Problem Statement:

Cryptography Simulation with mbedTLS/OpenSSL Library Usage and User Interaction.

Team Member:

Swapna K T

Contents:

Digital Certificates Generation:

Creating Certificates and RSA Key Pairs

1.Create Root Certificat: Self-signed root certificate (rootCA.crt) with RSA key size of 3072, SHA384, serial number 01.

Sh

```
openssl req -x509 -sha384 -newkey rsa:3072 -keyout rootCA.key -out rootCA.crt -set_serial 1
```

2. Generate RSA Key Pair for Alice: RSA key size of 3072, SHA384, signed with root CA, serial number 02.

Generate Alice's Private Key:

sh

```
openssl genpkey -algorithm RSA -out alice.key -pkeyopt rsa_keygen_bits:3072
```

Create Certificate Signing Request (CSR) for Alice:

sh

```
openssl req -new -key alice.key -out alice.csr -sha384 -subj "/CN=Alice.com"
```

Sign Alice's CSR with Root CA:

sh

```
openssl x509 -req -in alice.csr -CA rootCA.crt -CAkey rootCA.key -CAcreateserial -out alice.crt -days 365 -sha384 -set_serial 02
```

3. Generate RSA Key Pair for Bob: RSA key size of 3072, SHA384, signed with root CA, serial number 03.

Generate Bob's Private Key:

sh

openssl genpkey -algorithm RSA -out bob.key -pkeyopt rsa_keygen_bits:3072

Create Certificate Signing Request (CSR) for Bob:

sh

openssl req -new -key bob.key -out bob.csr -sha384 -subj "/CN=Bob.com"

Sign Bob's CSR with Root CA:

sh

openssl x509 -req -in bob.csr -CA rootCA.crt -CAkey rootCA.key -CAcreateserial -out bob.crt -days 365 -sha384 -set_serial 03

Crypto-wrapper Implementation Summary

HMAC-SHA256

- Function: CryptoWrapper::hmac_SHA256
- APIs:
  - EVP_MD_CTX_new: Create a message digest context.
  - EVP_PKEY_new_raw_private_key: Create a raw private key.
  - EVP_DigestSignInit: Initialize digest sign context.
  - EVP_DigestSignUpdate: Feed data to the context.
  - EVP_DigestSignFinal: Retrieve the signature.
  - EVP_MD_CTX_free: Free message digest context.
  - EVP_PKEY_free: Free the private key structure.
- Purpose: Ensure data integrity and authenticity.

HKDF-SHA256

- Function: CryptoWrapper::deriveKey_HKDF_SHA256
- APIs:
  - EVP_PKEY_CTX_new_id: Create key derivation context.
  - EVP_PKEY_derive_init: Initialize key derivation context.
  - EVP_PKEY_CTX_set_hkdf_md: Set message digest type.
  - EVP_PKEY_CTX_set1_hkdf_salt: Set salt value.
  - EVP_PKEY_CTX_set1_hkdf_key: Set initial keying material.
  - EVP_PKEY_CTX_add1_hkdf_info: Add application-specific information.
  - EVP_PKEY_derive: Derive the key.
  - EVP_PKEY_CTX_free: Free the key derivation context.
- Purpose: Derive strong cryptographic keys.

AES-GCM-256

-Function:CryptoWrapper::encryptAES_GCM256,
`CryptoWrapper::decryptAES_GCM256`

- **APIs**:
  - EVP_CIPHER_CTX_new: Create a cipher context.
  - EVP_EncryptInit_ex: Initialize encryption operation.
  - EVP_CIPHER_CTX_ctrl: Control cipher context parameters.
  - EVP_EncryptUpdate: Encrypt data.
  - EVP_EncryptFinal_ex: Finalize encryption.
  - EVP_CIPHER_CTX_free: Free cipher context.
  - EVP_DecryptInit_ex: Initialize decryption operation.
  - EVP_DecryptUpdate: Decrypt data.
  - EVP_DecryptFinal_ex: Finalize decryption.
- Purpose: Encrypt and decrypt data, ensuring confidentiality and integrity.

RSA-PSS

Function:CryptoWrapper::signMessageRsa3072Pss,
CryptoWrapper::verifyMessageRsa3072Pss

- APIs:
  - EVP_MD_CTX_create: Create message digest context.
  - EVP_get_digestbyname: Retrieve digest by name.
  - EVP_DigestSignInit: Initialize digest sign context.
  - EVP_DigestSignUpdate: Feed data to context.
  - EVP_DigestSignFinal: Retrieve the signature.
  - EVP_DigestVerifyInit: Initialize digest verify context.
  - EVP_DigestVerifyUpdate: Feed data to verify context.
  - EVP_DigestVerifyFinal: Check the signature.
  - EVP_MD_CTX_destroy: Destroy the context.
- Purpose: Sign and verify messages for secure authentication.

