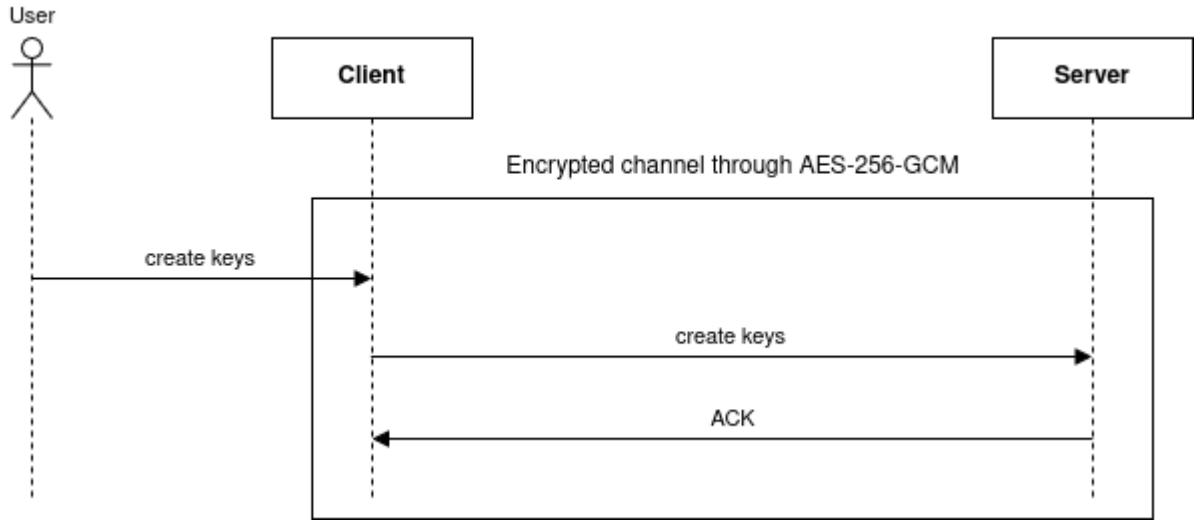
C -> S: [804 bytes] $Pubkey_{client}$

C -> S: [12 bytes][16 bytes][4 bytes][16 bytes] IV_1, tag_1, ct_{len}, E_k(password)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“NCK”)

4.2 Application protocol

Keys creation

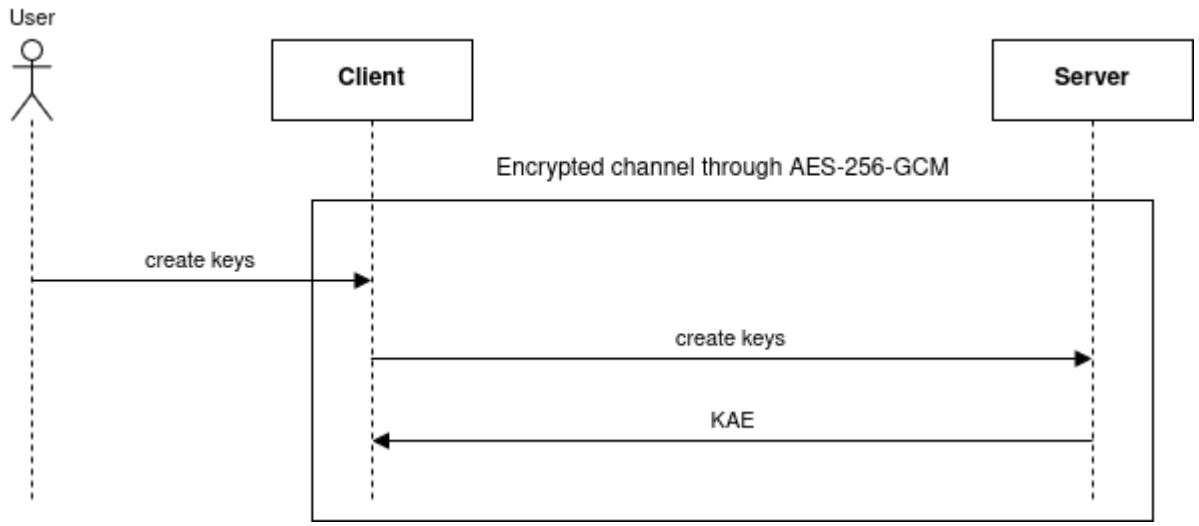


Message format

C -> S: [12 bytes][16 bytes][4 bytes][3 bytes] IV_1, tag_1, ct_{len}, E_k(“OP1”)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“ACK”)

