Laboratory_06: OpenSSL for AES, RSA and ECC

This laboratory covers OpenSSL tool applicability.

Installation

- sudo apt update
- sudo apt-get install openssl -y
- openssl version

OpenSSL for symmetric encryption

1. Create a file to encrypt:
    a. echo "lab for fun, hands-on learning." > secret.txt
2. Review the created file:
    a. cat secret.txt

OpenSSL for RSA

3. Generate a key pair with:
    a. openssl genrsa -out private.pem 1024
4. Review key pair created:
    a. cat private.pem
5. View the RSA key pair:
    a. openssl rsa -in private.pem -text -noout
6. Secure the encrypted key with 3-DES:
    a. openssl rsa -in private.pem -des3 -out key3des.pem
7. Export the public key:
    a. openssl rsa -in private.pem -out public.pem -outform PEM -pubout

8. Create a file named “secret.txt” and put a message into it. Next encrypt it with your public key:
    a. openssl pkeyutl -encrypt -pubin -inkey public.pem -in secret.txt -out file.bin
9. n decrypt with the private key:
    a. openssl pkeyutl -decrypt -inkey private.pem -in file.bin -out decrypted.txt

OpenSSL for Elliptic Curve Cryptography (ECC)

10. Generate a private key:
    a. openssl ecparam -name secp256k1 -genkey -out ecc_private.pem
11. View the details of the ECC parameters used with:
    a. openssl ecparam -in ecc_private.pem -text -param_enc explicit -noout
12. Generate your public key based on your private key with:
    a. openssl ec -in ecc_private.pem -text –noout
13. Extract the public key from your private key:

    ```
    openssl ec -in ecc_private.pem -pubout -out ecc_public.pem
    ```

14. Create a message file for signing:

    ```
    echo "This is a message to be signed with ECC" > ecc_message.txt
    ```

15. Sign the message with your ECC private key:

    ```
    openssl dgst -sha256 -sign ecc_private.pem -out ecc_signature.bin ecc_message.txt
    ```

16. Verify the signature using the public key:

    ```
    openssl dgst -sha256 -verify ecc_public.pem -signature
    ```

Comparing RSA and ECC Performance

1. **Key Generation Time Comparison**

 # Measure RSA key generation time (2048-bit)

time openssl genrsa -out rsa_2048.pem 2048

 # Measure RSA key generation time (4096-bit)

time openssl genrsa -out rsa_4096.pem 4096

 # Measure ECC key generation time (secp256k1)

time openssl ecparam -name secp256k1 -genkey -out ecc_256.pem

 # Measure ECC key generation time (secp521r1 - higher security)

time openssl ecparam -name secp521r1 -genkey -out ecc_521.pem

2. **File Size Comparison**

 # Compare key sizes

ls -l rsa_2048.pem rsa_4096.pem ecc_256.pem ecc_521.pem

3. **Signature Creation and Verification**

 # Create a sample file

echo "This is a test message for digital signature comparison" > message.txt

 # RSA Signing

openssl dgst -sha256 -sign rsa_2048.pem -out rsa_signature.bin message.txt

time openssl dgst -sha256 -sign rsa_2048.pem -out rsa_signature.bin message.txt

 # RSA Verification

openssl rsa -in rsa_2048.pem -pubout -out rsa_pub.pem

time openssl dgst -sha256 -verify rsa_pub.pem -signature rsa_signature.bin message.txt

 # ECC Signing

time openssl dgst -sha256 -sign ecc_256.pem -out ecc_signature.bin message.txt

 # ECC Verification

openssl ec -in ecc_256.pem -pubout -out ecc_pub.pem

time openssl dgst -sha256 -verify ecc_pub.pem -signature ecc_signature.bin message.txt

 # Compare signature sizes

ls -l rsa_signature.bin ecc_signature.bin

4. Encryption/Decryption Speed Comparison

 # Create a test file of 1MB for encryption tests

dd if=/dev/urandom of=testfile.bin bs=1M count=1

 # RSA encryption/decryption is limited by key size, so we'll encrypt a small chunk

head -c 100 testfile.bin > small_chunk.bin

 # RSA encryption (public key)

time openssl pkeyutl -encrypt -pubin -inkey rsa_pub.pem -in small_chunk.bin -out rsa_encrypted.bin

 # RSA decryption (private key)

time openssl pkeyutl -decrypt -inkey rsa_2048.pem -in rsa_encrypted.bin -out rsa_decrypted.bin

 # For ECC, use ECDH to generate a shared secret and then use symmetric encryption

 # Generate ephemeral ECDH keypair

openssl ecparam -name secp256k1 -genkey -out ecdh_temp.pem

openssl ec -in ecdh_temp.pem -pubout -out ecdh_temp_pub.pem

 # Derive shared secret (normally the other party would do this with their private key)

time openssl pkeyutl -derive -inkey ecc_256.pem -peerkey ecdh_temp_pub.pem -out shared_secret.bin

 # Use the shared secret to encrypt with AES-256-CBC

time openssl enc -aes-256-cbc -salt -in testfile.bin -out ecc_encrypted.bin -pass file:./shared_secret.bin

 # Decrypt

time openssl enc -d -aes-256-cbc -in ecc_encrypted.bin -out ecc_decrypted.bin -pass file:./shared_secret.bin