Intel Unnati Training

Summer Internship – 2024

Contents:

Team Name: The Achievers

Team Members:

1. K. Koushik
2. J. Dheeraj Lakshman
3. B. Thanuja

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

Digital Certificates Generation:

1. Creating a self-signed root certificate (rootCA.crt) with an RSA key size of 3072 with SHA384 and setting the serial number 01.

- openssl req -x509 -sha384 -newkey rsa:3072 -keyout rootCA.key -out rootCA.crt -set_serial 01

2. Generating RSA keypair of size 3072 with SHA384 for "Alice" and signing with root CA and setting serial number 02.

2. a. Generating Alice's Private Key.

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

2. b. Creating a Certificate Signing Request (CSR) for Alice with the common name "Alice.com".

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

2. c. Signing Alice's CSR with the root CA for producing Alice's certificate.

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

3. Generating RSA keypair of size 3072 with SHA384 for "Bob" and signing with root CA and setting serial number 03.

3.a. Generating Bob's Private Key.

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

3. b. Creating a Certificate Signing Request (CSR) for Bob with a the common name "Bob.com".

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

3. c. Signing Bob's CSR with the root CA for producing Bob's certificate.

- 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:

HMAC-SHA256:

Function: CryptoWrapper::hmac_SHA256 APIs:

EVP_MD_CTX_new: Creates a new message digest context.

EVP_PKEY_new_raw_private_key: Creates a new raw private key.

EVP_DigestSignInit: Initializes a digest sign context.

EVP_DigestSignUpdate: Feeds data to the digest sign context.

EVP_DigestSignFinal: Finalizes the digest sign and retrieves the signature.

EVP_MD_CTX_free: Frees the message digest context.

EVP_PKEY_free: Frees the private key structure.

Purpose: Create an HMAC using the SHA-256 hashing algorithm to ensure data integrity and authenticity.

HKDF-SHA256:

Function: CryptoWrapper::deriveKey_HKDF_SHA256 APIs:

EVP_PKEY_CTX_new_id: Creates a new key derivation context for HKDF.

EVP_PKEY_derive_init: Initializes a key derivation context.

EVP_PKEY_CTX_set_hkdf_md: Sets the message digest type for HKDF.

EVP_PKEY_CTX_set1_hkdf_salt: Sets the salt value for HKDF.

EVP_PKEY_CTX_set1_hkdf_key: Sets the initial keying material for HKDF.

EVP_PKEY_CTX_add1_hkdf_info: Adds application-specific information to the HKDF context.

EVP_PKEY_derive: Derives a key using the initialized HKDF context.
EVP_PKEY_CTX_free: Frees the key derivation context.

Purpose: Derive strong cryptographic keys from initial keying material, salt, and context information using SHA-256.

AES-GCM-256:

Function: CryptoWrapper::encryptAES_GCM256, CryptoWrapper::decryptAES_GCM256 APIs:

EVP_CIPHER_CTX_new: Creates a new cipher context.

EVP_EncryptInit_ex: Initializes the encryption operation.

EVP_CIPHER_CTX_ctrl: Controls cipher context parameters (e.g., setting the GCM IV).

EVP_EncryptUpdate: Encrypts data.

EVP_EncryptFinal_ex: Finalizes the encryption operation.

EVP_CIPHER_CTX_free: Frees the cipher context.

EVP_DecryptInit_ex: Initializes the decryption operation.

EVP_DecryptUpdate: Decrypts data.

EVP_DecryptFinal_ex: Finalizes the decryption operation.

Purpose: Encrypt and decrypt data using AES with 256-bit keys in GCM mode for confidentiality and integrity.

RSA-PSS:

Function: CryptoWrapper::signMessageRsa3072Pss, CryptoWrapper::verifyMessageRsa3072Pss APIs:

EVP_MD_CTX_create: Creates a new message digest context.

EVP_get_digestbyname: Retrieves a digest by name.

EVP_DigestSignInit: Initializes a digest sign context for RSA-PSS.