Keys creation (keys already exist)

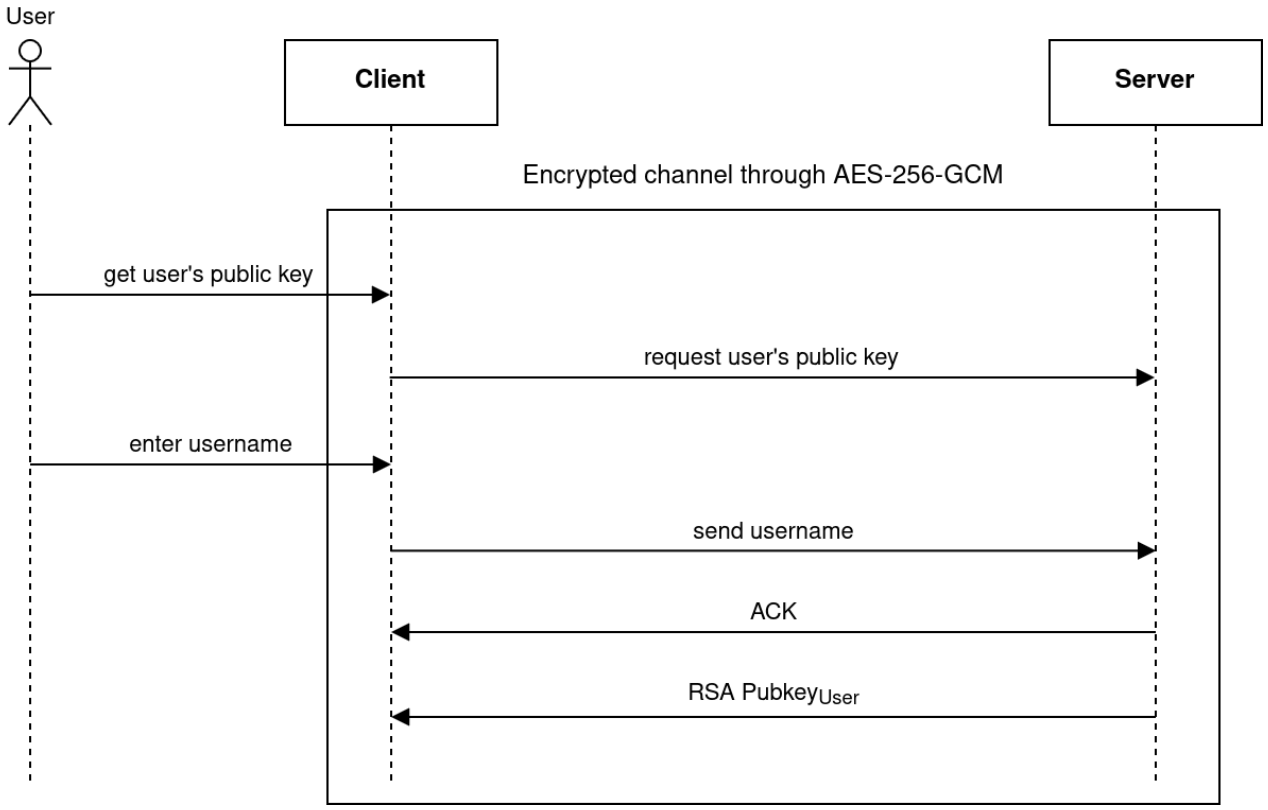


Message format

C -> S: [12 bytes][16 bytes][4 bytes][3 bytes] IV_1, tag_1, ct_{len}, E_k(“OP1”)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“KAE”)

