GUÍA 4: OPENSSL Y JCE – ALGUNAS SOLUCIONES

Ejercicio 1:

1. OpenSSL soporta los siguientes algoritmos: (se pueden ver con **`openssl enc -list`**)
    - aes
    - aria (Algoritmo Coreano, del año 2003)
    - bf (BlowFish)
    - camellia
    - cast
    - chacha20 (es un algoritmo de cifrado de flujo)
    - des
    - idea
    - des3 (triple des)
    - desX (variante de des)
    - rc2 (Rivest Cipher para cifrado de bloque)
    - rc4 (Rivest Cipher para cifrado de flujo)
    - seed (otro algoritmo coreano, del año 1998)
    - sm4 (algoritmo de origen chino, del año 2012)
2. Los modos pueden ser: ECB, CBC, CFB (1 y 8), OFB, CTR.
3. AES tiene los siguientes modos de cifrado, que dependen del tamaño de la clave: 128, 192, 256 (bits de clave)
    - *ECB (128, 192, 256)*
    - *CBC (128, 192, 256)*
    - *CFB (128, 192, 256) (este hace un shift de 128 bits)*
    - *CFB1 (128, 192, 256) (shift de 1 bit)*
    - *CFB8 (128, 192, 256) (shift de 8 bits)*
    - *CTR (128, 192, 256)*
    - *OFB (128, 192, 256)*
4. El Vector de inicialización es un valor representado por dígitos hexadecimales. Si no se indica, debe poder extraerse de la password.
5. Si se elige la opción –K debe usarse con la opción –iv y ambos son una cadena hexadecimal

Si se elige la opción –k o –pass la clave y el iv se derivan de la password indicada. En este último caso se puede escribir –p para que muestre la clave y el iv utilizados. (-P lo muestra pero no encripta).

6. Para que cada cifrado sea distinto cuando se usa la misma contraseña, se usa el valor de Salt.
   Se puede indicar que se aplique un SALT en particular con la opción –S seguida de una cadena hexadecimal, aunque por omisión openSSL genera un SALT. Si no se desea usar SALT, se utiliza la opción –nosalt.
   El Salt se usa cuando se da contraseña. A partir de la contraseña (password) se genera Key e IV usando el Salt. Si se da directamente Key e IV, no usa salt. Para ver qué se generó a partir del password, se puede usar la opción -p.
   Por ejemplo, el algoritmo PBKDF2 de derivación de clave e iv tiene los siguientes parámetros de entrada:

```
DK = PBKDF2(PRF, Password, Salt, c, dkLen)
```

   Donde:
   - PRF es una funcion seudoaleatoria (puede ser por ejemplo HMAC)
   - Password es la contraseña
   - Salt es la secuencia de bits de salt
   - c es el número de iteraciones que se usará en el algoritmo
   - y dkLen es la cantidad de bits que debe tener la clave (key) deseada.

DK es la clave derivada.

GUÍA 4: OPENSSL Y JCE – ALGUNAS SOLUCIONES

Ejercicio 2:

a) En AES 128 modo CBC: (encripción)

```
$ echo "basta de mosquitos" > inarch
$ openssl enc -aes-128-cbc -in inarch -out salida -e -k "reloj" -pbkdf2 -a
$ cat salida
U2FsdGVkX1/a2b0OwurOmBACzKJsok2usLNziduM1IwQX+q8edaLsAI+k5SvV5Z2
```

(desencripción)

```
$ openssl enc -aes-128-cbc -in salida -out descifrado -d -k "reloj" -pbkdf2
-a
$ cat descifrado
basta de mosquitos
```

b) Volviendo a cifrar. Se usó SALT, ya que los resultados son distintos. Al usar SALT, los resultados son distintos.

```
$ openssl enc -aes-128-cbc -in inarch -out salida -e -k "reloj" -pbkdf2 -a
$ cat salida
U2FsdGVkX1+aZy4rkOwdqzJMYP0zFuUUuv4Ic/Sd1h4LMbzRpJX35CbXu0NNHyao
```

(desencripción)

```
$ openssl enc -aes-128-cbc -in salida -out descifrado -d -k "reloj" -pbkdf2
-a
$ cat descifrado
basta de mosquitos
```

c) Realizando AES 128 en modo CBC dos veces seguidas **sin salt**:

```
$ openssl enc -aes-128-cbc -in inarch -out salida -e -k "reloj" -pbkdf2 -a -
nosalt
$ cat salida
orkv3Abh4wSPyYMhOJoQzhWaMWscCaSknKkbyTepUfo=
$ openssl enc -aes-128-cbc -in inarch -out salida -e -k "reloj" -pbkdf2 -a -
nosalt
$ cat salida
orkv3Abh4wSPyYMhOJoQzhWaMWscCaSknKkbyTepUfo=
```

