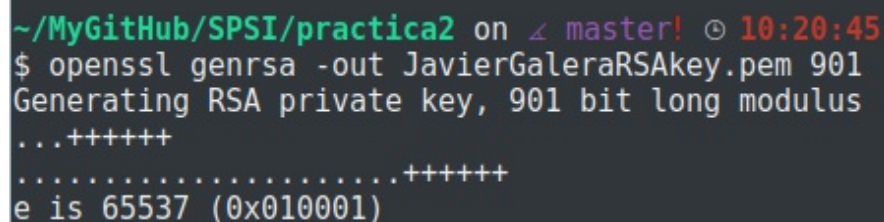


Practica 2 SPSI

Generar una clave RSA (que contiene el par de claves) de 901 bits. El nombre de la clave será RSAkey.pem. La pública no debe estar cifrada ni protegido.

Para generar una clave RSA utilizaremos el comando:

```
openssl genrsa -out <nombre>RSAkey.pem 901
```

A terminal window screenshot showing the command execution. The prompt is '~ /MyGitHub/SPSI/practica2 on master! 10:20:45'. The command entered is '\$ openssl genrsa -out JavierGaleraRSAkey.pem 901'. The output shows 'Generating RSA private key, 901 bit long modulus' followed by progress bars '...+++++' and '.....+++++', and finally 'e is 65537 (0x010001)'.

```
~/MyGitHub/SPSI/practica2 on master! 10:20:45
$ openssl genrsa -out JavierGaleraRSAkey.pem 901
Generating RSA private key, 901 bit long modulus
...+++++
.....+++++
e is 65537 (0x010001)
```

Aunque en la imagen resalte que el resultado de aplicar este comando sea el de generar una clave privada, no hay que causar alarma, la generación de la clave privada va ligada de la mano de la creación de una clave pública. Ambas están en el mismo archivo, para comprobarlo utilizaremos:

```
openssl rsa -in <nombre>RSAkey.pem -text -noout
```

