Applied Cryptography – Midterm Lab Exam

Task 1 – AES Encryption

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

1. Create a text file: secret.txt with the line:

This file contains top secret information

Step 1: Create the text file

To create secret.txt with the following content: I used this command

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

```
C:\Windows\System32>echo "This file contains top secret information." > secret.txt

C:\Windows\System32>
```

```
C:\Windows\System32>type secret.txt
"This file contains top secret information."

C:\Windows\System32>
```

Step 2: Encrypt using OpenSSL with AES-128-CBC

Used the following command to encrypt it:

openssl enc -aes-128-cbc -salt -pbkdf2 -in secret.txt -out secret.enc -pass pass:mysecretpass

```
C:\Windows\System32>openssl enc -aes-128-cbc -salt -pbkdf2 -in secret.txt -out secret.enc -pass pass:mysecretpass

C:\Windows\System32>
```

```
C:\Windows\System32>type secret.enc
Salted__ì]♥º├◄┘¥-(▒
ⁿ2µ*╠╔ ¡Kvæÿo▒£┼♦;ΣÇOÆdΦ6iUßVs┤├aR1Dü╗┘-τ       Üà
C:\Windows\System32>
```

Task 1B: Decrypt the File

Step 1: Decrypt the file with this command:

openssl enc -d -aes-128-cbc -pbkdf2 -in secret.enc -out decrypted.txt -pass pass:mysecretpass

```
C:\Windows\System32>openssl enc -d -aes-128-cbc -pbkdf2 -in secret.enc -out decrypted.txt -pass pass:mysecretpass

C:\Windows\System32>type decrypted.txt
"This file contains top secret information."

C:\Windows\System32>
```

Step 2: Check that it matches the original

I used this command: *fc secret.txt decrypted.txt*

```
C:\Windows\System32>fc secret.txt decrypted.txt
Comparing files secret.txt and DECRYPTED.TXT
FC: no differences encountered


C:\Windows\System32>
```

Task 2A: Generate ECC Keys

1. Generate private key on prime256v1 curve with this command:

openssl ecparam -name prime256v1 -genkey -noout -out ecc_private_key.pem

```
C:\Windows\System32>openssl ecparam -name prime256v1 -genkey -noout -out ecc_private_key.pem

C:\Windows\System32>
```

To view the private key (with details): *openssl ec -in ecc_private_key.pem -text -noout*

```
C:\Windows\System32>openssl ec -in ecc_private_key.pem -text -noout
read EC key
Private-Key: (256 bit)
priv:
    71:bf:30:33:78:53:3e:e7:fa:09:fa:4a:c7:b0:b3:
    e0:fd:4d:28:a2:df:df:74:4a:a9:d8:21:51:52:e7:
    df:03
pub:
    04:3a:2e:80:12:4f:4e:65:b7:66:b3:bd:3a:e5:25:
    3d:dc:19:e2:5c:c5:ef:d3:1c:95:c9:ec:8e:f0:3c:
    0b:c8:5d:90:96:db:6a:fb:5b:cd:4d:2a:e2:2b:86:
    53:69:32:83:7e:ad:ef:c1:7d:23:8f:6b:cc:44:1b:
    55:ee:af:60:09
ASN1 OID: prime256v1
NIST CURVE: P-256

C:\Windows\System32>
```

Step 2: Extract and save the corresponding public key: I used this command:

openssl ec -in ecc_private_key.pem -pubout -out ecc_public_key.pem

```
C:\Windows\System32>openssl ec -in ecc_private_key.pem -pubout -out ecc_public_key.pem
read EC key
writing EC key

C:\Windows\System32>
```

This is a command to view the public key:

openssl ec -in ecc_public_key.pem -pubin -text -noout

```
C:\Windows\System32>openssl ec -in ecc_public_key.pem -pubin -text -noout
read EC key
Public-Key: (256 bit)
pub:
    04:3a:2e:80:12:4f:4e:65:b7:66:b3:bd:3a:e5:25:
    3d:dc:19:e2:5c:c5:ef:d3:1c:95:c9:ec:8e:f0:3c:
    0b:c8:5d:90:96:db:6a:fb:5b:cd:4d:2a:e2:2b:86:
    53:69:32:83:7e:ad:ef:c1:7d:23:8f:6b:cc:44:1b:
    55:ee:af:60:09
ASN1 OID: prime256v1
NIST CURVE: P-256

C:\Windows\System32>
```

Task 2B: Sign and Verify Message

Step 1: Create the message file ecc.txt I used this command:

echo Elliptic Curves are efficient. > ecc.txt

```
C:\Windows\System32>echo Elliptic Curves are efficient. > ecc.txt

C:\Windows\System32>type ecc.txt
Elliptic Curves are efficient.

C:\Windows\System32>
```