Sin usar SALT, los resultados coinciden.

d) En el caso de cifrar con **modo ECB**, hay valores que coinciden (cada bloque se cifra de manera independiente, entonces bloques iguales cifran a cosas iguales).

```
$ echo "buen diabuen diabuen diabuen dia">intext
$ openssl enc -aes-128-ecb -in inarch -out salida -e -k "reloj" -pbkdf2
$ cat salida
Salted__\]H hbyD Ve Ц a hbyD Ve Ц a !cz+       =%
$ openssl enc -aes-128-ecb -in inarch -out salida -e -k "reloj" -pbkdf2
$ cat salida
Salted__ q siU tX Tu :9{s U tX Tu :9{s co!sy 7
```

Ejercicio 3:

```
$ openssl enc -aes-128-cbc -in ej3Cifrado -out descifrado -d -k "margarita"
-pbkdf2 -a -nosalt
$ cat descifrado
ya tenemos la tercera
```

GUÍA 4: OPENSSL Y JCE – ALGUNAS SOLUCIONES

Ejercicio 4:

```
$ openssl enc -d -in ej4C -out ej4D -aes-128-cbc -K "1234A" -iv "AAFF" -a -p
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
salt=8030DAF3B75D0000
key=1234A000000000000000000000000000
iv =AAFF0000000000000000000000000000
$ cat ej4D
por ahora es fácil Criptografia
```

a) La clave es de 128 bits = 16bytes
b) En hexa realmente se uso:

```
key=1234A000000000000000000000000000
iv =AAFF0000000000000000000000000000
```

Ejercicio 5:

```
$ python3 ceros.py>ceros
$ openssl enc -e -aes-128-ecb -in ceros -out ej5ecb -k "cripto" -pbkdf2 -a
$ cat ej5ecb
U2FsdGVkX1/8igL0cVWUG67skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Ku7JGMbbJsZ5fcFE3wnduC
ruyRjG2ybGeX3BRN8J3bgq7skYxtsmxnl9wUTfCd24Jm6AyzpfZGNTPmWR7KdVuz
```

```
$ openssl enc -e -aes-128-cbc -in ceros -out ej5cbc -k "cripto" -pbkdf2 -a
$ cat ej5cbc
U2FsdGVkX19WYGNoinIcRs8fa022ZjKRwi/BwbRMOUs1TP7hNhh4qJCNk+hH/SHl
bqxsEdnLznmtbaKWRXrs0BXJwPbRqoSD1hzvKQZcvTIaCkbCPptyk0HqdV3KzeiS
T9729cDSDnwhIuw7jb7vnRcq9thrUL9tnLvjrYoj+3EbV4b/6nQh9emVwNpzkHYZ
w9O+6FXPXZF+vWNIa3WElB/xgkk3tRqr0nSzkAsZQi6y/gfCznODHWcEVWkAy8L8
TvwOi5nltUcmvU4txcQB/utfHJgffggu/f08ZPA3n/tV0AbP7Qy4sO30oy6b9fmn
fDonNYsn7hd4AbvVwaUtFDkTJXWBCl91KH/ys9CkaQ1q4wq5+ZjaM0nq8q7Rgqz3
niMXgQqq7fd04Wq/HLIUSIJlCYxhSDQ7cKAuLUFhlfGj1HKqkFODaPr9t45R/hgh
O0BLFvx7Fmp54ienh9njHV0oCL8N9eEqixL9sBV0XQ1taV1D2VBtcfZ06P21WxHf
XX4oK/+GrIChiYzOPQ1RZ7upXHtq9FDfKS4I7O5kdih8PuUm5dXJvolPZ+IJW31X
L+xpxL4MzF9pL1w6Tc1XN/HcLzzhNnvVE6UeehBH+d+brTxA89lkVwjdVDKrjVBt
M3+ZUV7bbkKeJ5mwVruSqdOvt2w2KKSWngG16OCga/ORHwwjifMObIdRfRiYjKKk
MwQFg4I+mLxFPNTGuPwLFND5/WoM1nbYxqU0SKxSzpTuhD9WIkgUj8rV+/07cR8H
DxwMJNHnsRoRiaJaYGy5BVvWw5Q3a+uSJNuWFIiIYvn2IfeodmiTpLAarm+wzp8P
```

```
bLLKyPz9aT9skBOz+PFkFXsuSeWdRp3I9bwEcmxZyEkyUBg32JdddBfDHReDJFGo
gIpPBu3YOCtVDIvlkHj1v/cRXOluKdjlY7zrQIJVII0ghqpd467mK6iL4D8AxJRa
oH7vdkIVwhz7ADo0IqvUWOrHs5Ul4YV6ETD5KEWSKTY/kPquY4+OuagH3zWvC+tG
t4acgUClD3Sodlrt2uMRUdVYdtsM0Tq0CyZqIGgWjWQZ6vkOt8g6tbL9IETDC1cz
wQWo8j6iwZd4vNsV9MuOKx0b0ydpX9tyfBm4k35uP5jEEtbIlDdVRRty04GkK13z
DxrCyDUFLiHV5f/GbAm3+7yNgb/uVjp0TeGD5DdocaQNSpImD9ICU+ToKUZpJLUk
5JZ0SwOxcN1RcByM1yMasbmdTCNyR4ZO3M/TjCh5xCyAi7qd/ITIpJCZqq/qICt2
Nb9TDDfkgSOqj50gcndnBUleDAxWsvUhxnHj6K7UQNK/K4k8lARrSVplsaNDiBd6
WOHNr3Gb/u4F1AP6G1DB2P3T6y0KYz7D3iRs+54pAaFGdvTt6qOqGQ4tFqTE4JHt
```

