

Práctica 2: Criptosistemas asimétricos

Ángel Gómez Martín

agomezma@correo.ugr.es

Seguridad y Protección de Sistemas Informáticos

UGR 2018-19

Tareas

Para determinar la sintaxis de las ordenes a utilizar me he basado en el [manual de OpenSSL](#).

1

Para generar el par de claves pública y privada usando el algoritmo RSA, en formato PEM, sin contraseña y de 901 bits ejecuto lo siguiente:

```
openssl genrsa -out angelRSAkey.pem 901
```

- *genrsa*: Genera la clave con RSA.
- *-out nombre.pem*: Archivo de salida. En este caso formato PEM.
- *size*: Tamaño en bits de la clave. En este caso 901 bits.

```
✓ angel@juggernaut:1 → openssl genrsa -out angelRSAkey.pem 901
Generating RSA private key, 901 bit long modulus
.....+++++
.....+++++
e is 65537 (0x010001)
✓ angel@juggernaut:1 → cat angelRSAkey.pem
-----BEGIN RSA PRIVATE KEY-----
MIICFQIBAAJxGov73JA8WPLW8xeIkrTAh4bTSC9n1SDQ7Yc/Ss/zbvLL/TzczXNw
ZKZi5J8KKL3KHLh9cpFRxjBaX2+ztVKUYlkLFdmbNjgIZls6TIKJiQ0TdjRZFThK
FSYk+PYLtjI6Sodlh3qzwpQVngrmg8njq/UCAwEAAQJxCHWbKMn9sQyjnItfRgC7
aXSZV2Nj3xK6ATccACCMF8zxXh1K711d73TmBpx3jEwOYGuqJvCRzmVz57QGL52
RHm/ja8/gcpb1UGEDLsM8D6J43WlnUrHwX69Y+QJDsjZksELkC0nvpRSBR8n1s2n
/vECQd38n+ybK29IVbpZ1DyCsy2DdxnSiCLalShaPDuV+MJ1Epg3DY1gysaqWjP
PvIwwDKT6ERg0dRKEwISA43yl0nEmZQz+hntP+hdUHu4+khHD0puS24suUPZoM0r
bQXRmN0WUf7UQBNGYnMIyS6A0ECdj3LXAjkc+4fAL0TvbSqvTS6RSNe/MTwS99y
d7qS6ZK7BLPcW0SIH35Rt7UWNa0J05nTplxam5fQDsLiDkCOQMURBn9BpKFmH97
h0jprdn2zXhThNnNIHpJw7j7rWj2xRR2FSm5Fvm7bqsXmfmg+CazOMmZqp0owI5
ByCIg8HsXY5S7/OPw7qIower5KPEqLyZLwq1pyZfMaT+3Pyi7svC5FzAn6GcCSkP
rv0klHYo/Fmb
-----END RSA PRIVATE KEY-----
```

2

Para extraer la clave privada cifrada con AES-128 utilizo la siguiente orden:

```
openssl rsa -in angelRSAkey.pem -out angelRSApriv.pem -aes128
```

```
✓ angel@juggernaut:2 ➔ openssl rsa -in angelRSAkey.pem -out angelRSApriv.pem -aes128
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
✓ angel@juggernaut:2 ➔ cat angelRSApriv.pem
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,9EFF8B9544B96C48B471854D4EDE0FB3

SJ1k6opkSqUfVl6WLoog7A1JC6asKVf0u87tnFgxI0R2znFD4RvDlkXDYY8/07U6
5dUiFZy0CdGLMe0PogkUn6ED32laTnhmxcfsVDsmB1DKax90rkQ8KJ+cMH9YFzke
9b7eay1e+9+4B2RS2SG7tnZ9jF8izyMVUu06x+2u7qo0K+rmZBd88qQuirIQ0023
Y/Vzh6ZPW93sVtfWWNPrRjxxqz0gr6ibGHYhNXjzTFLvpLgHXbib8q0p03+n/nLs
hETYT7c8W2dT0TguYNmd7+oD+r8JN2wCpAXtg6XYQ4Vxj1kgoVBHDDlhofZst6Iy
oLJuC/Gey8QEw5X7sHwHMF3tYXbSe4xqHMptnkZeVfL/pVFeA1FhQ5mE69pWK7MO
6ndIDkp9I+0+J3e8QS+IuMMx+BufNcQu0n8dSKH9bRWXgpJE1js34EbyB8nxh3ac
HWrVh4YTk0bkkkgkL/FG7IgvLDbzVeyVzkX59gx7bYk2DwRE8ek5DTEeyWlqPasy
C+THF30ehz54tqKpqJ+1/ydQFLU9tqME3hkeQBc87wvDF0NWDiEcVBl3rDrISsoo
DThXRZrRMI3z5d5NUL0NkFtvwupNJC8tutLnK2l7ubbeT8rE68AhpLy2rW50/csn
sN6HTd9v3h9JJ0PYKn/uJa6oMhJEyugz0WSig4/aBaQD9UupaFYNfygM4bu4kQsB
KsfturK1eGNsQjnkMZA6xg==
-----END RSA PRIVATE KEY-----
```