Step 2: Sign the message using your private key

I used this command:

openssl dgst -sha256 -sign ecc_private_key.pem -out ecc_signature.der ecc.txt

```
PS C:\Windows\system32> openssl dgst -sha256 -sign ecc_private_key.pem -out ecc_signature.der ecc.txt
PS C:\Windows\system32>
```

To view signature (as hex) I used this command:

Format-Hex ecc_signature.der

```
PS C:\Windows\system32> Format-Hex ecc_signature.der


           Path: C:\Windows\system32\ecc_signature.der

           00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   30 44 02 20 6F 6A 14 8E 37 09 1D E1 E0 A3 43 9C  0D. oj.7..áà£C
00000010   63 98 23 4B 26 FB D1 1D 6D C6 AE B3 47 44 5C EE  c#K&ûÑ.mÆ®³GD\î
00000020   3F 07 F9 52 02 20 5E D3 15 79 25 53 AF 64 56 20  ?.ùR. ^Ó.y%S¯dV
00000030   75 FC 75 A1 E3 F5 01 92 8D 58 B1 77 E0 C4 68 06  uüu¡ãõ.X±wàÄh.
00000040   1A AD EE CA F4 C2                                 .-îÊôÂ


PS C:\Windows\system32>
```

Step 3: Verify the signature using your public key

```
C:\Windows\System32>openssl dgst -sha256 -verify ecc_public_key.pem -signature ecc_signature.der ecc.txt
Verified OK

C:\Windows\System32>
```

Task 3A: SHA-256 Hash

Step 1: Create the file data.txt

I used this command:

echo Never trust, always verify. > data.txt

```
C:\Windows\System32>echo Never trust, always verify. > data.txt

C:\Windows\System32>
```

```
C:\Windows\System32>type data.txt
Never trust, always verify.

C:\Windows\System32>
```

Step 2: Hash the file with SHA-256 using OpenSSL CLI

I used this command:

openssl dgst -sha256 data.txt

```
C:\Windows\System32>openssl dgst -sha256 data.txt
SHA2-256(data.txt)= 0253b56bfa6d0875980b05a3e089fd94c97accc9b8e8a4e7cec8062ba6abbbce

C:\Windows\System32>
```

Task 3B: HMAC using SHA-256

Step1. create HMAC with key secretkey123 for data.txt

I used this command:

openssl dgst -sha256 -hmac secretkey123 data.txt

```
C:\Windows\System32>openssl dgst -sha256 -hmac secretkey123 data.txt
HMAC-SHA2-256(data.txt)= 33fbf52890422c78372adb437b234f6cbea4adb760250c0fa053bf250fa93ff5
```

Task 3C: Integrity Check

Step 1: Change one letter in data.txt

I changed the last word verify to verifY (capital Y): and used this command:

echo Never trust, always verifY. > data.txt

```
C:\Windows\System32>echo Never trust, always verifY. > data.txt

C:\Windows\System32>type data.txt
Never trust, always verifY.

C:\Windows\System32>
```

Step 2: Recompute HMAC with the same key

For recomputing I used the same command:

openssl dgst -sha256 -hmac secretkey123 data.txt

The HMAC value changed completely compared to the original.

```
C:\Windows\System32>openssl dgst -sha256 -hmac secretkey123 data.txt
HMAC-SHA2-256(data.txt)= cf32edde98962e9fc10be4059f26d7535a97940bf37c8a41549e0869e15afb1f

C:\Windows\System32>
```

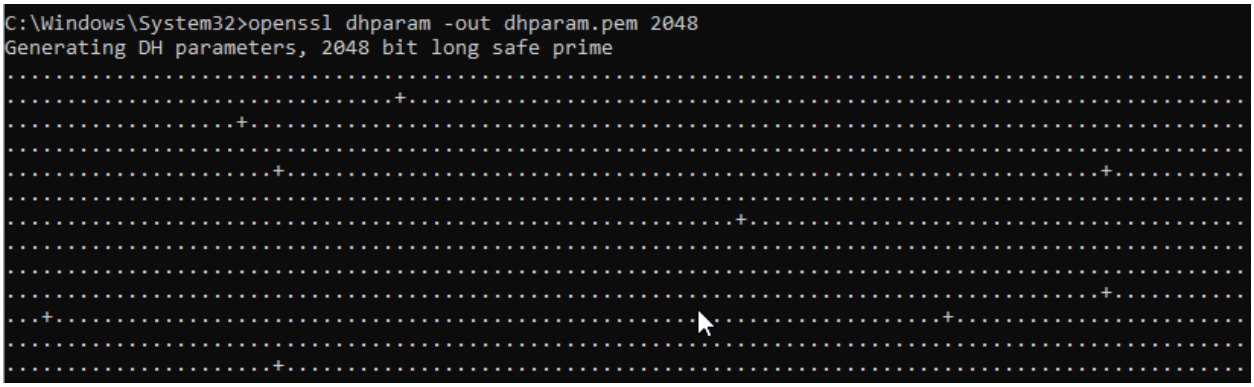
Task 3C – Integrity Check Explanation

I changed one letter in the file data.txt and then calculated the HMAC again using the same secret key secretkey123. The new HMAC value was very different from the first one. This happens because the HMAC depends on both the file's contents and the secret key. Even a tiny change in the file makes the HMAC change a lot. This is important because it helps us know if the file has been changed or tampered with. So, HMAC helps keep the data safe by making sure it hasn't been altered and comes from someone who knows the secret key.