En el primer caso hay bloques iguales, en el segundo no.

Ejercicio 6:

```
$openssl genrsa –out priv.key 1024
```

Ejercicio 7:

```
$openssl rsa –in priv.key –out pub.key -pubout
```

La clave “privada”:

```
$ cat priv.key
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQC93hmcVEo2IuH0GDjwh13YEQZVxpuA9v4YDQjclKFDY2TbFXHn
iDYBrz48XJ9IXNhbVfpWCnAc0UfHmPxDae/XCMy1JO/J85lMcEkykvsMvA0P/AXi
LMM7b29hLUlYLVMJ3407A7Pa46ibUJQr6yS614MXxBBik41ASJ+VaGDHFwIDAQAB
AoGAa4Uv6aiOEhQ08t8Yt5Vg3m/dr4t2GlVE1WfkQc3leNz7i1mclGo9czzDOLmW
9gql9ys8RadJj/gv9uc6zA5c4+ZxL4IJiraaiPKQo7Y/5mfH4oHLWhYT/z0kVzNO
NjivMEehu7OiEwhemb7/YWmOaoQ8i1dkkCXl0sKNl3wgt5ECQQDhwopl31ZcdJkG
iWs6Kbjqxypyeo5sfwl/jDT5oSbhzms3RS2Ipe7dlhl/fc1ZojbAVngykwNTkRyL
pY1lXjC5AkEA10zJaoYaX2EhDcMhD6lnr3w/7c/5tBkWlEeogOc4tDmAnI4Vkb/2
+XVqI6ITYBaAg0/ABe0XxW5UOj1AJ8KuTwJAdBa5XBEwM3Kxja70OIBj2jZ4GU30
Fs5sb+4E/6hiehLPByjHdVD+N2uLyQdDpSfIx7avMDJwr3QbHaQPxWRd4QJBANCU
771n1jz5WJCArpMByBXXBghbKMs2Uw2srq7TU7gmzVbewD7H/3mGKyICb0r0AeU1
ag9Tt2fqj2cs1+6tSTkCQC88PFTiq/juNVEJLtlnJuSFaBlPoOp88PMc4WGxkSO8
Ne+IzPzigWC7fio6XmHQZYtmfPXgIXyweM9CnbCFfCg=
-----END RSA PRIVATE KEY-----
```

