UNIVERSITY OF PISA

MSc in Computer Engineering

Project for Foundations Of Cybersecurity

Digital Signature Server

Professors:

Prof. Gianluca Dini

Students:

Francesco Barcherini (645413)

Antonio Ciociola (645324)

ACADEMIC YEAR 2024/2025

Contents

1 Introduction

The Digital Signature Service (DSS) consists of two main components: the client and the server. The server is responsible for managing multiple connections with clients, each of which represents a registered employee. The service is designed to ensure secure communication between users and the DSS while guaranteeing confidentiality, integrity, authentication, and perfect forward secrecy (PFS).

Before initiating any operation, a secure session must be established between the client and the server. This is achieved through an ephemeral Diffie-Hellman key exchange protocol, during which the client and server exchange their respective DH public keys. To prevent man-in-the-middle attacks and ensure authentication, the exchange is authenticated using the digital signature provided by the server. The client, in turn, is authenticated after the secure channel is established by providing their password over the encrypted connection.

Upon successful completion of the DH exchange, both the client and the server derive a shared secret, from which a symmetric session key is generated. This key, referred to as `shared_key`, is subsequently used to encrypt and decrypt all application-level messages using the AES-GCM (Galois/Counter Mode) authenticated encryption scheme.

The DSS server enables users to create and manage their own cryptographic keys, and to request digital signatures on documents. This functionality is implemented using RSA key pairs, where each user's private key is encrypted and protected by a user-defined passphrase. Such architecture allows users to delegate signing operations to the server without compromising key confidentiality.

All employee information is stored securely on the server. In particular, user passwords are never stored in plaintext: instead, they are hashed using the SHA-256 algorithm in combination with a unique cryptographic salt. This approach ensures resilience against dictionary attacks and safeguards user credentials in the event of a data breach.

2 Cryptographic Protocols

2.1 Secure Channel Establishment

To ensure confidentiality, integrity, and authentication of the exchanged messages, the DSS system establishes a secure channel between the client and the server using a hybrid protocol that combines the Diffie-Hellman key exchange with RSA-based digital signatures.

The protocol proceeds as follows:

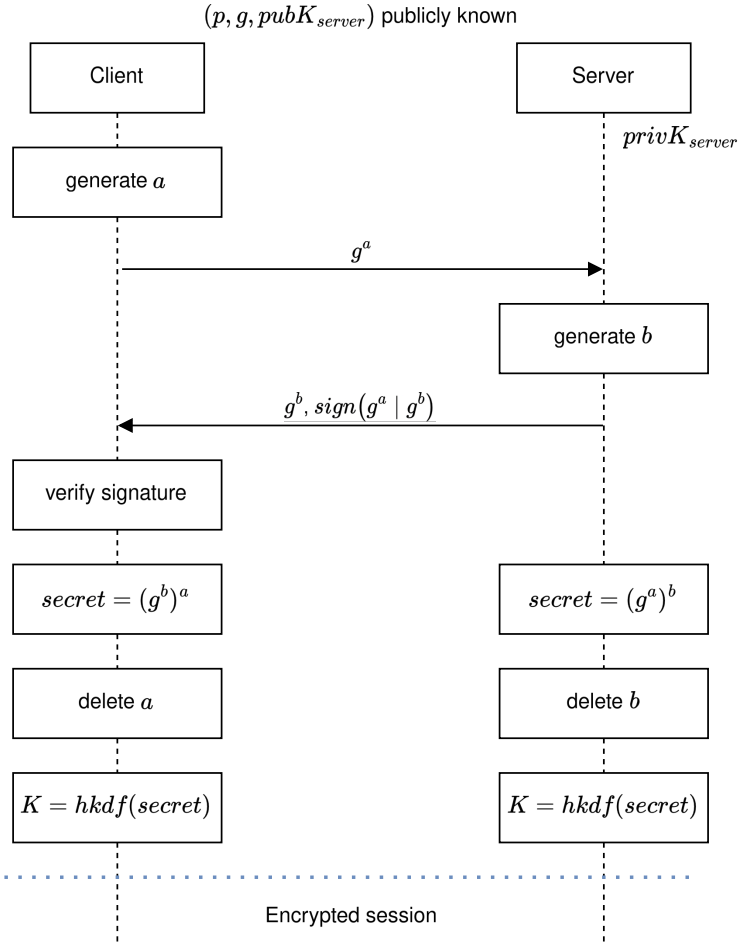

Figure 1: Diffie-Helman key exchange protocol

1. The client and server agree on a common cyclic group with generator g and a large prime modulus p. These parameters are assumed to be known to both parties.
2. The server's RSA public key is precomputed using a 2048-bit modulus. The corresponding private key is securely stored on the server. The RSA key pair can be generated using the following `openssl` commands:
   - `openssl genrsa -out rsa_privkey.pem 2048`
   - `openssl rsa -pubout -in rsa_privkey.pem -out rsa_pubkey.pem`

   The client is provisioned with the server's public key in advance.
3. The client generates a private Diffie-Hellman exponent $a \in \mathbb{Z}_p$, computes its public value $g^a \mod p$, and sends it to the server.
4. Upon receiving g^a, the server generates its own private exponent $b \in \mathbb{Z}_p$, computes its public value $g^b \mod p$, and prepares to authenticate itself.
5. To authenticate the exchange, the server computes a digital signature over the concatenation of the two DH public values, i.e., $\text{Sign}_{\text{RSA}}(g^a \| g^b)$, using its private

RSA key. This step ensures both integrity and origin authentication.

6. The server replies with its DH public key g^b and the digital signature $\sigma = \text{Sign}_{\text{RSA}}(g^a \| g^b)$. Including both values in the signature prevents man-in-the-middle attacks where an attacker could reuse a valid signature over a single DH component.
7. The client receives g^b and the signature σ. It verifies the signature using the known RSA public key of the server. If the verification succeeds, the authenticity of the server is established.
8. Both client and server compute the shared secret $s = g^{ab} \mod p$ using their respective DH private and the peer's public key.
9. The shared session key is derived from the secret s using HKDF.

This protocol guarantees mutual key agreement and provides perfect forward secrecy, while authentication is ensured via digital signatures. All subsequent messages between the client and server are encrypted using the symmetric key `K` with the authenticated encryption scheme (AES-GCM).

2.2 Encrypted Session

The exchange protocol of the encrypted session is described in the figure below:

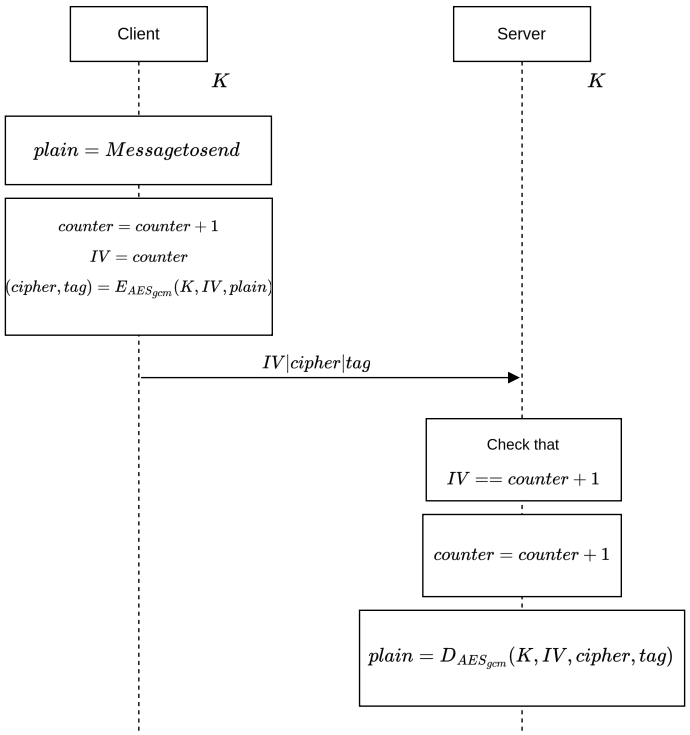

Figure 2: Symmetric encryption exchange protocol

Once the secure channel is established and the symmetric session key K is derived, all subsequent communication between client and server proceeds using symmetric

authenticated encryption. The encryption scheme adopted is AES in Galois/Counter Mode (AES-GCM) with a 256-bit key, which ensures both confidentiality and integrity of the messages.