EVP_DigestSignUpdate: Feeds data to the digest sign context.

EVP_DigestSignFinal: Finalizes the digest sign and retrieves the signature.

EVP_DigestVerifyInit: Initializes a digest verify context for RSA-PSS.

EVP_DigestVerifyUpdate: Feeds data to the digest verify context.

EVP_DigestVerifyFinal: Finalizes the digest verify and checks the signature.

EVP_MD_CTX_destroy: Destroys the message digest context.

Purpose: Sign messages and verify signatures using RSA-3072 with PSS padding for secure authentication.

Diffie-Hellman:

Function: CryptoWrapper::startDh APIs:

BN_get_rfc3526_prime_3072: Retrieves a 3072-bit RFC 3526 prime number.

BN_bin2bn: Converts binary data to a BIGNUM.

OSSL_PARAM_BLD_new: Creates a new parameter builder.

OSSL_PARAM_BLD_push_BN: Adds a BIGNUM parameter to the builder.

OSSL_PARAM_BLD_to_param: Converts the parameter builder to parameters.

EVP_PKEY_CTX_new_from_name: Creates a new key context from a name.

EVP_PKEY_fromdata_init: Initializes a key from data context.

EVP_PKEY_fromdata: Creates a key from data.

EVP_PKEY_CTX_new_from_pkey: Creates a new key context from an existing key.

Purpose: Generate public/private key pairs for secure key exchange using Diffie-Hellman algorithm.

RSA Key Management:

Function: CryptoWrapper::readRSAKeyFromFile, CryptoWrapper::writePublicKeyToPemBuffer, CryptoWrapper::loadPublicKeyFromPemBuffer APIs:

BIO_new_file: Creates a new file BIO.
PEM_read_bio_PrivateKey_ex: Reads a private key from a BIO in PEM format.

EVP_PKEY_CTX_new: Creates a new key context.

EVP_PKEY_free: Frees the private key structure.

BIO_free: Frees the BIO.

EVP_PKEY_CTX_get0_pkey: Retrieves the private key from the context.

EVP_PKEY_get_bn_param: Retrieves a BIGNUM parameter from the key.

BN_bn2bin: Converts a BIGNUM to binary data.

Purpose: Read RSA keys from files, convert keys to PEM format, and load keys from PEM buffers for cryptographic operations.

Context Management:

Function: CryptoWrapper::cleanKeyContext APIs:

EVP_PKEY_CTX_free: Frees the key context.

Purpose: Free the memory associated with cryptographic contexts to prevent memory leaks.

Usage Summary:

HMAC-SHA256: For creating message authentication codes to verify data integrity and authenticity.

HKDF-SHA256: For deriving secure keys from a combination of input keying material, salt, and context.

AES-GCM-256: For encrypting and decrypting data, ensuring confidentiality and data integrity with authenticated encryption.

RSA-PSS: For signing messages and verifying signatures to authenticate the source and integrity of messages.

Diffie-Hellman: For securely exchanging cryptographic keys over a public channel.

RSA Key Management: For handling RSA keys, including reading from files, writing to buffers, and loading from buffers.

Context Management: For managing cryptographic contexts and ensuring proper resource deallocation.

Protocol Flow Understanding

- Employ hybrid cryptography: symmetric and asymmetric.
- Use asymmetric cryptography to prevent man-in-the-middle attacks.
- 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:

- "Hello" (SIGMA#1): Send Alice's public key.
- "Hello Back" (SIGMA#2): Send Bob's public key, certificate, signature, and MAC.
- "Hello Done" (SIGMA#3): Send Alice's public key, certificate, signature, and MAC.

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.

Initialization and Session Handling:

- Start Diffie-Hellman for a client session.
- Read and handle payloads for server sessions.

Encryption and Decryption Mechanism

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

Session Termination

- Use Utils::securelyCleanMemory for releasing private key password.
- Clean DH context using CryptoWrapper's cleanDhContext.

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