Task 4A – Simulate DH Key Exchange with OpenSSL CLI

Step 1: Generate DH parameters (using 2048-bit MODP group)

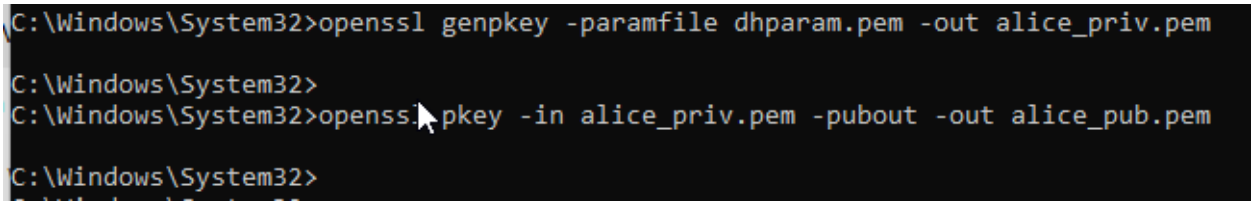
openssl dhparam -out dhparam.pem 2048



Step 2: Generate Alice's private key and public key

openssl genpkey -paramfile dhparam.pem -out alice_priv.pem

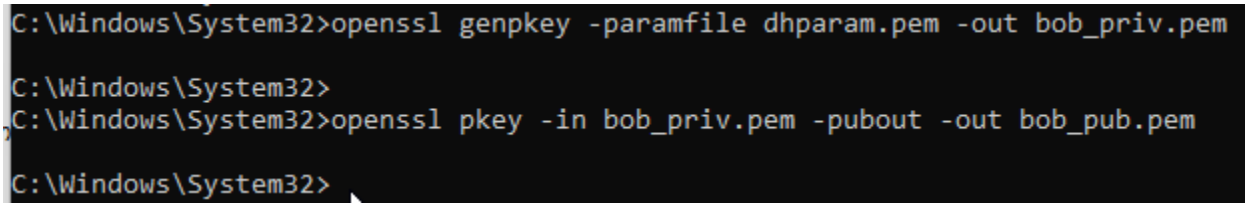
openssl pkey -in alice_priv.pem -pubout -out alice_pub.pem



Step 3: Generate Bob's private key and public key

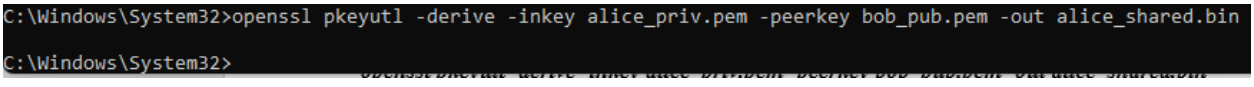
openssl genpkey -paramfile dhparam.pem -out alice_priv.pem

openssl pkey -in alice_priv.pem -pubout -out alice_pub.pem



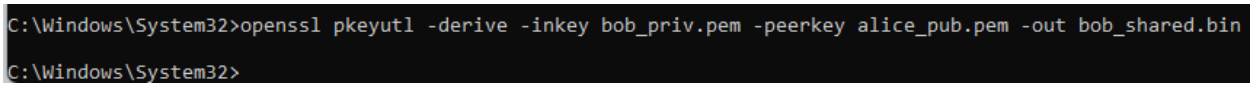
Step 4: Compute shared secret on Alice's side

openssl pkeyutl -derive -inkey alice_priv.pem -peerkey bob_pub.pem -out alice_shared.bin



Step 5: Compute shared secret on Bob's side

openssl pkeyutl -derive -inkey bob_priv.pem -peerkey alice_pub.pem -out bob_shared.bin