Public Key retrieval



Message format

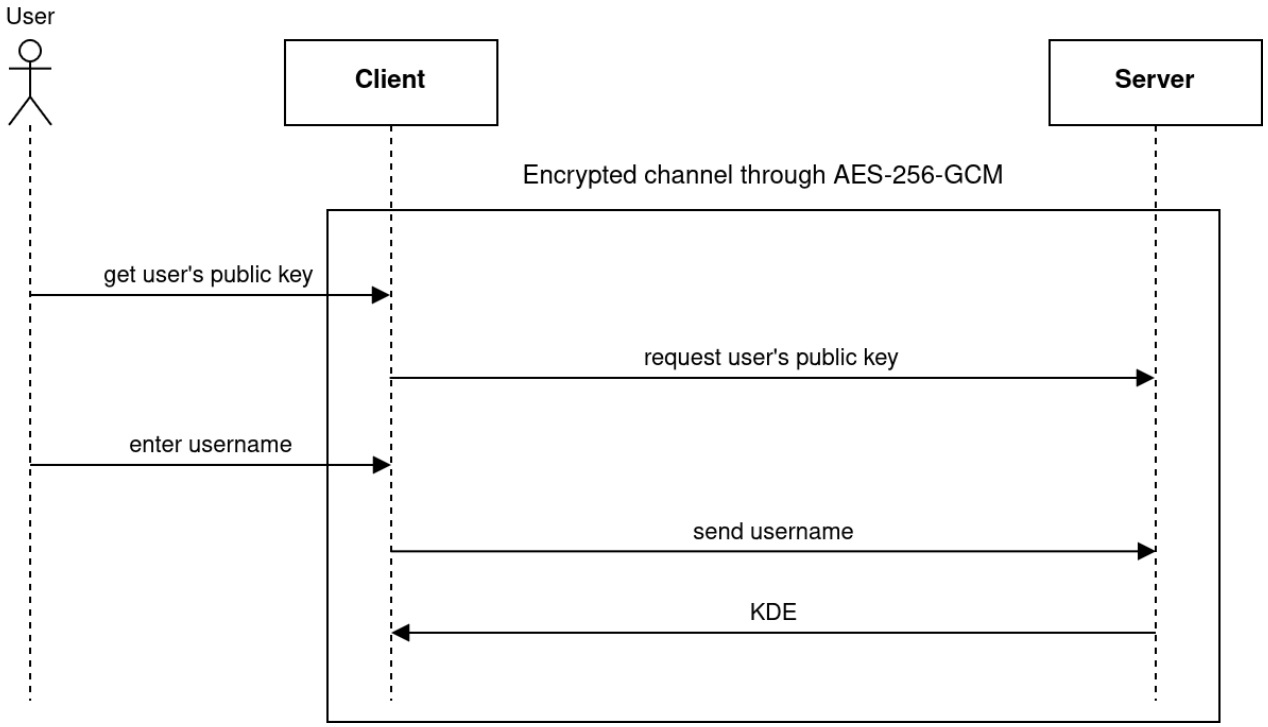
C -> S: [12 bytes][16 bytes][4 bytes][3 bytes] IV_1, tag_1, ct_{len}, E_k(“OP3”)

C -> S: [12 bytes][16 bytes][4 bytes][16 bytes] IV_2, tag_2, ct_{len}, E_k(username)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_3, tag_3, ct_{len}, E_k(“ACK”)

S -> C: [12 bytes][16 bytes][4 bytes][625 bytes] IV_4, tag_4, ct_{len}, E_k($Pubkey_{username}$)

Public Key retrieval (Key Doesn't Exist)



