Laboratory 3. Cryptography

Task 1. Symmetric Encryption

Step 1. Create a txt file (**clear_message.txt**) with the text ***"Este es un mensaje en texto claro."***

```
[ ]  1 !echo "Este es un mensaje en texto claro" > clear_message.txt
```

```
[ ]  1 ls -la
```

```
total 28
drwxr-xr-x 1 root root 4096 May 23 01:08 ./
drwxr-xr-x 1 root root 4096 May 23 00:37 ../
-rw-r--r-- 1 root root   34 May 23 01:02 clear_message.txt
```

Step 2. Use `cat`, to verify its content.

```
1 !cat clear_message.txt
```

```
Este es un mensaje en texto claro
```

Step 3. Use `openssl` to create an **encrypted_message** file from the **clear_message.txt file** with:

```
!openssl aes-256-cbc -e -in clear_message.txt -out encrypted_message
```

When required, enter an encryption key. As shown in the following image.

```
[ ]  1 !openssl aes-256-cbc -e -in clear_message.txt -out encrypted_message

enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

Step 4. Verify the contents of the **encrypted_message file**.

```
1 !cat encrypted_message
```

```
Salted__ï¿½SmsBO!ï¿½ï¿½\Ñrï¿½tï¿½ï¿½~|ï¿½ï¿½|ï¿½vï¿½ï¿½ï¿½ï¿½ï¿½ï¿½WizcqyD~ï¿½ï¿½5ï¿½
```

Step 5. Decrypt the **encrypted_message** message to the **encrypted_message.txt** file, using:

```
!openssl aes-256-cbc -d -in encrypted_message -out decrypt_message.txt
```

Enter the key from the previous step.

```
[ ]  1 !openssl aes-256-cbc -d -in encrypted_message -out decrypt_message.txt

enter AES-256-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

Step 6. Verify that the message was decrypted using `cat`.

```
1 !cat decrypt_message.txt

Este es un mensaje en texto claro
```

Step 7. To strengthen the encryption use `-iter` y `-pbkdf2`. Create a new encrypted file **2_encrypted_message** from **clear_message.txt**.

```
!openssl aes-256-cbc -pbkdf2 -iter 1000 -e -in clear_message.txt -
out 2_encrypted_message
```

Step 8. Enter the password to encrypt the **file 2_encrypted_message**.

```
[ ]  1 !openssl aes-256-cbc -pbkdf2 -iter 1000 -e -in clear_message.txt -out 2_encrypted_message

enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
```

Step 9. Verify that the **2_encrypted_message** file was created.

```
[ ]  1 ls -la

total 36
drwxr-xr-x 1 root root 4096 May 23 01:27 ./
drwxr-xr-x 1 root root 4096 May 23 00:37 ../
-rw-r--r-- 1 root root   34 May 23 01:27
-rw-r--r-- 1 root root   64 May 23 01:25 2_encrypted_message
-rw-r--r-- 1 root root   34 May 23 01:02 clear_message.txt
drwxr-xr-x 4 root root 4096 May 21 13:23 .config/
-rw-r--r-- 1 root root   34 May 23 01:08 decrypt_message.txt
-rw-r--r-- 1 root root   64 May 23 01:06 encrypted_message
drwxr-xr-x 1 root root 4096 May 21 13:23 sample_data/
```

Step 10. Use `cat` to read the contents of the **2_encrypted_message** file.

```
1 !cat 2_encrypted_message

Salted__&^��73&p��{x�b����:� N��:t���O�;48�x��G�ű���3��.�
```

Step 11. Decrypt the **2_encrypted_message** file in the **decrypt_message** file with the key used for encryption.

```
[ ]  1 !openssl aes-256-cbc -pbkdf2 -iter 1000 -d -in 2_encrypted_message -out 2_decrypt_message

enter AES-256-CBC decryption password:
```

Step 12. Using `cat` verifies that the file was decrypted.

```
[ ]  1 !cat 2_decrypt_message

Este es un mensaje en texto claro
```