Step 6: Show Alice's and Bob's public keys

type alice_pub.pem

```
C:\Windows\System32>type alice_pub.pem
-----BEGIN PUBLIC KEY-----
MIICJTCCARcGCSqGSIb3DQEDATCCAQgCggEBALE+27Od0jw8Roy1U8u8lLyHKKRw
T036fW5goWpFCMjaJTG8YGqj8nlGgVlar7Ac7q9xYKec9QDBQu+KrfxxVnOvrgp7
yVa67Z1Mm0k2Nnm0S6aB+L8V9y5juwsKdV4cTlwv3N98Bl9GuH1g/R8yRBSPd4BZ
ky3UmfSo97tupMdlYvCvYBn2XNJmVxNL/jJljCEGpEpuUrFJgu7dtuHn1ABAW573
r6i58g4FuSGZWncqPiP/yVRnoDAUxpcBluiYwtEdZ4CnJsRaK5I7zvI6UMqs9ZQT
19NJvIQCSiqBqjSalFU56QTGPOhwwoYP+mu/we7cdpZlqVrT0koyuLfpuwcCAQID
ggEGAAKCAQEAlOehGj69tv4E0vmk1SzQZDVMDI/oQJZFcisixc/TDfvAotp0k9NA
Ehg6pgQ7Tt4stSHpLvKTE5scNU5NxgBQ3yKJ5JXwMiX9saCHA7bjL8P7S3CC6v5X
QMOAvK1EiqhCXzwxD6ylIWpSkQxbXywHyM4Uf4PoOHUUGD0G5ogIf/NnxhEexVOu
zX0fWtLHZ9DbHSqIlFd7m3s7u0N40MJl89UAMLDRg49n8ZKJcuYMwb3RdTiVFJkI
7J9lSctvqLkIKZOghibavQYhVORhbRbiKu55Uqcj/PcOSWhkSeEy5HGRbXwJ9vI7
hpvbDfA6TLfVWN/zzR6MpXn/QVEGec7M7A==
-----END PUBLIC KEY-----

C:\Windows\System32>
```

type bob_pub.pem

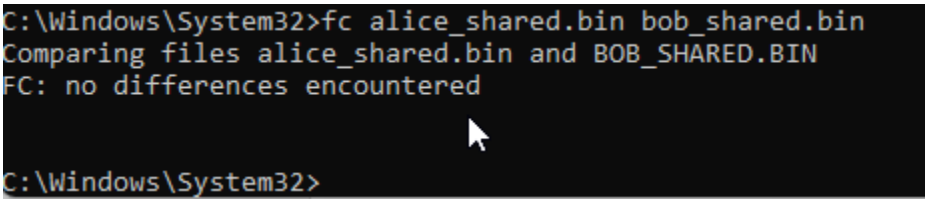
```
C:\Windows\System32>type bob_pub.pem
-----BEGIN PUBLIC KEY-----
MIICJTCCARcGCSqGSIb3DQEDATCCAQgCggEBALE+27Od0jw8Roy1U8u8lLyHKKRw
T036fW5goWpFCMjaJTG8YGqj8nlGgVlar7Ac7q9xYKec9QDBQu+KrfxxVnOvrgp7
yVa67Z1Mm0k2Nnm0S6aB+L8V9y5juwsKdV4cTlwv3N98Bl9GuH1g/R8yRBSPd4BZ
ky3UmfSo97tupMdlYvCvYBn2XNJmVxNL/jJljCEGpEpuUrFJgu7dtuHn1ABAW573
r6i58g4FuSGZWncqPiP/yVRnoDAUxpcBluiYwtEdZ4CnJsRaK5I7zvI6UMqs9ZQT
19NJvIQCSiqBqjSalFU56QTGPOhwwoYP+mu/we7cdpZlqVrT0koyuLfpuwcCAQID
ggEGAAKCAQEAq80SjejKei65YVsznS5nnVeLyaH1d01+MWm1W2Pi07s+IK0fczGW
WW4dlvq0HWIjTy7ZHwVdjsaFXTagpYHs5nOwLa0BMf6wSNHAILEBN4aJdPKhheNg
r1nxD3QHLRE/AcaJ+yiWJREqm1Db7JlDjmBznao/JrF7YJ0CJlirQ6gFoLgG93BP
lBOX4w/cr/zQ58r9LYbc/nE1bFIaovNzSqmFDyDNIzE0Rb9esRkPjU+B3yh03q+A
R7dszfV5tzmyuef2V3ajXlaFF9HAFa2IAAY1ilr0T1z4nRPjAdX4sFFquYf+0W6J
IPYbbcdLsaC5kU6x+oMLCDMsNMfcHELd7w==
-----END PUBLIC KEY-----

C:\Windows\System32>
```

Step 7: Check that both shared secrets are identical

fc alice_shared.bin bob_shared.bin

This output confirms that both parties independently derived the same shared secret, which is the core of how secure key exchange works.



Task 4B – Real-Life Application of Diffie-Hellman

Diffie-Hellman is used in real life to securely exchange encryption keys between two parties. One common use is in the TLS handshake, which happens when you visit a secure website (HTTPS). It's also used in secure messaging apps like Signal, where users can chat privately. Diffie-Hellman is important because it allows two people to agree on a shared secret key over the internet—even if someone is watching the connection. This makes sure their communication stays private and protected from attackers.