Each message is encrypted independently. The sender—depicted as the client in the diagram, though it could equally be the server—prepares the plaintext message and encrypts it using AES-GCM with the shared key K and a unique Initialization Vector (IV). The IV is implemented as a strictly increasing counter, incremented by one for each message sent. This counter-based IV design provides protection against replay attacks, as receiving parties can detect and discard messages with non-increasing IVs.

To prevent IV reuse the implementation includes a mechanism to monitor the counter for overflow. Upon detecting a potential overflow, the session is terminated to avoid any security breach due to IV reuse.

Every byte related to secrets (e.g. the shared key, plaintexts, passwords etc.) are nulled when they are not necessary anymore. In this way their values are deleted from memory when the corresponding data structures are deallocated.

2.2.1 User Authentication and Password Management

The user authenticates to the service by means of a password. The password of the first login is created off-line. At first login the user changes his password. The diagram of this operation is the following:

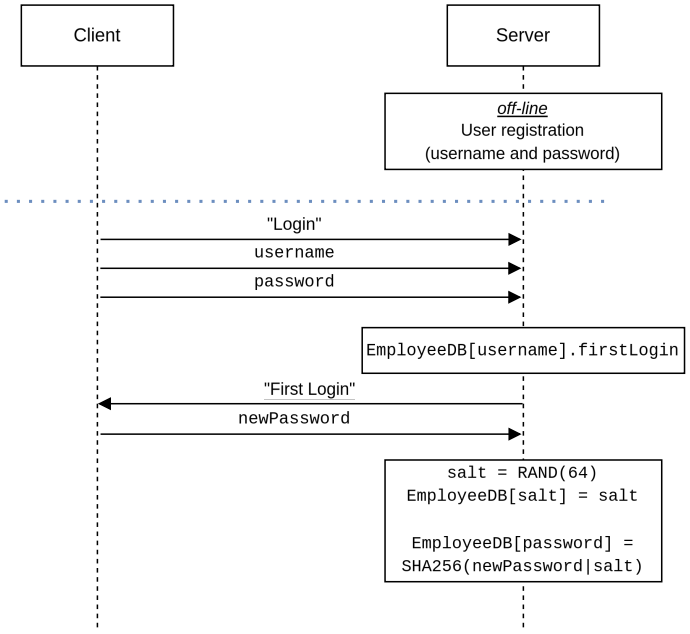

Figure 3: User authentication protocol

The server maintains, for each registered user, a set of attributes:

- **Username**;

- **Salt:** a random value of 64 bit;
- **Password hash:** the SHA-256 hash of the concatenation `password|salt`. This construction ensures that identical passwords across users result in different hashes, and avoids the need to store plaintext passwords;
- **RSA key pair:** the user's RSA-2048 public and private key pair, stored in PEM format, used for digital signature operations;
- **First login flag:** a boolean flag indicating whether the user has logged in at least once and changed the initial off-line generated password;
- **Key presence flag:** a boolean flag indicating whether the user currently possesses a valid RSA key pair;
- **Key deletion flag:** a boolean flag indicating whether the user has permanently deleted his key pair.

During authentication, the user submits their username and password to the server. The server retrieves the stored salt associated with the username and computes the SHA-256 hash of the received password concatenated with the salt. Authentication succeeds if and only if the computed hash matches the stored password hash.