Task 2. Asymmetric Encryption

Step 13. Use `openssl` to generate a private key **private-key.pem** of size 2048 bits.

```
!openssl genrsa -out private-key.pem 2048
```

Step 14. Verify that the private key was created.

```
[ ]  1 ls -la

total 44
drwxr-xr-x 1 root root 4096 May 23 02:41 ./
drwxr-xr-x 1 root root 4096 May 23 00:37 ../
-rw-r--r-- 1 root root   34 May 23 01:27 2_decrypt_message
-rw-r--r-- 1 root root   64 May 23 01:25 2_encrypted_message
-rw-r--r-- 1 root root   34 May 23 01:02 clear_message.txt
drwxr-xr-x 4 root root 4096 May 21 13:23 .config/
-rw-r--r-- 1 root root   34 May 23 01:08 decrypt_message.txt
-rw-r--r-- 1 root root   64 May 23 01:06 encrypted_message
-rw------- 1 root root 1704 May 23 02:40 private-key.pem
```

Step 15. Get the public key **public-key.pem** from the private **key private-key.pem**.

```
[ ]  1 !openssl rsa -in private-key.pem -pubout -out public-key.pem

writing RSA key
```

Step 16. Reads the contents of the public key **public-key.pem**.

```
[ ]  1 !cat public-key.pem

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArTDgPRV99y3VJY4J3+x4
44b3bdU2cAEjAactge25hmX482tPMWJVNDcOGm79hbyjQrjiKIcAIfhQLi+hXP2U
CbvmNM7kjVyPSujvaJTS1mgvb98pzOzRVzHYZadt3NZJRI7MUSPXR0+ouShC+3RI
+MXAEf0S3XJTsNNb2aZNZuQ+kUH60q04vQmvgQ1BijqLt55UdJKcDP+VFT/ive06
iObPbDpNNK/z59TPCCZ+sEERVdldGXWy/JfzTPDXUgo8cY3Q8xblQaM7DpoECGGY
kyOZ4qP/Rdwyie9VpTNOHf5tV/4NGw76RuHYOUuHFEKZ1F+l+MdpBBomArcY6VuV
1QIDAQAB
-----END PUBLIC KEY-----
```

Step 17. Reads the contents of the private key **private-key.pem**.

```
[ ]  1 !openssl rsa -in private-key.pem -text -noout
```

```
Private-Key: (2048 bit, 2 primes)
modulus:
    00:ad:30:e0:3d:15:7d:f7:2d:d5:25:8e:09:df:ec:
    78:e3:86:f7:6d:d5:36:70:01:23:01:a7:2d:81:ed:
    b9:86:65:f8:f3:6b:4f:31:62:55:34:37:0e:1a:6e:
    fd:85:bc:a3:42:b8:e2:28:87:00:21:f8:50:2e:2f:
    a1:5c:fd:94:09:bb:e6:34:ce:e4:8d:5c:8f:4a:e8:
    ef:68:94:d2:d6:68:2f:6f:df:29:cc:ec:d1:57:31:
    d8:65:a7:6d:dc:d6:49:44:8e:cc:51:23:d7:47:4f:
    a8:b9:28:42:fb:74:48:f8:c5:c0:11:fd:12:dd:72:
    53:b0:d3:5b:d9:a6:4d:66:e4:3e:91:41:fa:d2:ad:
    38:bd:09:af:81:0d:41:8a:3a:8b:b7:9e:54:74:92:
    9c:0c:ff:95:15:3f:e2:bd:ed:3a:88:e6:cf:6c:3a:
    4d:34:af:f3:e7:d4:cf:08:26:7e:b0:41:11:55:d9:
    5d:19:75:b2:fc:97:f3:4c:f0:d7:52:0a:3c:71:8d:
    d0:f3:16:e5:41:a3:3b:0e:9a:04:08:61:98:93:23:
    99:e2:a3:ff:45:dc:32:89:ef:55:a5:33:4e:1d:fe:
    6d:57:fe:0d:1b:0e:fa:46:e1:d8:39:4b:87:14:42:
    99:d4:5f:a5:f8:c7:69:04:1a:26:02:b7:18:e9:5b:
    95:d5
publicExponent: 65537 (0x10001)
privateExponent:
    05:e2:7d:6e:bb:3e:62:a4:8d:33:af:a3:58:97:4d:
    8b:96:f2:ec:fe:f6:7b:66:21:27:e0:63:d4:a0:87:
    67:70:80:93:2e:28:17:35:50:50:2c:0c:ba:76:34:
    a9:68:d2:f7:f9:eb:3e:aa:f3:a4:b3:d8:c4:46:51:
    b5:24:ad:4a:8a:b6:be:ea:f1:6b:77:8e:80:e2:45:
    a6:e7:e3:a5:6d:b8:80:3f:1f:74:50:99:a9:07:a4:
    62:de:86:5b:9f:71:06:e2:29:41:6f:e1:29:3c:4f:
    84:81:c0:8b:90:69:c8:56:2e:79:63:55:7d:99:1c:
    61:e0:e9:f9:37:c4:35:2f:94:3c:a7:d7:ef:20:4d:
    7f:cd:5c:f2:51:31:20:83:b0:4f:23:5f:b7:26:ae:
    45:ad:f0:2d:08:12:ef:65:cc:ad:33:f3:08:d7:dc:
    5b:fc:9b:9f:8e:96:35:bb:be:db:7f:b5:47:b2:06:
```