Con el parámetro -text, añade al archivo de las claves los valores de estas en formato texto, y con -noout conseguimos que no haya salida de la operación, por lo que se mostrarán por pantalla:

```
~/MyGitHub/SPSI/practica2 on  $\angle$  master!  $\odot$  10:29:12
$ openssl rsa -in JavierGaleraRSAkey.pem -text -noout
Private-Key: (901 bit)
modulus:
  18:67:5f:40:a6:ac:ee:ef:3f:7e:a3:ec:ed:05:49:
  9e:f4:27:42:de:7f:57:73:72:ad:41:de:6d:06:22:
  7c:43:f5:14:fa:c0:33:a1:42:6f:6e:5a:f8:46:6b:
  96:43:38:38:51:b3:85:0f:aa:3b:11:a8:61:8b:25:
  76:58:65:4b:7c:84:da:50:e4:97:0a:97:45:7b:48:
  a7:5d:21:38:79:32:fb:5a:c8:d0:12:a9:35:0f:26:
  ed:1f:af:d8:df:75:d8:29:04:b9:7e:b8:e4:8c:2f:
  43:48:a5:88:54:6b:06:13
publicExponent: 65537 (0x10001)
privateExponent:
  0b:16:b5:96:77:e5:ee:6f:f1:e7:06:6f:7a:c7:c3:
  4b:21:df:d1:27:ae:af:3a:fb:29:b4:db:6f:a8:b0:
  f4:cc:20:49:d7:22:8c:93:42:cf:c5:52:3a:ac:2c:
  e8:cb:44:79:7a:ce:5a:b0:e2:86:12:9b:f9:ee:52:
  d4:aa:e4:02:f4:2d:5f:04:86:0e:06:e7:06:40:13:
  41:7b:bb:fc:42:3c:0c:d8:75:f7:e9:dd:b9:36:cf:
  d4:7a:b5:ee:1d:8f:0c:b8:87:b2:df:bc:9d:7b:79:
  c8:d2:f6:84:0b:7e:3b:59
prime1:
  07:d4:0e:dd:38:e8:98:22:e0:75:35:b5:82:3a:79:
  2e:52:42:5d:6e:c3:fc:58:c3:36:34:81:37:c9:a6:
```

```
prime1:
  07:d4:0e:dd:38:e8:98:22:e0:75:35:b5:82:3a:79:
  2e:52:42:5d:6e:c3:fc:58:c3:36:34:81:37:c9:a6:
  12:b7:b3:38:3b:d9:bb:fe:02:65:a5:20:ca:b1:ab:
  64:ce:af:65:a1:4d:13:c4:d4:ed:0b:bd
prime2:
  03:1e:0b:5b:c5:8c:d1:61:84:94:b1:1d:b3:4d:93:
  67:68:f3:e8:b9:27:1d:27:74:ec:da:e4:94:96:e8:
  e1:aa:b3:af:ce:2d:de:aa:68:95:ef:6a:16:64:d8:
  a3:11:d4:70:1f:29:2e:2b:d8:1f:0e:0f
exponent1:
  02:9a:af:1f:35:11:5b:9b:2e:a4:8f:6b:84:99:43:
  3a:ff:ae:88:3a:93:04:5d:db:03:2b:50:59:68:c1:
  d0:b4:3f:f0:d3:28:49:fb:e4:7c:40:04:ba:a9:84:
  65:12:c5:12:c0:28:4e:0d:4a:f0:58:d9
exponent2:
  00:cb:3f:13:51:5d:ca:d9:ae:3d:7c:5a:57:17:13:
  ca:8b:75:4b:39:14:98:1d:3b:6f:33:54:71:ef:79:
  42:38:0a:69:64:f6:53:b6:4c:ca:9a:de:06:f4:0a:
  24:85:de:f7:eb:47:be:a5:95:f6:5a:cb
coefficient:
  07:0a:2a:d1:aa:b0:04:af:24:89:36:af:f4:7f:7a:
  e0:59:c5:c5:34:9d:fb:8a:62:fe:71:91:35:0e:68:
  a2:61:32:1e:b9:43:bb:48:b1:3d:ec:42:71:cd:fe:
  6a:3f:4a:fa:1c:4c:4b:56:41:ad:e5:c6
```

Extraer la clave privada contenida en el archivo RSAkey.pem a otro archivo que tenga por nombre

RSApriv.pem. Este archivo deberá estar protegido por contraseña cifrándolo con AES-128. Mostrar sus valores.

Para extraer la parte privada de nuestra clave utilizamos:

```
openssl rsa -in <nombre>RSAkey.pem -out <nombre>RSApriv.pem -des
```

NOTA: Como contraseña utilizamos por recomendación del profesor: 0123456789

```
~/MyGitHub/SPSI/practica2 on ↵ master! ⌚ 18:52:57
$ openssl rsa -in JavierGaleraRSAkey.pem -out JavierGaleraRSApriv.pem -des
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Ahora mostramos el resultado de nuestra clave privada:

```
openssl rsa -in <nombre>RSApriv.pem -text -noout
```

```
~/MyGitHub/SPSI/practica2 on ↵ master! ⌚ 18:55:36
$ cat JavierGaleraRSApriv.pem
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-CBC,480E6E341C93680D

