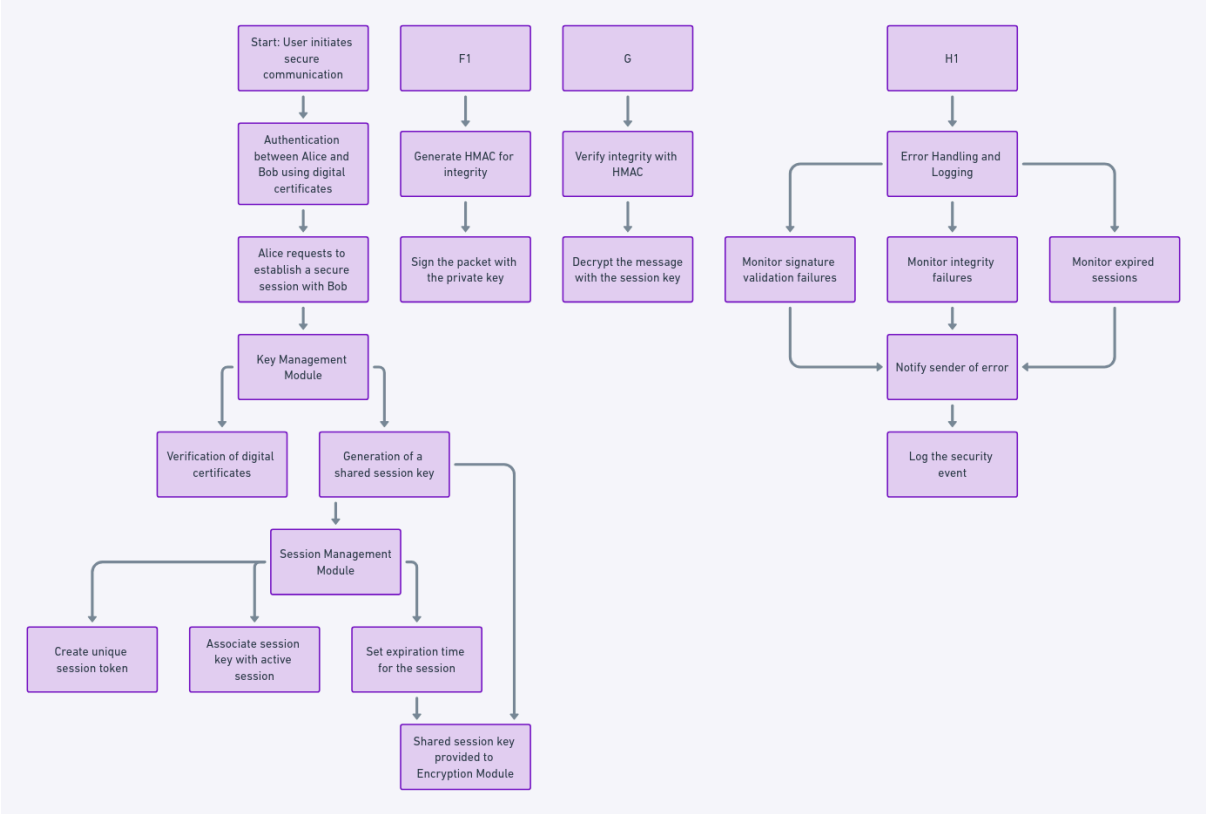
System Architecture



Modules

Key Management

This module should be responsible for:

1. Generate secure keys (asynchronous and session keys).
2. Exchange keys using a secure protocol such as Diffie-Hellman.
3. Securely store and rotate keys.

Pseudocode:

1. Generate private key:

- Use a cryptographic library to generate an RSA/ECC private key.

- Export the associated public key.

2. Exchange public key:

- Transmit Alice's public key to Bob.

- Receive Bob's public key.

3. Generate session key:

- Use Diffie-Hellman or ECC to generate a shared session key.

- Derive the final key using a secure hash (SHA-256).

4. Store keys:

- Store private keys in an HSM or secure storage.

5. Rotate keys:

- Every X messages or after Y time, regenerate session keys.

Session Management

This module is responsible for:

1. Create unique tokens per session.
2. Associating session keys with tokens.
3. Set and verify expiration times.

Pseudocode:

1. Generate a session token:

- Create a unique identifier (UUID).

- Associate the token with the generated session key.

