Digital Signature Server Report

Foundations of Cybersecurity Project

Giovanni Neglia mat.579821 Matteo Giuffrè mat.678007

June 2025

1 Introduction

The project implements a **Digital Signature Service (DSS)**, a trusted third-party system that manages cryptographic key pairs to generate digital signatures on the behalf of users (organization employees).

1.1 Tools

The system was developed using the following tools and technologies:

- **Python 3.12**: primary programming language used for both client and server implementation.
- **cryptography package**: used to implement key exchanges, encryption/decryption, digital signatures and session key derivation.
- **bcrypt package**: used to provide hashing and secure password verification.
- **mysql-connector-python package**: used for MySQL database connectivity.
- **MySQL Server**: used to store permanent data.

2 Project Specifications

The DSS provides the following functionalities:

- Create a user and log into the service.
- Generate and store an asymmetric key pair.
- Digitally sign documents.
- Retrieve user public key.
- Delete key pair.

Users authenticate via passwords and interact with the DSS through a secure channel that ensures **Perfect Forward Secrecy (PFS)**, **data integrity**, **replay attack protection** and **non-malleability**.

2.1 Design choices

Mutual Authentication The authentication protocol consists of:

- **Initial Handshake**:
  1. The client generates an ephemeral **Elliptic-Curve-Diffie-Hellman (ECDH) key pair** and sends the public one, *epkC*, along with a **nonce**, *nonceC*, to the server.
  2. The server generates its ephemeral ECDH key pair and responds with: *epkS*, *nonceS*, and an **Elliptic Curve Digital Signature Algorithm (ECDSA)** signature over *(epkS || epkC || nonceC || nonceS)* using the server private key stored in the file *dss_private.pem*
  3. The client verifies the signature using the **DSS public key**.

4. A session key that will be used with **Advanced Encryption Standard Galois/-Counter Mode (AES-GCM)** is derived using **HMAC-based Key Derivation Function (HKDF)**

- **User Authentication**:
  1. The users sends *username* and *password* encrypted with AES-GCM using the session key established.
  2. If it is the first login, the user must change their password.

CreateKeys generation The service checks whether the user already has a key pair or if they have deleted them. If this is not the case, generates a new **Elliptic-curve cryptography (ECC)** key pair and saves them in the database (private key encrypted with a **passphrase** using AES-256, public key in plaintext).

SignDoc operation The service attempts to retrieve the user's private key from the database. If the key is found, decrypts it using the passphrase and then signs the documents using ECDSA-SHA256.

DeleteKeys function Key pairs erasure happens setting the keys values to *null* and updating the *key_del* flag inside the DB.

User registration To register a new user, the service generates a temporary password in the format *"usernametempXXXX"* where "XXXX" is a random number. Then, computes the **bcrypt** hash of the password and stores the username and hash in the database, along with setting *is_new = 1*. Once logged in, the user will be asked to update his password and requested to perform a "regular" login.

Perfect Forward Secrecy Each session uses a unique ephemeral ECDHE key.

Nonce and session keys Nonces and session keys expire after 30 minutes to prevent replay attacks.

Message format After the handshake, all messages are sent in JSON format and encrypted with AES-GCM using the session key established.

2.2 Exchanged messages format

Initial Handshake	
Direction	Data
Client → Server	epkC (client's ECDH public key) + nonceC (16 random bytes)
Server → Client	epkS (server's ECDH public key) + nonceS (16 random bytes) + ECDSA signature (skDSS, epkS || epkC || nonceC || nonceS)

Table 1: Initial handshake process messages format

Login	
Direction	Data (AES-GCM encrypted using session key)
Client → Server	`{"username": str, "password": str, "nonceC": hex}`
Server → Client	`{"login_result": bool, "change_pwd": bool, "nonceS": hex}`

Table 2: Login operation messages format

Generic DSS Operations	
Direction	Data (AES-GCM encrypted using session key)
Client → Server	`{"service": int, "username": str, "nonceC": hex, //Additional fields per service}`
Server → Client	`{"status": bool, "nonceS": hex, //Additional fields }`

