Applied Cryptography
Week 5/ Midterm

Student Name: **Mikheil Davidovi**
Class: **Applied Cryptography**

Submission Date: **November 8, 2025**

Tools and Materials:
Command Prompt
OpenSSL
Windows PowerShell

Task 1 – AES Encryption

Methods:

Task 1A: Encrypt a file using AES-128-CBC

1) Create the plaintext file
Command:

```
echo "This file contains top secret information." > secret.txt
```

2) Encrypt a file
Command:

```
openssl enc -aes-128-cbc -salt -pbkdf2 -in secret.txt -out secret.enc -pass pass:'Intelligence'
```

Command:

```
openssl enc -aes-128-cbc -salt -pbkdf2 -in secret.txt -out secret.enc
```

Task 1B: Decrypt secret.enc

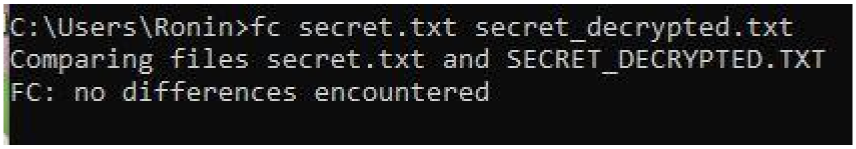
3) Decrypt the file
Command:

```
openssl enc -d -aes-128-cbc -salt -pbkdf2 -in secret.enc -out secret_decrypted.txt -pass pass:'Intelligence'
```

4) Verify the file
Command:

```
fc secret.txt secret_decrypted.txt
```

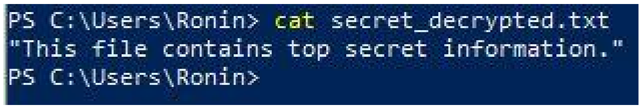
Output:



```
C:\Users\Ronin>fc secret.txt secret_decrypted.txt
Comparing files secret.txt and SECRET_DECRYPTED.TXT
FC: no differences encountered
```

Command:
cat secret_decrypted.txt

Output:



```
PS C:\Users\Ronin> cat secret_decrypted.txt
"This file contains top secret information."
PS C:\Users\Ronin>
```

Artifacts:

- secret.txt
- secret.enc
- secret_decrypted.txt

Task 2 – ECC Signature Verification

Methods:

Task 2A: Generate ECC keys

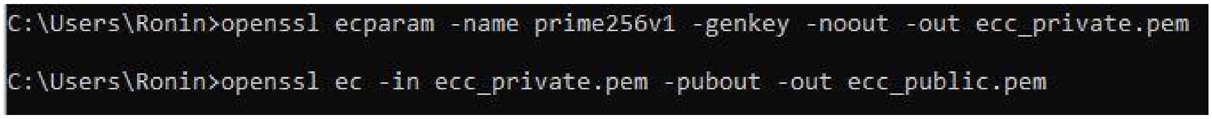
1) Generate the ECC private key
Command:
openssl ecparam -name prime256v1 -genkey -noout -out ecc_private.pem

2) Extract the public key

Command:

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

Output:



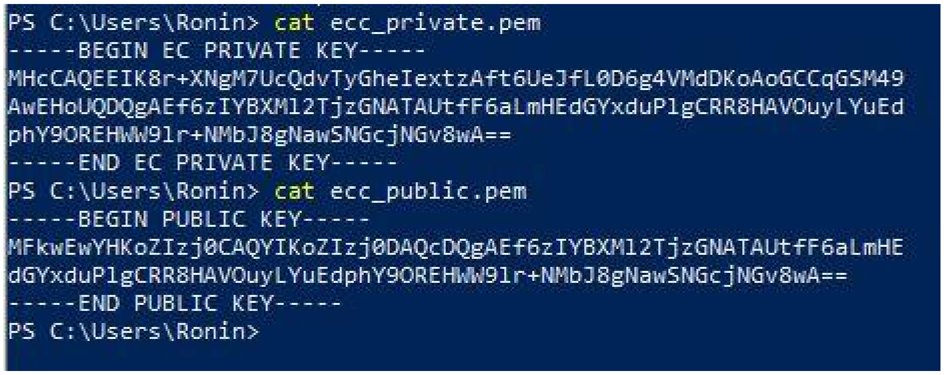
```
C:\Users\Ronin>openssl ecparam -name prime256v1 -genkey -noout -out ecc_private.pem

C:\Users\Ronin>openssl ec -in ecc_private.pem -pubout -out ecc_public.pem
```