La clave se guarda en formato base 64.
Para ver en formato texto ponemos:

```
$ openssl rsa -in priv.key -text -out privada.txt
writing RSA key
$ cat privada.txt
RSA Private-Key: (1024 bit, 2 primes)
modulus:
    00:bd:de:19:9c:54:4a:36:22:e1:f4:18:38:f0:87:
    5d:d8:11:06:55:c6:9b:80:f6:fe:18:0d:08:dc:94:
    a1:43:63:64:db:15:71:e7:88:36:01:af:3e:3c:5c:
    9f:48:5c:d8:5b:55:fa:56:0a:70:1c:d1:47:c7:98:
    fc:43:69:ef:d7:08:cc:b5:24:ef:c9:f3:99:4c:70:
    49:32:92:fb:0c:bc:0d:0f:fc:05:e2:2c:c3:3b:6f:
    6f:61:2d:49:58:2d:53:09:df:8d:3b:03:b3:da:e3:
    a8:9b:50:94:2b:eb:24:ba:d7:83:17:c4:10:62:93:
    8d:40:48:9f:95:68:60:c7:17
publicExponent: 65537 (0x10001)
privateExponent:
    6b:85:2f:e9:a8:8e:12:14:34:f2:df:18:b7:95:60:
    de:6f:dd:af:8b:76:1a:55:44:d5:67:e4:41:cd:e5:
    78:dc:fb:8b:59:9c:94:6a:3d:73:3c:c3:38:b9:96:
    f6:0a:a5:f7:2b:3c:45:a7:49:8f:f8:2f:f6:e7:3a:
```

```
    cc:0e:5c:e3:e6:71:2f:82:09:8a:b6:9a:88:f2:90:
    a3:b6:3f:e6:67:c7:e2:81:cb:5a:16:13:ff:3d:24:
    57:33:4e:36:38:af:30:47:a1:bb:b3:a2:13:08:5e:
    99:be:ff:61:69:8e:6a:84:3c:8b:57:64:90:25:e5:
    d2:c2:8d:97:7c:20:b7:91
prime1:
    00:e1:c2:8a:65:df:56:5c:74:99:06:89:6b:3a:29:
    b8:ea:c7:2a:72:7a:8e:6c:7f:09:7f:8c:34:f9:a1:
    26:e1:ce:6b:37:45:2d:88:a5:ee:dd:96:19:7f:7d:
    cd:59:a2:36:c0:56:78:32:93:03:53:91:1c:8b:a5:
    8d:65:5e:30:b9
prime2:
    00:d7:4c:c9:6a:86:1a:5f:61:21:0d:c3:21:0f:a9:
    67:af:7c:3f:ed:cf:f9:b4:19:16:94:47:a8:80:e7:
    38:b4:39:80:9c:8e:15:91:bf:f6:f9:75:6a:23:a2:
    13:60:16:80:83:4f:c0:05:ed:17:c5:6e:54:3a:3d:
    40:27:c2:ae:4f
exponent1:
    74:16:b9:5c:11:30:33:72:b1:8d:ae:f4:38:80:63:
    da:36:78:19:4d:f4:16:ce:6c:6f:ee:04:ff:a8:62:
    7a:12:cf:07:28:c7:75:50:fe:37:6b:8b:c9:07:43:
    a5:27:c8:c7:b6:af:30:32:70:af:74:1b:1d:a4:0f:
    c5:64:5d:e1
exponent2:
    00:d0:94:ef:bd:67:d6:3c:f9:58:90:80:ae:93:01:
    c8:15:d7:06:08:5b:28:cb:36:53:0d:ac:ae:ae:d3:
    53:b8:26:cd:56:de:c0:3e:c7:ff:79:86:2b:22:02:
    6f:4a:f4:01:e5:35:6a:0f:53:b7:67:ea:8f:67:2c:
    d7:ee:ad:49:39
coefficient:
    2f:3c:3c:54:e2:ab:f8:ee:35:51:09:2e:d9:67:26:
    e4:85:68:19:4f:a0:ea:7c:f0:f3:1c:e1:61:b1:91:
    23:bc:35:ef:88:cc:fc:e2:81:60:bb:7e:2a:3a:5e:
    61:d0:65:8b:66:7c:f5:e0:21:7c:b0:78:cf:42:9d:
    b0:85:7c:28
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQC93hmcVEo2IuH0GDjwh13YEQZVxpuA9v4YDQjclKFDY2TbFXHn
iDYBrz48XJ9IXNhbVfpWCnAc0UfHmPxDae/XCMy1JO/J85lMcEkykvsMvA0P/AXi
LMM7b29hLUlYLVMJ3407A7Pa46ibUJQr6yS614MXxBBik41ASJ+VaGDHFwIDAQAB
AoGAa4Uv6aiOEhQ08t8Yt5Vg3m/dr4t2GlVE1WfkQc3leNz7i1mclGo9czzDOLmW
9gql9ys8RadJj/gv9uc6zA5c4+ZxL4IJiraaiPKQo7Y/5mfH4oHLWhYT/z0kVzNO
NjivMEehu7OiEwhemb7/YWmOaoQ8i1dkkCXl0sKNl3wgt5ECQQDhwopl31ZcdJkG
iWs6Kbjqxypyeo5sfwl/jDT5oSbhzms3RS2Ipe7dlhl/fc1ZojbAVngykwNTkRyL
pY1lXjC5AkEA10zJaoYaX2EhDcMhD6lnr3w/7c/5tBkWlEeogOc4tDmAnI4Vkb/2
+XVqI6ITYBaAg0/ABe0XxW5UOj1AJ8KuTwJAdBa5XBEwM3Kxja70OIBj2jZ4GU30
Fs5sb+4E/6hiehLPByjHdVD+N2uLyQdDpSfIx7avMDJwr3QbHaQPxWRd4QJBANCU
771n1jz5WJCArpMByBXXBghbKMs2Uw2srq7TU7gmzVbewD7H/3mGKyICb0r0AeU1
ag9Tt2fqj2cs1+6tSTkCQC88PFTiq/juNVEJLtlnJuSFaBlPoOp88PMc4WGxkSO8
Ne+IzPzigWC7fio6XmHQZYtmfPXgIXyweM9CnbCFfCg=
-----END RSA PRIVATE KEY-----
```