Table 3: Other DSS operations messages format

2.3 Communication protocol sequence diagrams

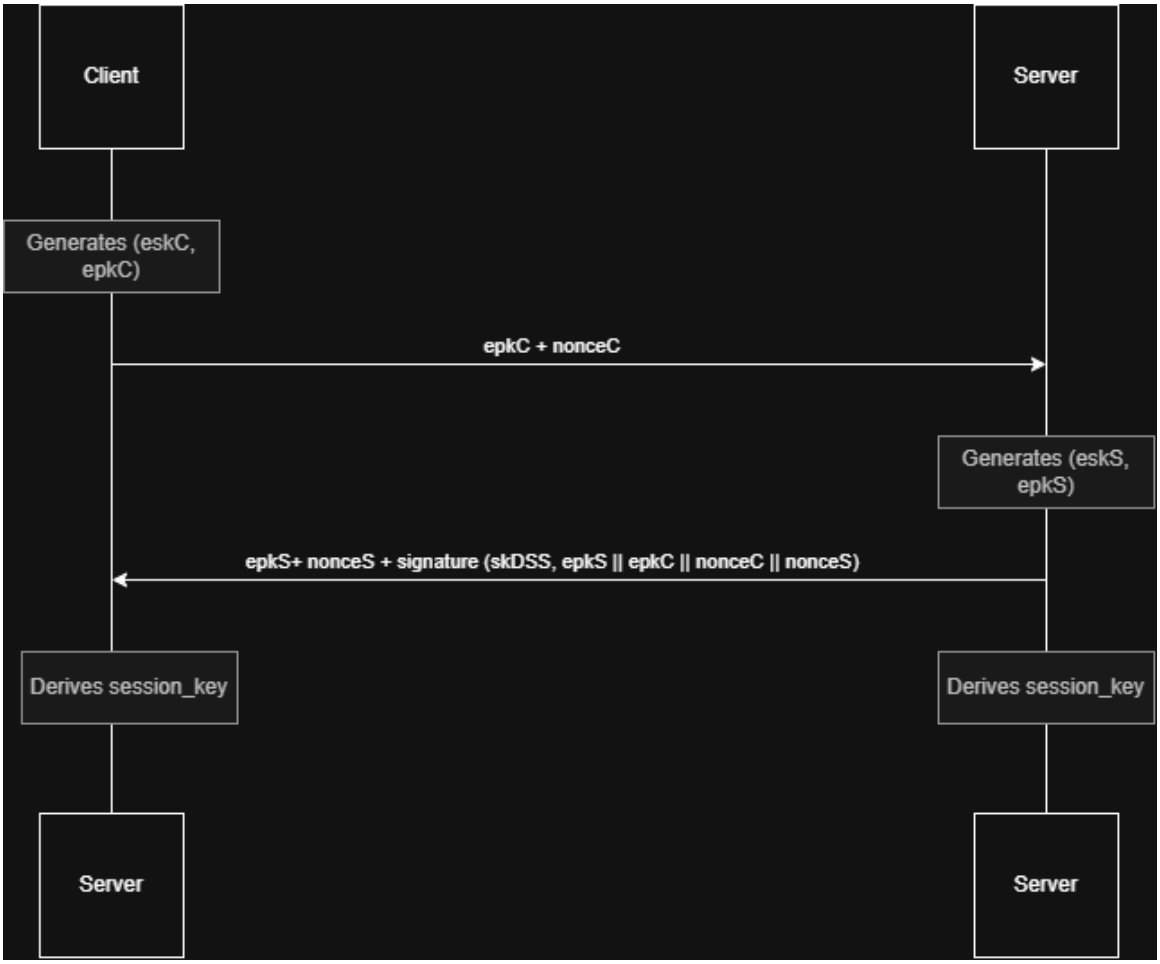

Figure 1: Handshake sequence diagram