3) Verify key files

Commands:

cat ecc_private.pem
cat ecc_public.pem

Outputs:



```
PS C:\Users\Ronin> cat ecc_private.pem
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIK8r+XNgM7UcQdvTyGheIextzAft6UeJfL0D6g4VMdDKoAoGCCqGSM49
AwEHoUQDQgAEf6zIYBXMl2TjzGNATAUtfF6aLmHEdGYxduPlgCRR8HAVOuyLYuEd
phY9OREHWW9lr+NMbJ8gNawSNGcjNGv8wA==
-----END EC PRIVATE KEY-----
PS C:\Users\Ronin> cat ecc_public.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEf6zIYBXMl2TjzGNATAUtfF6aLmHE
dGYxduPlgCRR8HAVOuyLYuEdphY9OREHWW9lr+NMbJ8gNawSNGcjNGv8wA==
-----END PUBLIC KEY-----
PS C:\Users\Ronin>
```

Task 2B: Sign and verify a message

4) Create a file

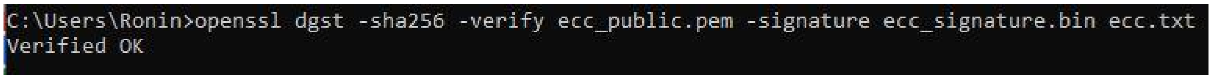
Command:

echo "Elliptic Curves are efficient." > ecc.txt

5) Sign the message
Command:
openssl dgst -sha256 -sign ecc_private.pem -out ecc_signature.bin ecc.txt

6) Verify the signature
Command:
openssl dgst -sha256 -verify ecc_public.pem -signature ecc_signature.bin ecc.txt

Output:



```
C:\Users\Ronin>openssl dgst -sha256 -verify ecc_public.pem -signature ecc_signature.bin ecc.txt
Verified OK
```

Artifacts:

- ecc_private.pem
- ecc_public.pem
- ecc.txt
- ecc_signature.bin

Task 3 – Hashing & HMAC

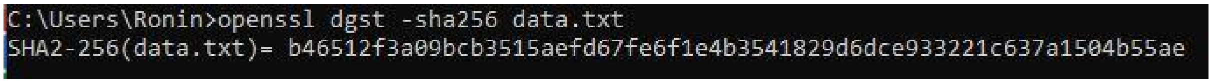
Methods:

Task 3A: SHA-256 Hash

1) Create a file
Command:
echo "Never trust, always verify." > data.txt

2) Compute the hash
Command:
openssl dgst -sha256 data.txt

Output:



```
C:\Users\Ronin>openssl dgst -sha256 data.txt
SHA2-256(data.txt)= b46512f3a09bcb3515aefd67fe6f1e4b3541829d6dce933221c637a1504b55ae
```

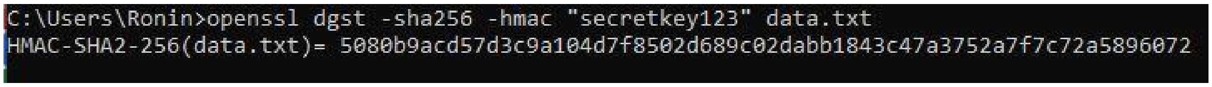
Task 3B: HMAC using SHA-256

3) Use the key

Command:

openssl dgst -sha256 -hmac "secretkey123" data.txt

Output:



```
C:\Users\Ronin>openssl dgst -sha256 -hmac "secretkey123" data.txt
HMAC-SHA2-256(data.txt)= 5080b9acd57d3c9a104d7f8502d689c02dabb1843c47a3752a7f7c72a5896072
```

Task 3C: Integrity Check

4) Modify the file

Action:

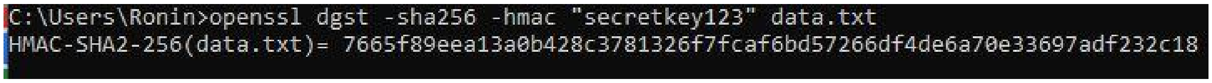
Never trust, always verify. **changed to** Never trust, always verify!