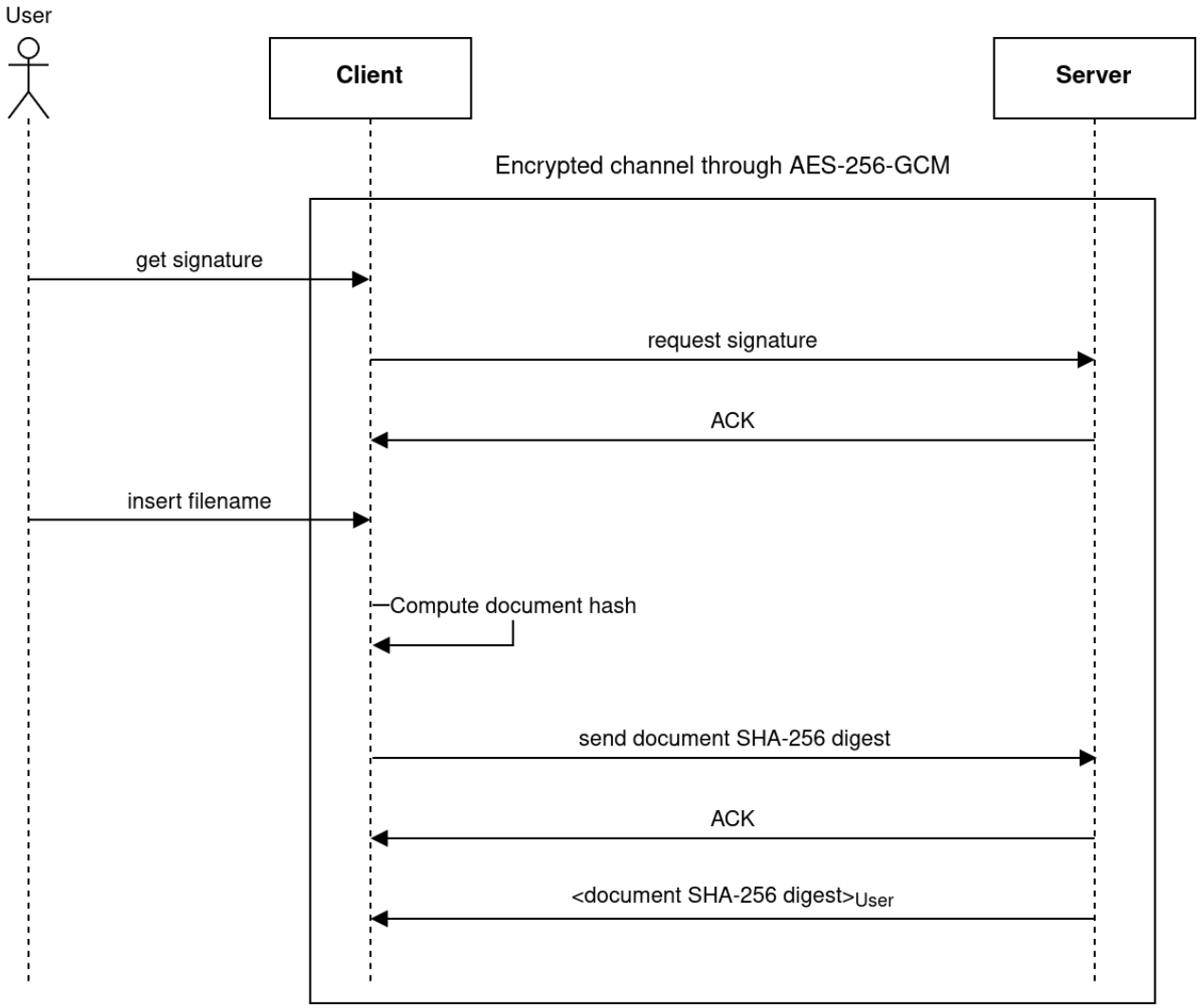
Message format

C -> S: [12 bytes][16 bytes][4 bytes][3 bytes] IV_1, tag_1, ct_{len}, E_k("OP3")

C -> S: [12 bytes][16 bytes][4 bytes][16 bytes] IV_2, tag_2, ct_{len}, E_k(username)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_3, tag_3, ct_{len}, E_k("KDE")

Signature Request



Message format

C -> S: [12 bytes][16 bytes][4 bytes][3 bytes] IV_1, tag_1, ct_{len}, E_k(“OP2”)

