Lab 5 : Secure communication

- **Author** : KATCHALA MELE Abdoulaye, SATARZAI Mirwis
- **Group** : Cyber G2

Authenticated Key Exchange Protocol Documentation

this part outlines a secure, mutually authenticated key exchange protocol that enables two parties, Client and Server, to establish a shared secret key. The protocol leverages asymmetric cryptography (RSA) for mutual authentication and Diffie-Hellman (DH) for secure shared secret generation. Both parties use certificates to prove their identities and ensure the authenticity of the key exchange process.

Message Sequence and Cryptographic Algorithms

Initialization:

- Client and Server each generate an RSA key pair:
- Both parties generate self-signed X.509 certificates to encapsulate their public keys for identity verification.
- **Algorithms**:
    - RSA (2048-bit) for key pair generation.
    - OpenSSL for certificate self signing.

Step 1: Client initiates key exchange

- Client sends their self-signed certificate including their public RSA key and a random nonce as challenge.
- Server can use this certificate to verify Client's identity in later steps.
- **Algorithms**:
    - Secure random number generator to generates nonces.
    - JSON for message serialization and deserialization.

Step 2: Server verifies and responds (Cert, Sign, Nonce)

- Server verifies the signature to confirm the integrity and authenticity of Client's certificate.
- Responds with:
    - Server's self-signed certificate including their public RSA key.
    - A digital signature of the nonce generated by Client proving ownership of their private key.
    - A new random nonce as a challenge to Client.
- **Algorithms**:
    - RSA-SHA256 for signing and verification.
    - Secure random number generator to generates nonces.
    - JSON for message serialization and deserialization.

Step 3: Client verifies, sign and send DH public key

- Client verifies the signature and authenticity of Server's certificate.
- Responds with:
    - A digital signature of the nonce generated by Server proving ownership of their private key.
    - their DH public key to initiate the key exchange.
    - Signature of the DH public key.
- **Algorithms**:
    - RSA-SHA256 for signing and verification.
    - Generate a random 2048-bit DH private key. (Using RFC 3526 Group of 2048-bit)
    - JSON for message serialization and deserialization.

Step 4: Server finalizes key exchange

- Server verifies the signature of their previous nonce to authenticate Client.
- Responds with :
    - their DH public key and its signature of the DH public key.
    - Session ID and its signature to be used in the symmetric communication channel.
- **Algorithms**:
    - RSA-SHA256 for signing and verification.
    - Generate a random 2048-bit DH private key. (Using RFC 3526 Group of 2048-bit)
    - JSON for message serialization and deserialization.

Step 5: Everyone compute secret key.

- Both parties compute the shared secret using the DH key exchange.
- **Algorithms**:
    - Compute the shared secret using the DH key exchange. (Using RFC 3526 Group of 2048-bit)
    - JSON for message serialization and deserialization.

Fallback Mechanisms and Error Handling

The protocol includes fallback mechanisms and error handling to ensure robustness and security:

- If a certificate signature fails verification, the protocol is aborted.
- If a signed nonce fails verification, the protocol is terminated.
- If the DH public key signature fails verification, the protocol is aborted.
- If the session ID signature fails verification, the protocol is aborted.
- If the shared secret derived from the DH key exchange does not match, the protocol is aborted.

This protocol ensures robust mutual authentication and secure shared secret generation by combining RSA for identity verification and Diffie-Hellman for key exchange. It mitigates replay, man-in-the-middle, and impersonation attacks, making it suitable for establishing a secure communication channel.

Communication Channel Design Documentation

This part specifies the design of a symmetric communication channel to securely transmit messages after a shared secret is established. The design incorporates encryption for confidentiality, integrity checks for authenticity, and measures to protect against common attacks such as replay and man-in-the-middle (MITM) attacks.

Session Details

The shared secret derived from the DH key exchange is used to encrypt and authenticate messages between Client and Server. The session includes the following security features:

- To encrypt messages, we use AES-GCM with a 256-bit key derived from the shared secret.
- GMAC and tag generated by AES-GCM ensures integrity and authenticity.
- Use of a unique session ID combined with a sequence number in each encrypted message. `Message = AES-GCM(SessionID || SeqNum || Payload).`
- One party send a termination message (`FIN {sid}`) encrypted with shared key at the end of the session to ensure a clean closure. (Using CTRL+D in the app for example)

Algorithms used

- **Nonce generator** to create unique nonces corresponding to the IVs used in AES-GCM for each message.
- **AES-GCM** with 256-bit keys provides strong encryption that is computationally infeasible to break. GCM mode generates an authentication tag that ensures data integrity and authenticity in a single operation and is highly efficient and widely supported in modern hardware.
- **Session ID**: Ensures messages belong to a specific communication session.
- **Sequence Numbers**: Guarantee the order of messages and prevent duplicates.
- **JSON** for message serialization and deserialization to ensure compatibility and ease of use.

Attack Mitigation Measures

1. **Replay Attacks**: Sequence numbers and session IDs prevent replay attacks by ensuring messages are unique and in the correct order.
2. **Man-in-the-Middle (MITM) Attacks**: Mutual authentication and integrity check of messages prevent MITM attacks by ensuring the identity of the communicating parties and the integrity of the messages.

The symmetric communication channel design leverages AES-GCM for efficient and secure encryption and integrity. Replay attacks are mitigated through session IDs and sequence numbers, while session termination is secured using digitally signed messages. The overall design ensures strong protection against known attacks and provides a robust mechanism for maintaining secure communications between authenticated parties.

Tools and Libraries

- **Random**: Used for generating random nonces, session IDs, and large numbers.
- **Sympy**: Used for generating large prime numbers for RSA key generation.
- **OpenSSL**: Used for certificate self-signing, and RSA-SHA256 signing and verification.
- **PyCryptodome**: Python library for AES-GCM encryption and decryption and RSA key generation from existing keys.
- **Hashlib**: Used for SHA256 hashing operations.
- **Base64**: Used for encoding and decoding messages.
- **JSON**: Used for message serialization and deserialization.
- **Github copilot**: Used for generating code snippets and suggestions.
- **ChatGPT**: Used for generating text and documentation.