Step 18. Encrypts the file **clear_message.txt** (from Task 1) into **rsa_ciphertext**, using the public key created earlier **public-key.pem**.

```
[ ]  1 !openssl pkeyutl -encrypt -in clear_message.txt -out rsa_ciphertext -inkey public-key.pem -pubin
```

Step 19. Verify that the **rsa_ciphertext** file was created.

```
[ ]  1 ls -la

total 48
drwxr-xr-x 1 root root 4096 May 23 02:49 ./
drwxr-xr-x 1 root root 4096 May 23 00:37 ../
-rw-r--r-- 1 root root   34 May 23 01:27 2_decrypt_message
-rw-r--r-- 1 root root   64 May 23 01:25 2_encrypted_message
-rw-r--r-- 1 root root   34 May 23 01:02 clear_message.txt
drwxr-xr-x 4 root root 4096 May 21 13:23 .config/
-rw-r--r-- 1 root root   34 May 23 01:08 decrypt_message.txt
-rw-r--r-- 1 root root   64 May 23 01:06 encrypted_message
-rw------- 1 root root 1704 May 23 02:40 private-key.pem
-rw-r--r-- 1 root root  451 May 23 02:41 public-key.pem
-rw-r--r-- 1 root root  256 May 23 02:49 rsa_ciphertext
drwxr-xr-x 1 root root 4096 May 21 13:23 sample_data/
```

Step 20. Read the contents of the **rsa_ciphertext file**.

```
[ ]  1 !cat rsa_ciphertext

fz
[illegible]
```

Step 21. Decrypt the **rsa_ciphertext** file with the private key **private-key.pem**, and obtain the **rsa_decrypt.txt file**.

```
!openssl pkeyutl -decrypt -in rsa_ciphertext -inkey private-key.pem
-out rsa_decrypt.txt
```

Step 22. Verify that the **rsa_decrypt.txt** file was created.

```
[ ]  1 ls -la

total 52
drwxr-xr-x 1 root root 4096 May 23 02:51 ./
drwxr-xr-x 1 root root 4096 May 23 00:37 ../
-rw-r--r-- 1 root root   34 May 23 01:27 2_decrypt_message
-rw-r--r-- 1 root root   64 May 23 01:25 2_encrypted_message
-rw-r--r-- 1 root root   34 May 23 01:02 clear_message.txt
drwxr-xr-x 4 root root 4096 May 21 13:23 .config/
-rw-r--r-- 1 root root   34 May 23 01:08 decrypt_message.txt
-rw-r--r-- 1 root root   64 May 23 01:06 encrypted_message
-rw------- 1 root root 1704 May 23 02:40 private-key.pem
-rw-r--r-- 1 root root  451 May 23 02:41 public-key.pem
-rw-r--r-- 1 root root  256 May 23 02:49 rsa_ciphertext
-rw-r--r-- 1 root root   34 May 23 02:51 rsa_decrypt.txt
drwxr-xr-x 1 root root 4096 May 21 13:23 sample_data/
```