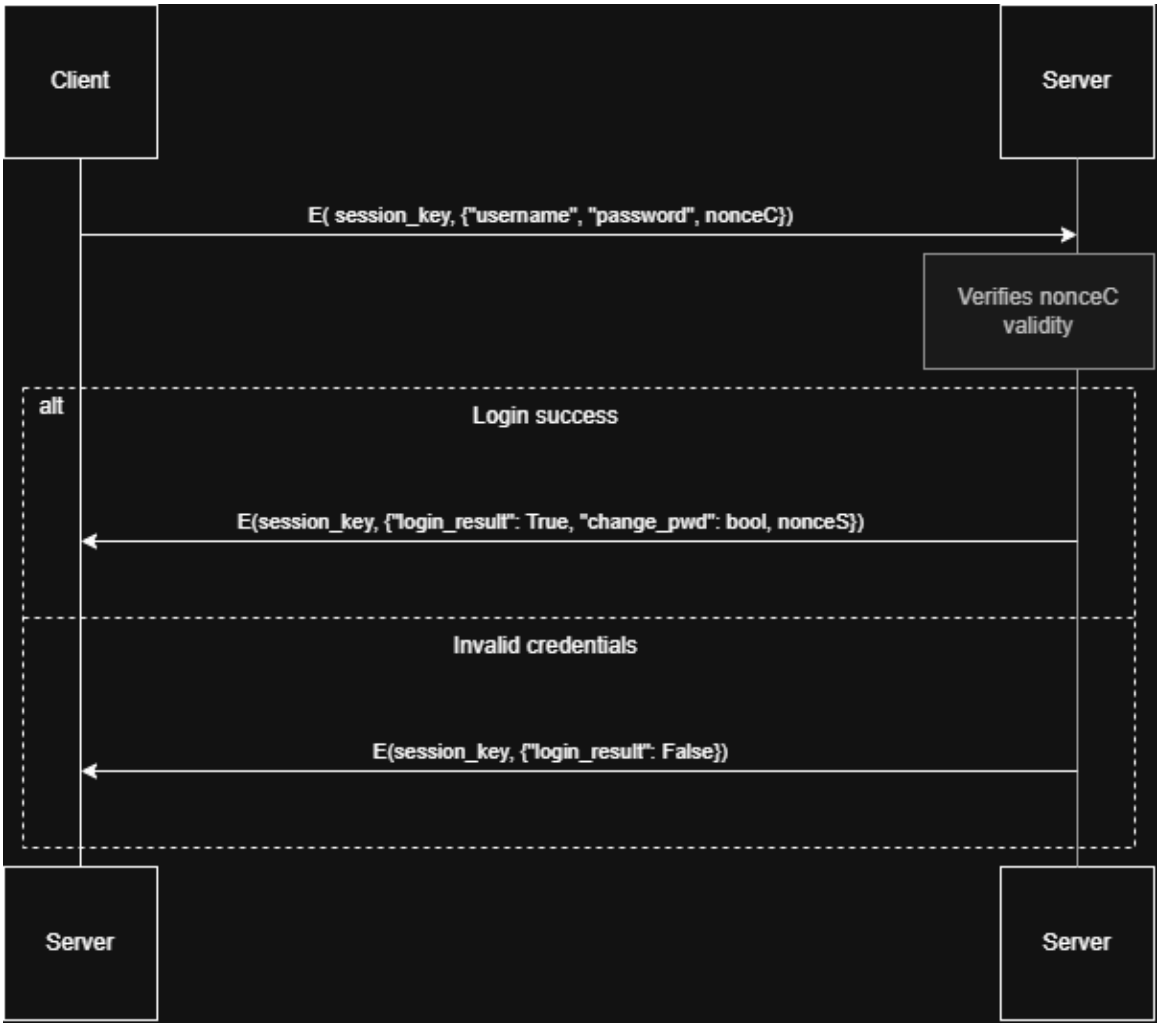

Figure 2: Login sequence diagram

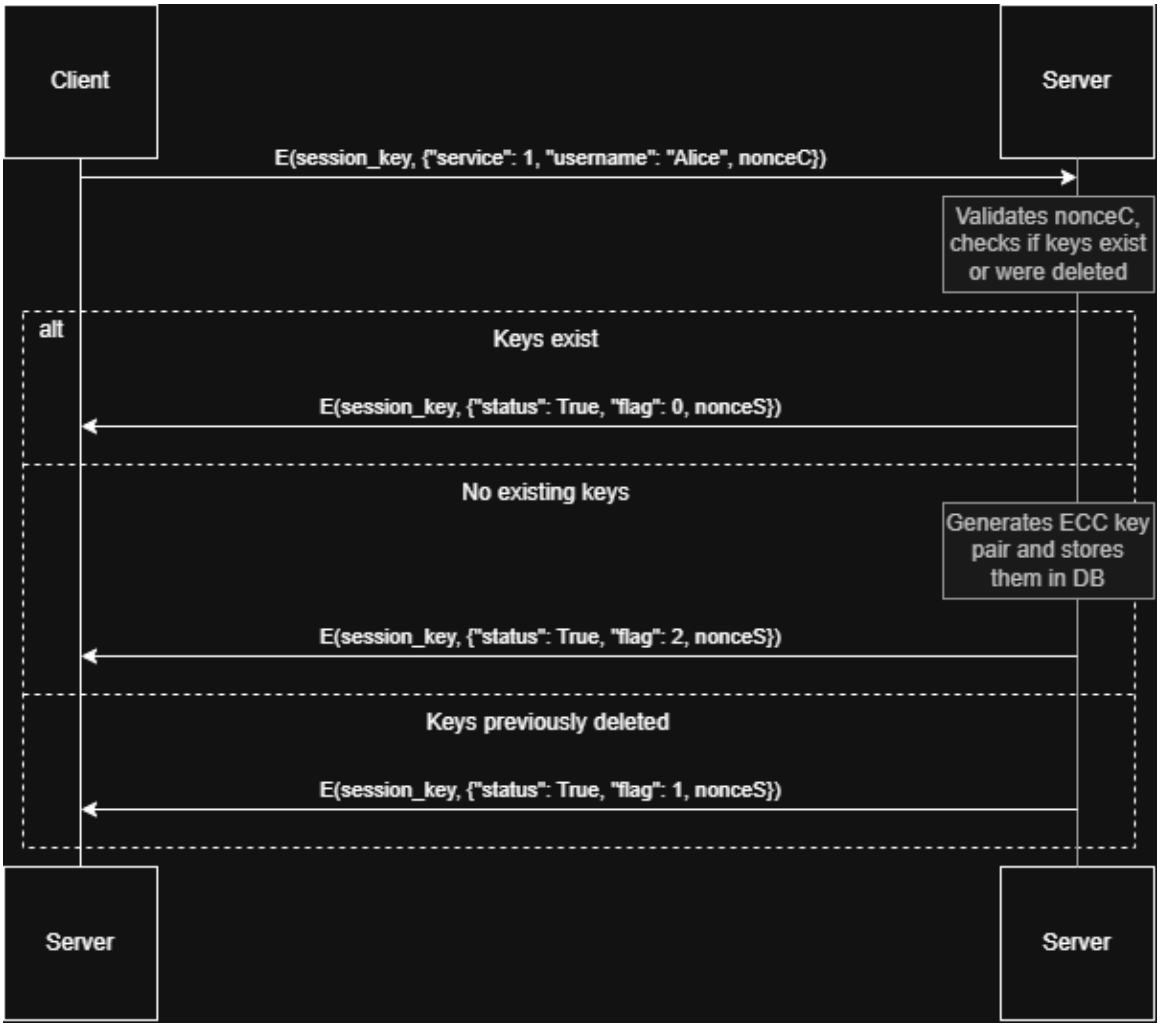