5) Recompute the HMAC

Command:

openssl dgst -sha256 -hmac "secretkey123" data.txt

Output:



```
C:\Users\Ronin>openssl dgst -sha256 -hmac "secretkey123" data.txt
HMAC-SHA2-256(data.txt)= 7665f89eea13a0b428c3781326f7fcaf6bd57266df4de6a70e33697adf232c18
```

6) Explanation

Changing even one character in data.txt produces a completely different HMAC value. This demonstrates integrity protection: HMAC detects any unauthorized modification. Because it uses both a secret key and the message content, attackers cannot forge a valid HMAC without knowing the key.

Artifacts:
data.txt

Task 4 – Diffie-Hellman Key Exchange

Methods:

Task 4A: Simulate DH Key Exchange

1) Generate DH parameters

Command:

```
openssl genpkey -genparam -algorithm DH -out dhparam.pem -pkeyopt dh_paramgen_prime_len:2048
```

2) Generate Alice's private and public key

Commands:

```
openssl genpkey -paramfile dhparam.pem -out alice_private.pem
openssl pkey -in alice_private.pem -pubout -out alice_public.pem
```

3) Generate Bob's private and public key

Commands:

```
openssl genpkey -paramfile dhparam.pem -out bob_private.pem
openssl pkey -in bob_private.pem -pubout -out bob_public.pem
```

4) Derive shared secret keys

Commands:

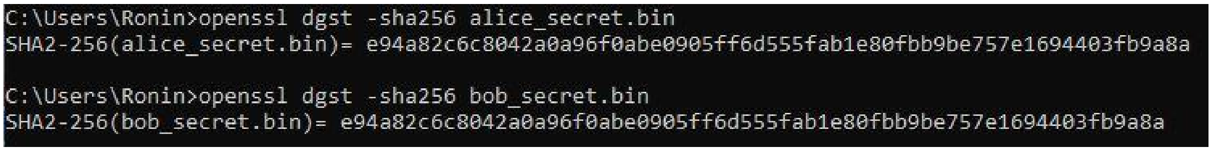
openssl pkeyutl -derive -inkey alice_private.pem -peerkey bob_public.pem -out alice_secret.bin
openssl pkeyutl -derive -inkey bob_private.pem -peerkey alice_public.pem -out bob_secret.bin

5) Compare secrets to verify they match

Commands:

openssl dgst -sha256 alice_secret.bin
openssl dgst -sha256 bob_secret.bin

Outputs:



```
C:\Users\Ronin>openssl dgst -sha256 alice_secret.bin
SHA2-256(alice_secret.bin)= e94a82c6c8042a0a96f0abe0905ff6d555fab1e80fbb9be757e1694403fb9a8a

C:\Users\Ronin>openssl dgst -sha256 bob_secret.bin
SHA2-256(bob_secret.bin)= e94a82c6c8042a0a96f0abe0905ff6d555fab1e80fbb9be757e1694403fb9a8a
```

Task 4B: Real-Life Application

6) Explanation

The Diffie-Hellman (DH) key exchange is used in many secure communication protocols such as TLS (HTTPS), SSH and Signal messaging. It allows two parties (a client and a server) to derive a shared secret key over an insecure channel without directly transmitting the key. This shared key is then used to establish encrypted communication using symmetric ciphers.

DH provides forward secrecy; even if long-term keys are compromised later, past session keys remain secure because they were derived independently for each session. It ensures confidentiality and authenticity in secure protocols used daily on the internet.

Artifacts:

- dhparam.pem

- alice_private.pem
- alice_public.pem
- bob_private.pem
- bob_public.pem
- alice_secret.bin
- bob_secret.bin