Step 23. Use `cat` to read the contents of the **rsa_decrypt.txt file**.

```
1 !cat rsa_decrypt.txt

Este es un mensaje en texto claro
```

Task 3. Certificates

Task 3.1

Step 24. Use `openssl` to generate a certificate signing request using:

```
!openssl req -new -nodes -newkey rsa:4096 -keyout key.pem -out cert.csr
```

Step 25. Write the solicitation options.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:MX
State or Province Name (full name) [Some-State]:CDMX
Locality Name (eg, city) []:BENITO JUAREZ
Organization Name (eg, company) [Internet Widgits Pty Ltd]:TRINITY CORP
Organizational Unit Name (eg, section) []:HACKING TEAM
Common Name (e.g. server FQDN or YOUR name) []:TRINITY
Email Address []:HACK@TRINITYCORP.HACK
```

Step 26. Verify that files "**cert.csr**" and "**key.pem**" were created.

```
1 ls -la

total 32
drwxr-xr-x 1 root root 4096 Jun  5 21:59 ./
drwxr-xr-x 1 root root 4096 Jun  5 20:32 ../
-rw-r--r-- 1 root root 1769 Jun  5 21:59 cert.csr
drwxr-xr-x 4 root root 4096 Jun  4 13:31 .config/
-rw------- 1 root root 3272 Jun  5 21:59 key.pem
```

Go to https://certstream.calidog.io/ and visualise in real time the new certificates created and signed.

Task 3.2 Self Signed Certificate

Step 27. Use `openssl` to create a self-signed certificate (**domain.crt**) without an existing private key and CSR.

```
!openssl x509 -signkey key.pem -in cert.csr -req -days 365 -out domain.crt
```

Step 28. Verify if the certificate request self-signature was successful.

```
1 !openssl x509 -signkey key.pem -in cert.csr -req -days 365 -out domain.crt
```

```
Certificate request self-signature ok
subject=C = MX, ST = CDMX, L = BENITO JUAREZ, O = TRINITY CORP, OU = HACKING TEAM, CN = TRINITY, emailAddress = HACK@TRINITYCORP.HACK
```

Step 29. Use openssl to open in plain text the certificate created.

```
!openssl req -text -in cert.csr -noout -verify
```

```
1 !openssl x509 -text -noout -in domain.crt
```

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            55:50:88:d6:84:33:45:97:45:90:ff:b7:4a:8a:8f:57:50:f4:37:a3
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = MX, ST = CDMX, L = BENITO JUAREZ, O = TRINITY CORP, OU = HACKING TEAM, CN = TRINITY, emailAddress = HACK@TRINITYCORP.HACK
        Validity
            Not Before: Jun  5 22:14:29 2024 GMT
            Not After : Jun  5 22:14:29 2025 GMT
        Subject: C = MX, ST = CDMX, L = BENITO JUAREZ, O = TRINITY CORP, OU = HACKING TEAM, CN = TRINITY, emailAddress = HACK@TRINITYCORP.HACK
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:b8:ff:0a:3e:d8:20:06:e5:25:3d:27:45:d4:e4:
                    bc:29:25:61:a9:2d:1c:d3:6c:5d:cc:e2:b5:69:8b:
                    86:99:72:fe:22:f7:ca:db:42:99:1a:0b:e2:4c:08:
                    3f:a4:df:d2:26:fd:7e:0d:2e:70:2a:6f:7f:a5:78:
                    e5:48:e3:33:6c:c3:ca:ee:83:54:67:b5:3a:53:29:
```