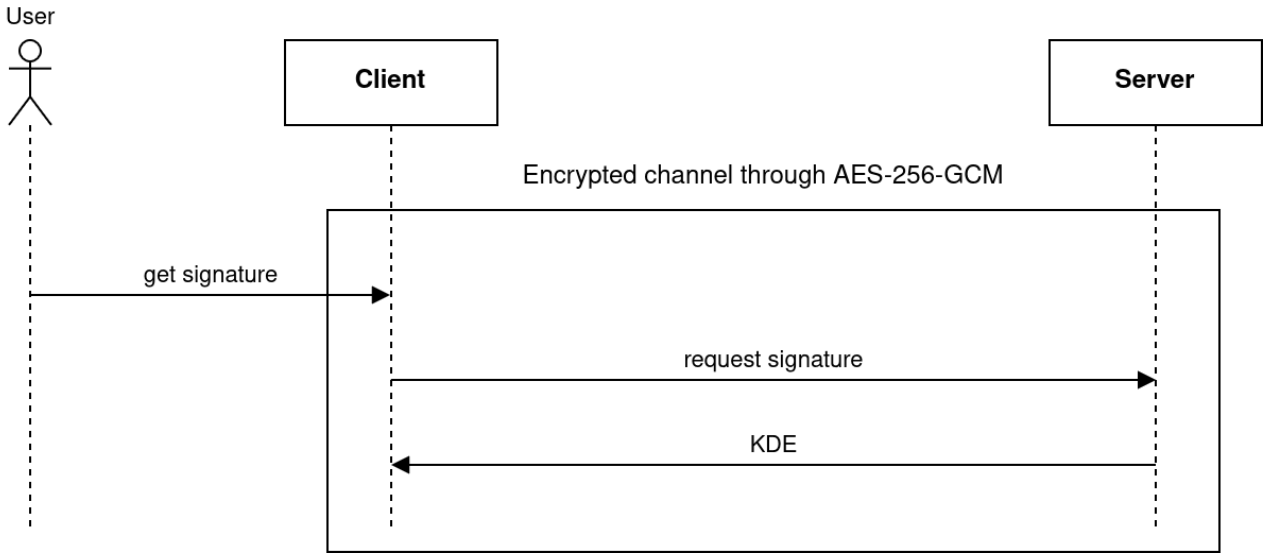
S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“ACK”)

C -> S: [12 bytes][16 bytes][4 bytes][32 bytes] IV_3, tag_3, ct_{len}, E_k(document hash)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_4, tag_4, ct_{len}, E_k(“ACK”)

S -> C: [12 bytes][16 bytes][4 bytes][384 bytes] IV_5, tag_5, ct_{len}, E_k(<document hash>$_{User}$)

Signature Request (Keys Don’t Exist)