Figure 3: CreateKeys sequence diagram

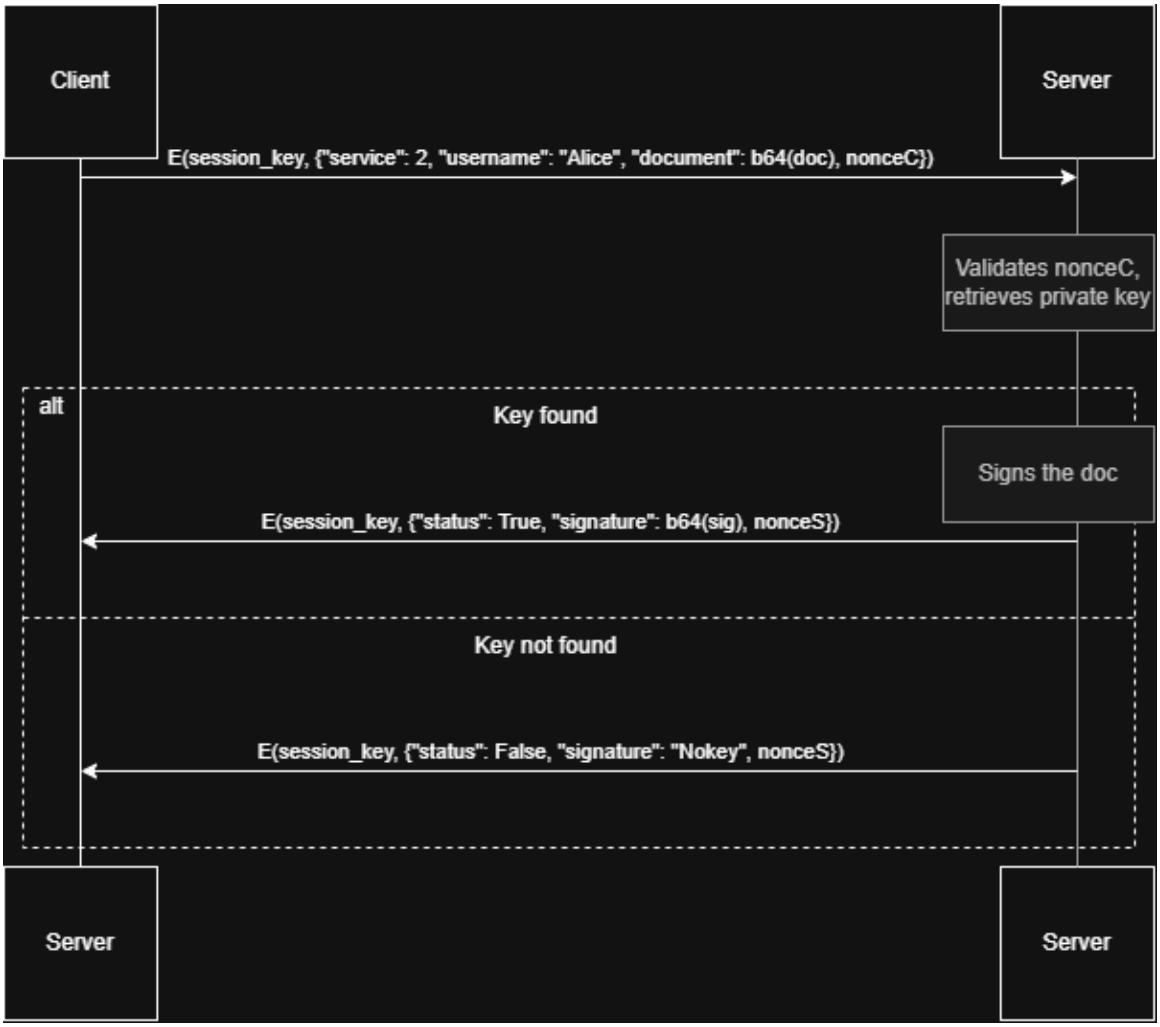