- `openssl rsa` es la orden utilizada para manejar claves RSA.
- Como contraseña he utilizado: 0123456789

3

Extraigo la clave pública con la orden siguiente:

```
openssl rsa -in angelRSAkey.pem -out angelRSApub.pem -pubout
```

```
✓ angel@juggernaut:2 ➔ openssl rsa -in angelRSAkey.pem -out angelRSApub.pem -pubout
writing RSA key
✓ angel@juggernaut:2 ➔ cat angelRSApub.pem
-----BEGIN PUBLIC KEY-----
MIGMA0GCsQGSib3DQEBAQUAA3sAMHgCcRqL+9yQPFj5VvMXiJK0wIeG00gvZ9Ug
002HP0rP8275S/083M1zcGSmYuSfCi9yh5YfXKRUCYwWl9vs7VSlGJZCxXZmzY4
CGZbokyCiYkDk3Y0WRU4ShUmJPj2C7Yy0uaHZYd6s8KUFZ4K5oPJ46v1AgMBAAE=
-----END PUBLIC KEY-----
```

- `-pubout`: Extrae la clave pública.
- No es necesario añadir `-aes128` pues no queremos que la clave pública esté cifrada.

4

La orden utilizada es la siguiente:

```
openssl rsautl -in a1024.bin -out a1024_encrypt.bin -inkey angelRSApub.pem -pubin -
encrypt
```

- *-in archivo*: Archivo de entrada.
- *-out archivo*: Archivo de salida.
- *-inkey key*: Clave usada para cifrar.
- *-pubin*: Indica que se está introduciendo la clave pública.
- *-encrypt*: Indica que se quiere cifrar.

```
✓ angel@juggernaut:4 ➔ openssl rsautl -in a1024.bin -out a1024_encrypt.bin -inkey angelRSAPub.pem -pubin
  -encrypt
RSA operation error
140550604042688:error:0406D06E:rsa routines:RSA_padding_add_PKCS1_type_2:data too large for key size:../
crypto/rsa/rsa_pk1.c:125:
```

El error que aparece indica que el tamaño del archivo de datos a cifrar es demasiado grande para esta clave. Esto se produce porque para cada tamaño de clave sólo se puede cifrar hasta un tamaño máximo de bytes.

Para calcular este tamaño pasamos a bytes el tamaño de la clave y le restamos un padding (11 bytes PKCS#1 v1.5). En el caso de nuestra clave de 901 bits el tamaño máximo del archivo debería ser:

$901/8 - 11 = 112,625 - 11 = 101,625 \sim 101$ bytes.

Puesto que el archivo (*a1024.bin*) pesa 256 bytes (1024 bits), no puede ser cifrado con esta clave.

Referencias:

<https://tls.mbed.org/kb/cryptography/rsa-encryption-maximum-data-size>

<https://tools.ietf.org/html/rfc2313>

5

1

Como criptosistema simétrico elijo AES-192-OFB. La elección ha sido arbitraria pues ha sido el primer criptosistema que he visto al revisar la práctica anterior.

2

Genero el archivo *sessionkey* con la siguiente orden:

```
openssl rand -out sessionkey -hex 24
```

- *rand*: Para generar una cadena aleatoria.
- *-out archivo*: Archivo de salida.
- *-hex*: Cadena en hexadecimal.
- *bytes*: Tamaño en bytes de la cadena. En este caso 24, pues AES-192-OFB necesita una clave de 192 bits (24 bytes).