2. Verify the validity of the session:

- Check that the token is still active.

- Validate that the session has not expired.

3. Expire session:

- Delete the token and associated key if the session has expired.

Encryption/Decryption

This module implements:

1. Message encryption with AES-256 GCM.
2. Generation and verification of HMACs for integrity.
3. Message decryption and integrity validation.

Pseudocode:

1. Message encryption:

- Use AES-256 GCM to encrypt the message.

- Generate an HMAC to ensure integrity.

2. Message decryption:

- Verify the HMAC.

- If valid, decrypt the message.

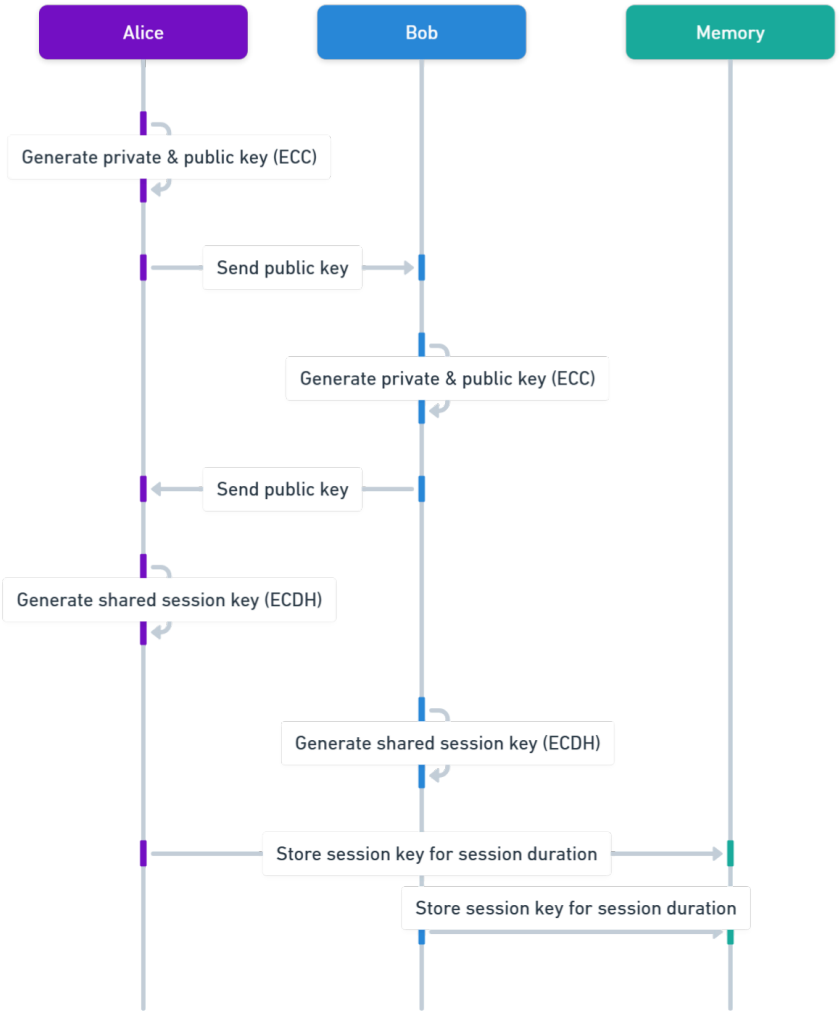
Identified Threats and Solutions

Vulnerability	**Solution**
Man-in-the-Middle (MITM) attack	Validate public keys with digital certificates signed by a CA (Certificate Authority).
Replay Attacks	Include a unique nonce and timestamps in each message to prevent messages from being reused.
Private Key Compromise	Store keys in HSM (Hardware Security Module) and rotate them periodically.
Message Manipulation	Use HMAC (e.g., HMAC-SHA256) to verify the integrity of each message.
Expired or Invalid Sessions	Implement session management with unique tokens and validation of expiration times.