2.2.2 User's Keys and Digital Signature

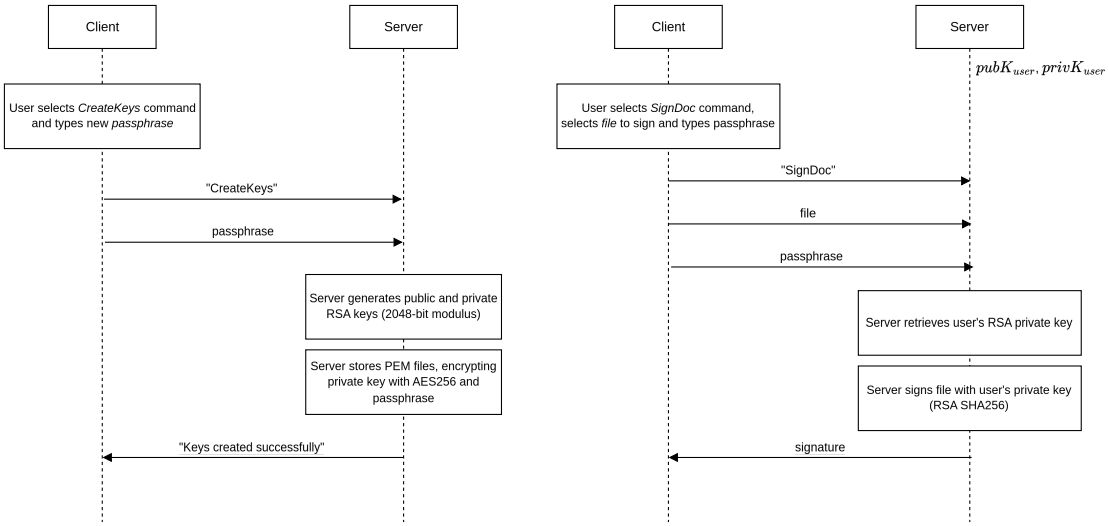

Figure 4: User's RSA keys and SHA256 digital signature

Each user is associated with a personal RSA key pair with 2048-bit modulus, generated and stored on the server side. To ensure confidentiality of the private key, it is encrypted with a passphrase provided by the user at the moment of key generation. The encrypted private key is stored in PEM format.

Once the key pair is created, the user's public key becomes accessible to all authenticated users through the `GetPublicKey` operation. This operation reads the corresponding PEM file and returns the public key as a string.

The user retains the option to delete their key pair at any time. Upon deletion, the associated flags are updated, and the keys are permanently removed from the server. In such cases, the user loses the ability to perform digital signature operations unless a new key pair is generated via a offline registration process.

To digitally sign a file, the user issues a signing request to the server. This request includes the content of the document to be signed and the passphrase corresponding to the encrypted RSA private key in PEM format.

Upon receiving the request, the server performs the following steps:

1. decrypts the private key using the provided passphrase;
2. computes the SHA-256 digest of the input document;
3. generates the digital signature by applying the RSA private key to the digest;
4. returns the signature to the user.

2.3 Secure Channel Properties

To summarize, the secure channel provides the following guarantees:

- **Perfect Forward Secrecy**, ensured by using ephemeral Diffie-Hellman key exchange.
- **Integrity and Non-Malleability**, ensured by employing AES-GCM authenticated encryption.
- **Replay Protection**, ensured by using unique initialization vector (IV) for each message.

3 Exchanged Messages

3.1 Key Exchange Format

The first message format is used during the initial channel establishment phase, specifically during the Diffie-Hellman key exchange. This message carries the public Diffie-Hellman values and the server's digital signature to authenticate the exchange.

Figure 5: Format of the initial message used for Diffie-Hellman key exchange and server authentication

3.2 Encrypted Session Format

The second message format defines the structure of each message exchanged within the encrypted session.

Before encryption, the plaintext is padded using PKCS#7 padding to conceal the true length of the message. Specifically, messages are padded to the next multiple of 64 bytes. This padding strategy provides a balance between security and efficiency:

- Padding to 64-byte blocks prevents leaking the exact message length, which is essential as AES-GCM operates in a stream cipher mode and otherwise exposes length information.
- The 64-byte block size is chosen because all messages are typically shorter than 64 bytes, so padding them to a uniform size prevents leaking information about their exact length.
- At the same time, the padding is not so large as to unnecessarily waste bandwidth for small or frequent messages.

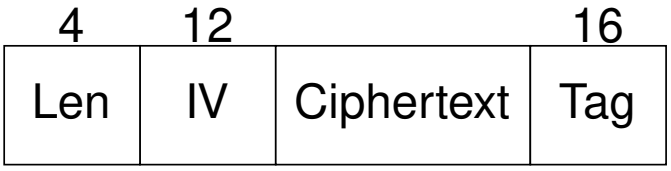

Figure 6: Format of each encrypted message exchanged within the secure session