Modifico el archivo *sessionkey* y le añado *-aes-192-ofb*.

```
✓ angel@juggernaut:5 ➔ cat sessionkey
189752ffc4e4688dc4d2474bd3d3c4486a411dcb8eb4e376
-aes-192-ofb
```

3

Para cifrar el archivo *sessionkey* utilizo la orden siguiente (es la misma que se usa para cifrar el archivo *a1024.bin*):

```
openssl rsautl -in sessionkey -out sessionkey_crypt -inkey angelRSApub.pem -pubin -encrypt
```

```
✓ angel@juggernaut:5 ➔ openssl rsautl -in sessionkey -out sessionkey_crypt -inkey angelRSApub.pem -pubin -encrypt
✓ angel@juggernaut:5 ➔ cat sessionkey_crypt
❖❖) [B][B] 8[B]/E❖❖❖❖J [B][B] [B][B] C❖❖
❖❖❖❖mr❖)V❖TV||❖❖❖❖{
```

4

La orden usada para cifrar el archivo *a1024.bin* usando el criptosistema simétrico es la siguiente:

```
openssl enc -aes-192-ofb -in a1024.bin -out a1024_crypt.bin -pass file:sessionkey
```

6

Utilizo la orden anterior para cifrar el archivo *a1024.bin*, obteniendo el siguiente resultado:

```
53 61 6C 74 65 64 5F 5F EE BD AA CA E3 CF 04 B4
CB 87 27 29 34 2E 2C 83 68 F3 C6 1D 64 9F A7 A2
D0 C4 29 6D 6A 6C 25 ED 15 99 F4 20 3F C7 8E 63
E2 C5 AC C7 FB 4E 51 34 D0 DB C4 2C E5 2E 12 2C
E8 B7 9D DF 51 AA 25 87 1C C7 33 74 2D 98 EB 73
0B C8 D5 A9 84 44 A8 C6 83 BC 4E 99 DC 06 71 BE
AF 60 D2 67 6E 07 A9 11 B4 07 64 DC D3 46 E6 C4
66 FD E3 E5 37 3B 9B F7 03 56 3F 61 80 16 A5 D1
A4 99 6C D8 F0 25 3F 51 E3 09 DA 58 F6 20 23 AC
```

En la anterior imagen se puede observar el archivo cifrado usando el sistema simétrico que he creado en el ejercicio anterior. Ahora paso a descifrar el archivo con la clave privada para ver si coincide con el archivo original.

En primer lugar descifro el archivo *sessionkey_crypt*, que es el que me permitirá descifrar el mensaje, para ello utilizo la siguiente orden:

```
openssl rsautl -in sessionkey_crypt -out sessionkey_decrypt -inkey angelRSApriv.pem -decrypt
```

```
✓ angel@juggernaut:6 ➔ openssl rsautl -in sessionkey_crypt -out sessionkey_decrypt -inkey angelRSApriv.pem -decrypt
Enter pass phrase for angelRSApriv.pem:
✓ angel@juggernaut:6 ➔ cat sessionkey_decrypt
189752ffc4e4688dc4d2474bd3d3c4486a411dcb8eb4e376
-aes-192-ofb
✓ angel@juggernaut:6 ➔
```

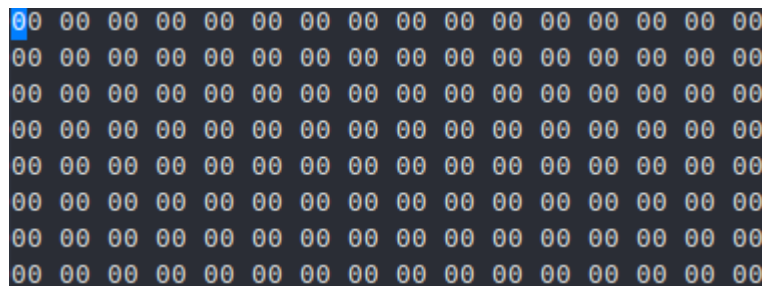
Al descifrar usando la clave privada se necesita una contraseña, ésta es la cifra dicha clave (la que puse en el ejercicio 2). Por otro lado se observa que el archivo *sessionkey_decrypt* se ha descifrado correctamente.

Ahora utilizando este archivo puedo descifrar el archivo *a1024_crypt.bin* para ver si coincide con el original; para ello uso la orden que usaba en el ejercicio 5.4, pero cambiando algunos flags:

```
openssl enc -aes-192-ofb -d -in a1024_crypt.bin -out a1024_decrypt.bin -pass
file:sessionkey_decrypt
```

- *-d*: Indica que queremos descifrar.