Flow Diagrams

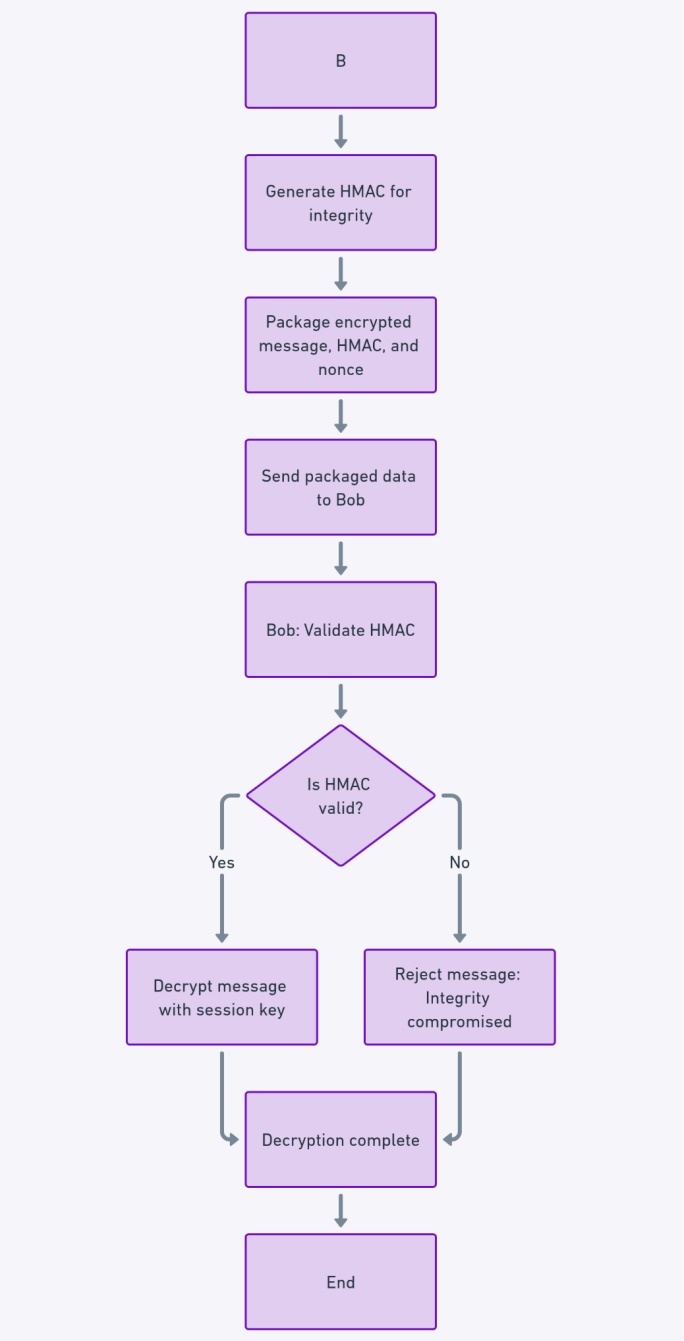
Key Flow

1. Alice generates her private and public key (ECC).
2. Alice and Bob exchange public keys.
3. Each generates a shared session key using ECDH.
4. The session key is stored in memory for the duration of the session.