La clave “pública” en base 64:

```
$ cat pub.key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC93hmcVEo2IuH0GDjwh13YEQZV
xpuA9v4YDQjclKFDY2TbFXHniDYBrz48XJ9IXNhbVfpWCnAc0UfHmPxDae/XCMy1
JO/J85lMcEkykvsMvA0P/AXiLMM7b29hLUlYLVMJ3407A7Pa46ibUJQr6yS614MX
xBBik41ASJ+VaGDHFwIDAQAB
```

```
-----END PUBLIC KEY-----
```

Vemos con formato texto qué tiene: (ojo que acá hay que poner -pubin)

```
$ openssl rsa -in pub.key -text -pubin -out publica.txt
writing RSA key
$ cat publica.txt
RSA Public-Key: (1024 bit)
Modulus:
    00:bd:de:19:9c:54:4a:36:22:e1:f4:18:38:f0:87:
    5d:d8:11:06:55:c6:9b:80:f6:fe:18:0d:08:dc:94:
    a1:43:63:64:db:15:71:e7:88:36:01:af:3e:3c:5c:
    9f:48:5c:d8:5b:55:fa:56:0a:70:1c:d1:47:c7:98:
    fc:43:69:ef:d7:08:cc:b5:24:ef:c9:f3:99:4c:70:
    49:32:92:fb:0c:bc:0d:0f:fc:05:e2:2c:c3:3b:6f:
    6f:61:2d:49:58:2d:53:09:df:8d:3b:03:b3:da:e3:
    a8:9b:50:94:2b:eb:24:ba:d7:83:17:c4:10:62:93:
    8d:40:48:9f:95:68:60:c7:17
Exponent: 65537 (0x10001)
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC93hmcVEo2IuH0GDjwh13YEQZV
xpuA9v4YDQjclKFDY2TbFXHniDYBrz48XJ9IXNhbVfpWCnAc0UfHmPxDae/XCMy1
JO/J85lMcEkykvsMvA0P/AXiLMM7b29hLUlYLVMJ3407A7Pa46ibUJQr6yS614MX
xBBik41ASJ+VaGDHFwIDAQAB
-----END PUBLIC KEY-----
```

Entonces, el archivo con la clave privada, guarda información para obtener la clave pública, ya que guarda el módulo N, e y d. En cambio, el archivo con la clave pública sólo tiene información pública: el módulo N y e. El módulo tiene 1024 bits.[1]

Ejercicio 9:

- ¿Cuántas personas podrían enviarte un mensaje encriptado correctamente?

Cualquier persona que posea mi clave pública.

- ¿Puede alguien enviarte un mensaje haciéndose pasar por otra persona?

Si, eso es posible.

Ejercicio 10:

- ¿Con qué clave deberá desencriptar el mensaje tu compañero?

Con mi clave pública.

- ¿Puede tener la duda de que vos se lo enviaste? ¿Por qué?

No, sólo yo con mi clave privada pude encriptarlo.

- ¿Se mantiene la confidencialidad del mensaje? ¿Por qué?

No, cualquiera puede abrir el mensaje y ver lo qué dice.

- ¿Qué se aseguró con este procedimiento?

Se aseguró autenticidad de origen.

Ejercicio 11:

- ¿Cómo sabrá tu compañero que el archivo que le enviaste fue enviado por vos?

Al hacer el hash del documento enviado, debe coincidir con la desencripción con mi clave pública del archivo cifrado.

[1] Si te interesa saber más sobre este tema: https://www.flu-project.com/2019/10/generacion-claves-rsa-Parte-3.html (visitada abril 2024)

GUÍA 4: OPENSSL Y JCE – ALGUNAS SOLUCIONES

Ejercicio 13:

Para firmar:

```
$ openssl dgst -sha1 -sign priv.key -out salidafirmada des.pdf
```

Para verificar:

```
$ openssl dgst -sha1 -verify pub.key -signature salidafirmada des.pdf
Verified OK
```

Debe obtenerse Verify OK.