El archivo (*a1024_crypt.bin*) se descifra correctamente (en *a1024_decrypt.bin*) y vemos que contiene el mismo contenido que el archivo original (*a1024.bin*)

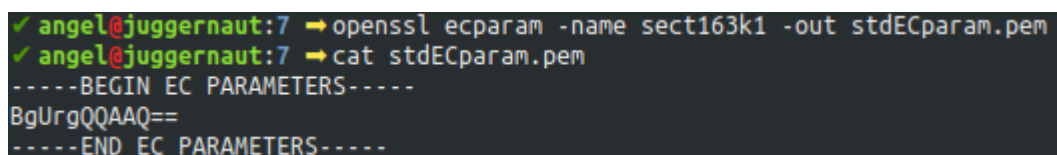


7

La curva que he elegido es B-163, que corresponde en OpenSSL con *sect163k1*. Para generar el archivo *stdECparam.pem* uso la orden siguiente:

```
openssl ecparam -name sect163k1 -out stdECparam.pem
```

- *-name*: Nombre de la curva elíptica.



Referencias:

- <http://www.secg.org/SEC2-Ver-1.0.pdf>
- https://wiki.openssl.org/index.php/Command_Line_Elliptic_Curve_Operations

8

Para generar la clave a partir del archivo anterior (*stdECparam.pem*) creado utilizo la siguiente orden:

```
openssl ecparam -in stdECparam.pem -out ange1ECkey.pem -genkey
```

```

✓ angel@juggernaut:8 ➔ openssl ecparam -in stdECparam.pem -out angelECkey.pem -genkey
✓ angel@juggernaut:8 ➔ cat angelECkey.pem
-----BEGIN EC PARAMETERS-----
BgUrgQQAQ==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MFMCQAEEFQND0S/31C/a7KsaokIoR0drU7cXdKAHBgUrgQQAaEuAywABAOZ+egW
TYHBguY8TbbRaKkbh4fBawBZZTy39rcWg+1r/yBdBBzGcuLvqQ==
-----END EC PRIVATE KEY-----
✓ angel@juggernaut:8 ➔

```

Al generar la clave se observa que también se ha incluido la curva elíptica. Para evitar esto agrego *-noout* a la orden anterior.

```
openssl ecparam -in stdECparam.pem -out angelECkey.pem -genkey -noout
```

```

✓ angel@juggernaut:8 ➔ openssl ecparam -in stdECparam.pem -out angelECkey.pem -genkey -noout
✓ angel@juggernaut:8 ➔ cat angelECkey.pem
-----BEGIN EC PRIVATE KEY-----
MFMCQAEEFQMLCe0Wx3qYrU+BBRE9+NZaTuNGcKAHBgUrgQQAaEuAywABAFukZm6
m8V/MSav5pDvAo/wr1KcxQKQiT2PJHhpvna+AGlRS1oj9EJqrg==
-----END EC PRIVATE KEY-----

```

Referencias: https://wiki.openssl.org/index.php/Command_Line_Elliptic_Curve_Operations

9

Del mismo modo que lo hacía en el ejercicio 2 extraigo y cifro la clave privada con AES-128

```
openssl ec -in angelECkey.pem -out angelECpriv.pem -aes128
```

```

✓ angel@juggernaut:9 ➔ openssl ec -in angelECkey.pem -out angelECpriv.pem -aes128
read EC key
writing EC key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
✓ angel@juggernaut:9 ➔ cat angelECpriv.pem
-----BEGIN EC PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,F782C1970BB7D0B4AD49098FE186F2FE

fTrSHy7+nxImYee5rNJRm4BSrlhiHiJmkkYZAW3TLfHCmLXDfhCPLhrn2/kr+n9f
/Mxv0bgQfsYvqnSuf0zeyLmfM8KtU3EUuSjk7pxFcKEeDq/8a5gDZPaObMd8nef/
-----END EC PRIVATE KEY-----

```

Como contraseña utilizo la que se sugiere en el guión de la práctica: 0123456789

Referencias: https://wiki.openssl.org/index.php/Command_Line_Elliptic_Curve_Operations

10

Para extraer la clave pública lo hago del mismo modo que lo hice en el ejercicio 3:

```
openssl ec -in angelECkey.pem -out angelECpub.pem -pubout
```

```
✓ angel@juggernaut:10 → openssl ec -in angelECkey.pem -out angelECpub.pem -pubout
read EC key
writing EC key
✓ angel@juggernaut:10 → cat angelECpub.pem
-----BEGIN PUBLIC KEY-----
MEAwEAYHKoZIZj0CAQYFK4EEAAEDLAAEB9SRmbqbxX8xJq/mk08Cj/CvUpzFApCJ
PY8keGm+dr4AaVFLWiP0Qmqu
-----END PUBLIC KEY-----
✓ angel@juggernaut:10 →
```

Referencias: https://wiki.openssl.org/index.php/Command_Line_Elliptic_Curve_Operations