Figure 4: SignDoc sequence diagram

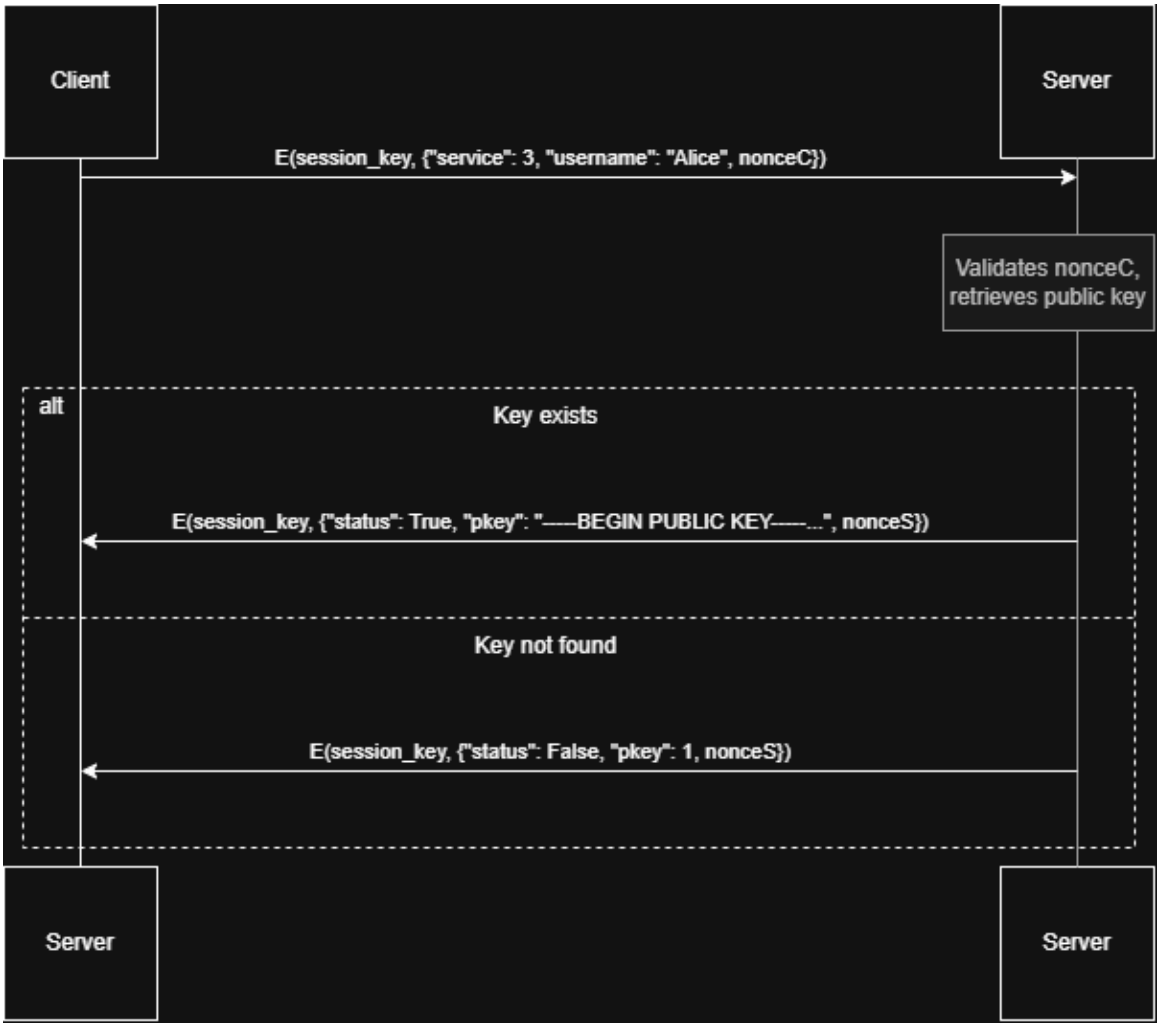

Figure 5: GetPublicKey