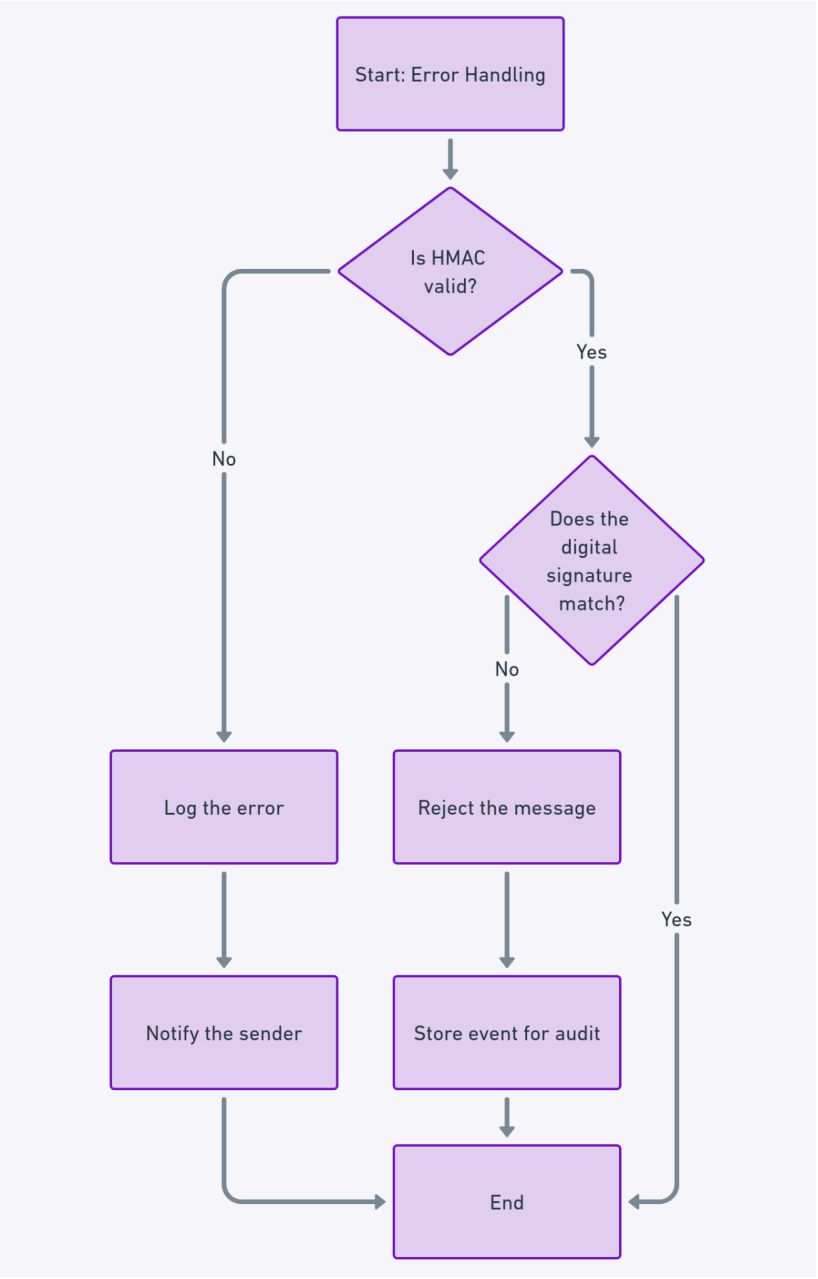
2. Message Flow

1. **Encryption (Alice)**:
    - The message is encrypted with AES-256 GCM.
    - An HMAC is generated to ensure integrity.
    - The encrypted message and HMAC are packaged together with the nonce.
2. **Decryption (Bob)**:
    - Bob validates the integrity of the message with the HMAC.
    - If valid, he decrypts the message using the session key.



3. Error Handling

1. If the HMAC is invalid:
    - Log the error.
    - Notify the sender.
2. If the digital signature does not match:
    - Reject the message.
    - Store the event for audit.



Implementation Considerations

Technical Requirements

1. **Cryptographic Libraries**:
    - Use of `cryptography` (Python) for all cryptographic operations.
    - Consider **OpenSSL** or **Libsodium** for higher performance environments or different languages.
2. **Secure Key Storage**:
    - Implement **Hardware Security Modules (HSM)** to store private keys.
    - Alternatively, use operating system encrypted storage, such as **TPM (Trusted Platform Module)** on compatible hardware.
3. **Secure Transport**:
    - Messages should be transmitted exclusively over HTTPS/TLS 1.2 or higher.
    - Strict validation of SSL/TLS certificates.
4. **Session Management**:
    - Associate session keys with unique tokens and limit the duration of each session (e.g., 24 hours or 100 messages).
    - Use UUIDs to generate tokens and ensure that they are impossible to predict.

Deployment Procedures

1. **Prepare Production Environment**:
    - Set up a secure server with TLS/HTTPS enabled.
    - Ensure certificates are signed by a trusted CA.
2. **Operational Security**:
    - Limit administrative access to the server and system configuration files.
    - Implement multi-factor authentication for administrative roles.
3. **Pre-Deployment Testing**:
    - Run load tests to simulate multiple concurrent sessions (e.g., 1000+ sessions).
    - Validate all security features (integrity, authenticity, error handling).
4. **Monitoring and Auditing**:
    - Implement a log system to record security events, such as:
    - Token reuse attempts.
    - Message integrity errors.
    - Certificate validation failures.

Advanced Functions

1. Digital Signatures

- Alice signs the encrypted message with her private key.
- Bob validates the signature using Alice's public key.

2. Prevention of Replay Attacks

- Each message includes a **unique nonce** and a **timestamp**.

- Bob verifies that:
    - The nonce has not been used previously.
    - The timestamp is within an acceptable range.