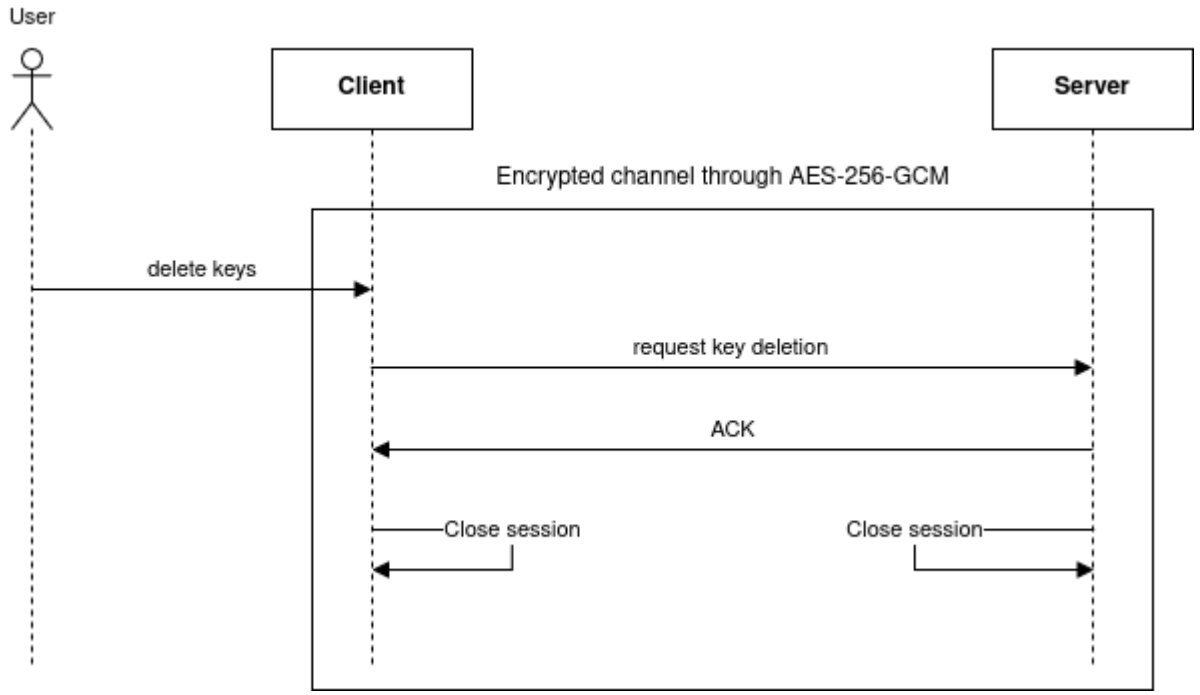


Message format

C -> S: [12 bytes][16 bytes][4 bytes][3 bytes] IV_1, tag_1, ct_{len}, E_k(“OP2”)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“KDE”)

Keys deletion

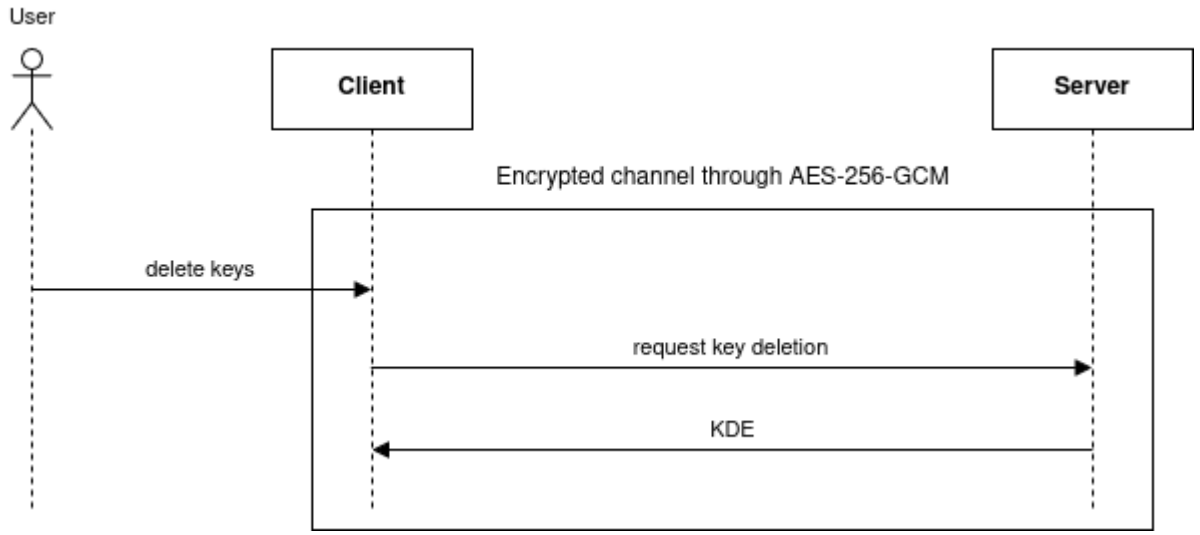


Message format

C -> S: [12 bytes][16 bytes][4 bytes][3 bytes] IV_1, tag_1, ct_{len}, E_k(“OP4”)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“ACK”)

Keys deletion (Keys Don’t Exist)