sequence diagram

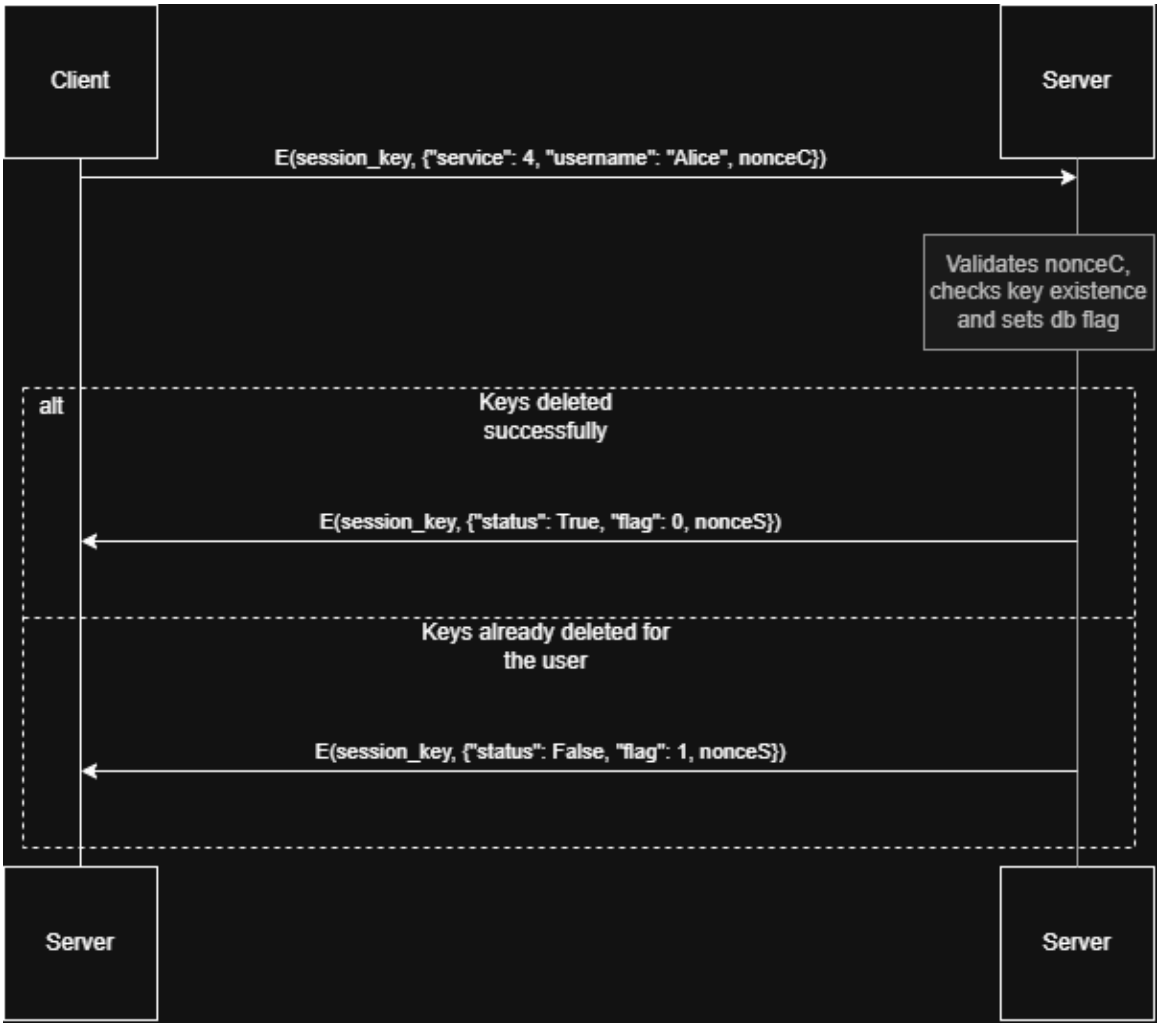

Figure 6: DeleteKeys sequence diagram