Diffie-Hellman

- Function: CryptoWrapper::startDh
- APIs:
  - `BN_get_rfc3526_prime_3072`: Retrieve a 3072-bit prime number.

  - `BN_bin2bn`: Convert binary data to BIGNUM.
  - `OSSL_PARAM_BLD_new`: Create a parameter builder.
  - `OSSL_PARAM_BLD_push_BN`: Add BIGNUM to builder.
  - `OSSL_PARAM_BLD_to_param`: Convert builder to parameters.
  - `EVP_PKEY_CTX_new_from_name`: Create key context from name.
  - `EVP_PKEY_fromdata_init`: Initialize key from data context.
  - `EVP_PKEY_fromdata`: Create key from data.
  - `EVP_PKEY_CTX_new_from_pkey`: Create key context from existing key.
- **Purpose**: Generate public/private key pairs for secure key exchange.

RSA Key Management
- **Function**: `CryptoWrapper::readRSAKeyFromFile`, `CryptoWrapper::writePublicKeyToPemBuffer`, `CryptoWrapper::loadPublicKeyFromPemBuffer`
- **APIs**:
  - `BIO_new_file`: Create a file BIO.
  - `PEM_read_bio_PrivateKey_ex`: Read private key from BIO.
  - `EVP_PKEY_CTX_new`: Create key context.
  - `EVP_PKEY_free`: Free private key structure.
  - `BIO_free`: Free the BIO.
  - `EVP_PKEY_CTX_get0_pkey`: Retrieve private key from context.
  - `EVP_PKEY_get_bn_param`: Retrieve BIGNUM parameter.
  - `BN_bn2bin`: Convert BIGNUM to binary.
- **Purpose**: Manage RSA keys for cryptographic operations.

Context Management
- **Function**: `CryptoWrapper::cleanKeyContext`
- **APIs**:
  - `EVP_PKEY_CTX_free`: Free key context.
- **Purpose**: Prevent memory leaks by freeing cryptographic contexts.Usage Summary

- **HMAC-SHA256**: Creates message authentication codes to verify data integrity and authenticity.
- **HKDF-SHA256**: Derives secure keys from input keying material, salt, and context.

- **AES-GCM-256**: Encrypts and decrypts data with authenticated encryption, ensuring confidentiality and integrity.
- **RSA-PSS**: Signs messages and verifies signatures to authenticate source and integrity.
- **Diffie-Hellman**: Securely exchanges cryptographic keys over a public channel.
- **RSA Key Management**: Handles RSA keys, including reading, writing, and loading keys.
- **Context Management**: Manages cryptographic contexts and ensures proper resource deallocation.

Protocol Flow Understanding

- Employ hybrid cryptography: symmetric and asymmetric.
- Use asymmetric cryptography to prevent man-in-the-middleattacks.
- Authenticate remote party using the SIGMA protocol.
- Switch to symmetric cryptography for message exchange.
- Execute the SIGMA protocol for each new session.

SIGMA Protocol Steps:

1. **Alice Initiates (SIGMA#1)**:
    - Alice generates and sends her public key to Bob.
2. **Bob Responds (SIGMA#2)**:
    - Bob receives Alice's public key.
    - Bob sends back:
        - His public key.
        - His certificate (to prove his identity).
        - A signature (to ensure the message integrity and authenticity).
        - A MAC (to ensure the message has not been tampered with).
3. **Alice Confirms (SIGMA#3)**:
    - Alice receives Bob's public key and certificate.
    - Alice sends back:
        - Her public key again (to maintain the session context).
        - Her certificate (to prove her identity).
        - A signature (to ensure the message integrity and authenticity).
        - A MAC (to ensure the message has not been tampered with).

SIGMA Protocol Implementation:

Prepare SIGMA Message:

- Read the local certificate and private key.
- Concatenate local and remote DH buffers.
- Sign concatenated buffer.
- Derive MAC key from a shared secret.
- Prepare HMAC and pack SIGMA message.

Verify SIGMA Message:

- Unpack SIGMA message.
- Verify certificate and public key.
- Verify signature over concatenated buffer.
- Derive the MAC key and prepare HMAC.
- Compare HMACs.

Encryption and Decryption Mechanism

- Derive session key from a shared secret.
- Use CryptoWrapper for encryption and decryption with AAD as themessage type.

Key Learnings

- HMAC-SHA256 & HKDF Key
- AES-GCM-256 Encryption/Decryption
- Diffie-Hellman Key Exchange
- Error Handling
- Memory Management
- Constants and Buffer Sizes
- Library Integration
- Digital Signature (RSA)
- Certificate Verification
- SIGMA Protocol
- Client-Server Model Simulation
- Protection against Man-in-the-Middle Attacks