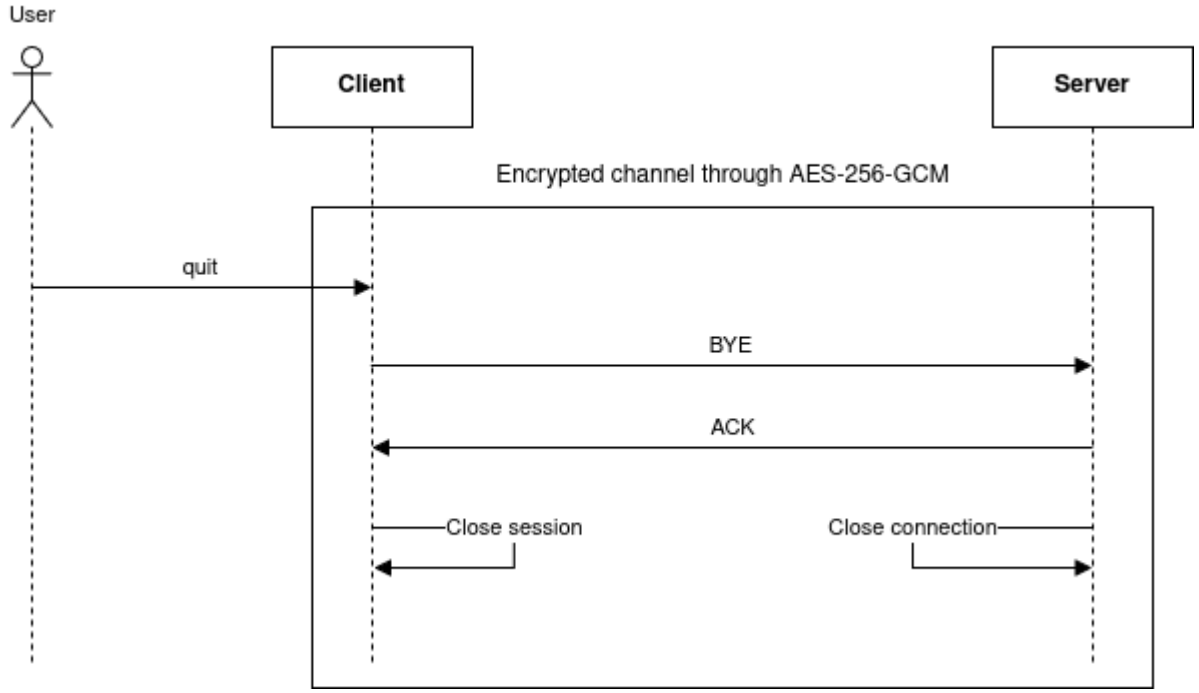


Message format

C -> S: [12 bytes][16 bytes][4 bytes][3 bytes] IV_1, tag_1, ct_{len}, E_k(“OP4”)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“KDE”)

Session closure



Message format

C -> S: [12 bytes][16 bytes][4 bytes][3 bytes] IV_1, tag_1, ct_{len}, E_k(“BYE”)

S -> C: [12 bytes][16 bytes][4 bytes][3 bytes] IV_2, tag_2, ct_{len}, E_k(“ACK”)

5 Conclusions

In this report, we presented a complete and secure implementation of a Digital Signature Service (DSS) tailored for organizational use. The proposed architecture ensures confidentiality, integrity, and authenticity of all interactions between the client and the server through the integration of modern cryptographic primitives and secure communication protocols.

The service design adheres to best practices in cryptographic engineering, including the use of Ephemeral Diffie-Hellman (EDH) for session key derivation, RSA-3072 for digital signatures, and AES-256-GCM for authenticated encryption. The authentication protocol guarantees mutual trust establishment and resists replay and tampering attempts, aligning with the project's goals of achieving perfect forward secrecy and non-malleability.

By supporting core operations such as key generation, public key retrieval, document signing, and key deletion, the system offers a robust foundation for secure digital identity management. The format of exchanged messages and the provided sequence diagrams clarify protocol behaviors and ensure reproducibility.

Overall, the DSS demonstrates a practical and secure approach to managing digital signatures in a controlled and auditable environment, with potential for real-world deployment and future extensions, such as certificate integration or audit logging mechanisms.