ke/9+W5aaHeXfmuzh0rCnMW/80DYFEIKRDb6K1lGN9/MYPYhk2lQl20sWSfktbnJ
CQHwIYZn5xSGVvFRPrTQz3RcSJUEr4df21I05jy/3oVvSVX1x1wX8Qfw3gt3v0NY
yCdRQV5XnJFCj fRlogUymBjZ8SFZRF628kxLZNzVLEFZzHN0ABvu4JJb/F6s7ude
9Q1ujT2oMjENYPnR4YN0W0X/c1fPcbM7ZDsAcvn7zbKmdRHWsgA1qppdIUCkmIQc
esEpoiZwM7U4bq4+Z+ZCLNn+YnHf80KcebiBx8pkDHt0+sBYbjF3yUxqoY9+856D
oM1UohdmU78bN30poYj6U3fhTkRVmBYqhSagCit85+Ql4s1S4ZTW7Cx60TQkuZoa
WdJAzF0J50ontEGtAEpfbhKLSXSaeRsK1F/iJG1fSiyn96K4w28mrssZCmI3Q8I6
zp288zF4ouKERXnDuB+9G4nPPbrKMv5AbSowDuAf+3wmCF3W96NsoKv4PgytaFm7
MtGdjeAWozsrz3vbHWKra7DGpw9AJl9ny01KeFf63bZo2WBnyCpb9n320KJv6uSU
0GMTnRCGXDmtauTbD8TYS2YjbNhjcZqW0UAAyMvUlcLCy4iZ5M6r3hyFiiR9IImX
zKEAeU+nDimSWnbTFqsoRUYHcfcMotXCg/fIzqPDAsAcbaMf6KMSVty4ED7lzt3W
Ik1VpYWVsNCrdEE0v1B9tw==
-----END RSA PRIVATE KEY-----
```

Comprobamos en el apartado "DEK-info" que el archivo ha sido cifrado por DES en modo "CBC"

Extraer en RSApub.pem la clave pública contenida en el archivo RSAkey.pem no debe estar cifrado ni protegido. Mostrar sus valores.

Para extraer la parte pública del archivo utilizamos:

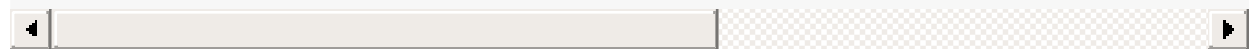
```
openssl rsa -in <nombre>RSAkey.pem -pubout -out <nombre>RSApub.pem
```

```
~/MyGitHub/SPSI/practica2 on  $\angle$  master!  $\odot$  10:54:07  
$ openssl rsa -in JavierGaleraRSAkey.pem -pubout -out JavierGaleraRSApub.pem  
writing RSA key
```

Ahora mostraremos el resultado de nuestra clave pública:

```
openssl rsa -in <nombre>RSApub.pem -pubin -text -noout
```

NOTA: por defecto a la hora de extraer, openssl entiende que las claves son privadas,



```
~/MyGitHub/SPSI/practica2 on  $\angle$  master!  $\odot$  10:57:36  
$ openssl rsa -in JavierGaleraRSApub.pem -pubin -text -noout  
Public-Key: (901 bit)  
Modulus:  
  1a:49:ed:26:ef:ce:2b:4a:9f:2d:50:bd:05:de:cf:  
  c5:7c:e0:79:b2:73:29:d6:bb:9d:f7:16:51:17:1d:  
  ae:6d:a0:bc:8e:01:33:66:0f:44:dd:5d:44:ac:98:  
  1e:6f:48:5c:aa:84:c4:8a:c3:10:f2:59:ec:67:7a:  
  88:c9:7b:f4:8e:a0:91:c9:c5:7d:c9:4d:a5:4d:d0:  
  6b:e1:a1:d0:67:38:0e:6d:e4:f0:f1:42:aa:a8:46:  
  43:52:64:91:b5:26:0a:3b:c5:75:b0:b6:91:6a:69:  
  44:14:93:8e:0f:93:f1:b1  
Exponent: 65537 (0x10001)
```

Reutilizar el archivo binario input.bin de 1024 bits, todos ellos con valor a 0, de la practica anterior. Intentar cifrar input.bin con la clave pública y explicar el mensaje de error obtenido.

Para intentar cifrar el archivo input.bin utilizaremos el comando:

```
openssl rsautl -encrypt -inkey <nombre>RSAPub.pem -pubin -in input.bin -out resultado
```

```
~/MyGitHub/SPSI/practica2 on / master! @ 10:26:57
$ openssl rsautl -encrypt -inkey JavierGaleraRSAPub.pem -pubin -in input1.bin -out resultado.ssl
RSA operation error
140522747445696:error:0406D06E:rsa routines:RSA_padding_add_PKCS1_type_2:data too large for key size:../crypto/rsa/rsa_pk1.c:125:
```

Nos muestra un error:

```
RSA operation error
140522747445696:error:0406D06E:rsa routines:RSA_padding_add_PKCS1_type_2:data too la
```

"KeyPubli: 901 bits" ; "Input.bin: 1024 bits" ;

El parámetro "rsautl" no cifra ningún dato de entrada que sea más grande que el tamaño de la clave RSA. $901 < 1024$, por lo tanto no puede cifrar el archivo.

Diseñar un cifrado híbrido, con RSA como criptosistema asimétrico. El modo de proceder será el siguiente:

1. El emisor debe seleccionar un sistema **simétrico** con su correspondiente modo de op
2. El emisor generará un archivo de texto, llamado por ejemplo "**sessionkey**" con dos
3. El archivo "**sessionkey**" se cifrará con la clave pública del receptor.
4. El mensaje se cifrará utilizando el criptosistema **simétrico**, la clave se generará
-pass file:sessionkey.s

- Apartado 1: Escogeremos el modo ECB-256.
- Apartado 2: Creación del archivo.

Para nuestra clave utilizaremos el comando: openssl rand -hex 32

De esta forma generará en la primera línea los 64 bits que necesitamos.

```
~/MyGitHub/SPSI on ◀ master! @ 11:36:16
$ openssl rand -hex 32
6e1996dcdcf96b8222615b61d6a6fa59fc17942d8e28ec5b76751fdf687845e9
```

En la segunda línea contendrá: -bf-ecb

Comprobamos el estado del archivo final:

```
~/MyGitHub/SPSI/practica2 on ◀ master! @ 11:44:37
$ cat sessionkey
6e1996dcdcf96b8222615b61d6a6fa59fc17942d8e28ec5b76751fdf687845e9
-bf-ecb
```

- Apartado 3: Para cifrar el archivo "sessionkey" con la clave pública del receptor, primero necesitamos la clave pública de este:

Para ello seguiremos los puntos anteriores para ellos "" tomará el valor de "Receptor".

```
~/MyGitHub/SPSI/practica2 on ◀ master! @ 1:28:03
$ openssl genrsa -out ReceptorRSAkey.pem 901
Generating RSA private key, 901 bit long modulus
.....++++++
.+++++
e is 65537 (0x010001)

~/MyGitHub/SPSI/practica2 on ◀ master! @ 1:28:50
$ openssl rsa -in ReceptorRSAkey.pem -pubout -out ReceptorRSAPub.pem
writing RSA key
```

Utilizamos el siguiente comando:

```
openssl rsautl -encrypt -in sessionkey -out sessionkey.enc -inkey ReceptorRSAPub.pem
```

```
~/MyGitHub/SPSI/practica2 on ◀ master! @ 1:29:57
$ openssl rsautl -encrypt -in sessionkey -out sessionkey.enc -inkey ReceptorRSAPub.pem -pubin
```

- Apartado 4: Por último una vez tenemos el archivo sessionkey.enc cifraremos con ECB-256 y utilizando como clave el archivo "sessionkey.enc" el mensaje del emisor, el mensaje estará en el archivo "mensaje.txt".

Para ello utilizamos el comando:

```
openssl enc -aes-256-ecb -pass file:sessionkey.enc -in mensaje.txt -out
mensaje.enc
```

```
~/MyGitHub/SPSI/practica2 on master! 1:36:42
$ openssl enc -aes-256-ecb -pass file:sessionkey.enc -in mensaje.txt -out mensaje.enc

~/MyGitHub/SPSI/practica2 on master! 1:43:03
$ cat mensaje.enc
Salted__0;0cU0m00p60b3000
Y0[00000x0{00
00001j0>00c0%
```

Utilizando el criptosistema híbrido diseñado, se debe cifrar el archivo input.bin con la propia clave pública. Y a continuación, descifrarlo con la clave privada y compararlo con el resultado original.

Ciframos input.bin:

```
openssl enc -aes-256-ecb -pass file:sessionkey -in input1.bin
-out encryptInput.bin
```

Con esto utilizamos la clave de sesión generada anteriormente(sessionkey).

El archivo cifrado quedaría tal que:

```
~/MyGitHub/SPSI/practica2 on master! 2:12:47
$ xxd encryptInput.bin
00000000: 5361 6c74 6564 5f5f e840 2ab9 03bf 8c57  Salted_@*...W
00000010: 871c ad4f 9f0f 772f 27a2 f560 c2dd fbe9  ...0..w/'...
00000020: 871c ad4f 9f0f 772f 27a2 f560 c2dd fbe9  ...0..w/'...
00000030: 871c ad4f 9f0f 772f 27a2 f560 c2dd fbe9  ...0..w/'...
00000040: 871c ad4f 9f0f 772f 27a2 f560 c2dd fbe9  ...0..w/'...
00000050: 871c ad4f 9f0f 772f 27a2 f560 c2dd fbe9  ...0..w/'...
00000060: 871c ad4f 9f0f 772f 27a2 f560 c2dd fbe9  ...0..w/'...
00000070: 871c ad4f 9f0f 772f 27a2 f560 c2dd fbe9  ...0..w/'...
00000080: 871c ad4f 9f0f 772f 27a2 f560 c2dd fbe9  ...0..w/'...
00000090: ae5b 6566 4edc 80dc 559d 007a 648a 3abf  .[efN...U..zd..
```

Ya tenemos cifrado el archivo, ahora vamos a descifrarlo. Al utilizar el cifrado híbrido le daríamos al receptor el sessionkey cifrado y en mensaje cifrado. El receptor debe descifrar con su clave privada el sessionkey y una vez hecho esto aplicar el descifrado del criptosistema simétrico que se encuentra en el archivo "sessionkey".

(Previamente hemos cifrado sessionkey con nuestra clave pública, pues el emisor y el receptor en este caso soy yo mismo).

```
~/MyGitHub/SPSI/practica2 on   master!   2:18:39
$ cat sessionkey
6e1996dcdcf96b8222615b61d6a6fa59fc17942d8e28ec5b76751fdf687845e9
-ecb-256-

~/MyGitHub/SPSI/practica2 on   master!   2:18:58
$ openssl rsautl -encrypt -in sessionkey -out sessionkey.enc -inkey JavierGaleraRSAPub.pem -pubin
```

Utilizamos:

```
openssl rsautl -decrypt -in sessionkey.enc -out sessionkey.denc
-inkey JavierGaleraRSAPriv.pem
```

Mostramos el resultado:

```
~/MyGitHub/SPSI/practica2 on   master!   2:22:45
$ openssl rsautl -decrypt -in sessionkey.enc -out sessionkey.denc -inkey JavierGaleraRSAPriv.pem

~/MyGitHub/SPSI/practica2 on   master!   2:22:54
$ cat sessionkey.denc
6e1996dcdcf96b8222615b61d6a6fa59fc17942d8e28ec5b76751fdf687845e9
-ecb-256-
```

Podemos comprobar que el resultado era el esperado, una vez tenemos el archivo descifrado de forma correcta, usamos el cifrado simétrico para descifrar el archivo encryptInput.bin, para ello utilizamos:

```
openssl enc -aes-256-ecb -pass file:sessionkey.denc
-in encryptInput.bin -out decryptInput.bin
```

```
~/MyGitHub/SPSI/practica2 on   master!   2:28:02
$ openssl enc -aes-256-ecb -d -pass file:sessionkey.denc -in encryptInput.bin -out decryptInput.bin

~/MyGitHub/SPSI/practica2 on   master!   2:28:08
$ xxd decryptInput.bin
00000000: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000040: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000050: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
```

El sistema funciona, hemos conseguido descifrar el mensaje.

#Generar un archivo stdECparam.pem que contenga los parámetros públicos de una de las curvas elípticas contenidas en las transparencias de teoría. Si no se logra localizarla realizar el resto de la práctica con una curva cualquiera a vuestra elección de las disponibles en OpenSSL. Mostrar los valores.

Para ver el listado de curvas elípticas disponibles en OpenSSL usaremos el comando **ecparam**.

Con este comando también podremos manipular y generar dichas curvas.

Tipos(algunas) de curvas predefinidas en OpenSSL:

```
~/MyGitHub/SPSI/practica2 on ◀ master! @ 17:56:37
$ openssl ecparam -list_curves
secp112r1 : SECG/WTLS curve over a 112 bit prime field
secp112r2 : SECG curve over a 112 bit prime field
secp128r1 : SECG curve over a 128 bit prime field
secp128r2 : SECG curve over a 128 bit prime field
secp160k1 : SECG curve over a 160 bit prime field
secp160r1 : SECG curve over a 160 bit prime field
secp160r2 : SECG/WTLS curve over a 160 bit prime field
secp192k1 : SECG curve over a 192 bit prime field
secp224k1 : SECG curve over a 224 bit prime field
secp224r1 : NIST/SECG curve over a 224 bit prime field
secp256k1 : SECG curve over a 256 bit prime field
secp384r1 : NIST/SECG curve over a 384 bit prime field
secp521r1 : NIST/SECG curve over a 521 bit prime field
prime192v1 : NIST/X9.62/SECG curve over a 192 bit prime field
prime192v2 : X9.62 curve over a 192 bit prime field
prime192v3 : X9.62 curve over a 192 bit prime field
prime239v1 : X9.62 curve over a 239 bit prime field
prime239v2 : X9.62 curve over a 239 bit prime field
prime239v3 : X9.62 curve over a 239 bit prime field
prime256v1 : X9.62/SECG curve over a 256 bit prime field
sect113r1 : SECG curve over a 113 bit binary field
sect113r2 : SECG curve over a 113 bit binary field
sect131r1 : SECG/WTLS curve over a 131 bit binary field
```

En mi caso escogeremos ***prime256v1***.

Crearemos el archivo mediante:

```
openssl ecparam -out stdECparam.pem -name prime256v1
```

El archivo creado:

```
~/MyGitHub/SPSI/practica2 on ◀ master! @ 18:03:02
$ openssl ecparam -out stdECparam.pem -name prime256v1

~/MyGitHub/SPSI/practica2 on ◀ master! @ 18:04:01
$ ls
decryptInput.bin  ej2  ej4  ej6  encryptInput.bin  JavierGaleraRSAkey.pem  JavierGaleraRSApub.pem  mensaje.txt  practica2.pdf  ReceptorRSApriv.pem  resultado.ssl  sessionkey.denc  stdECparam.pem
ej1              ej3  ej5  ej7  input1.bin        JavierGaleraRSApriv.pem  mensaje.enc           practica2.md  ReceptorRSAkey.pem  ReceptorRSApub.pem  sessionkey      sessionkey.enc

~/MyGitHub/SPSI/practica2 on ◀ master! @ 18:04:03
$ cat stdECparam.pem
-----BEGIN EC PARAMETERS-----
BgqhkJOPQMBw==
-----END EC PARAMETERS-----

~/MyGitHub/SPSI/practica2 on ◀ master! @ 18:04:09
$
```

información del archivo:



Comprobamos que hay 2 líneas: ***ASN1 OID***__ y __***NIST CURVE***.

1. OID es un nombre usado para identificar al objeto, ASN1 (Abstract Syntax Notation One) es

un protocolo de nivel de presentación del modelo OSI. ASN1 es una norma para representar datos de forma abstracta, haciendo posible que sean validos independientemente de la máquina empleada y sus formas de representación internas.

2. NIST(The National Institute of Standards and Technology) CURVE tiene el nombre de la curva en el standard, en nuestro caso P-256.

#Generar una clave para los parámetros anteriores. La clave se almacenará en ECkey.pem y no es necesario protegerla por contraseña.

Para ello, y a partir del parámetro anterior **ecparam**, escribimos en la terminal:

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

Añadimos **-genkey** para indicarle que genere la clave privada asociada a la curva elíptica con los parámetros introducidos.

Así el archivo queda tal que:

```
~/MyGitHub/SPSI/practica2 on ◀ master! @ 18:22:07
$ openssl ecparam -in stdECparam.pem -out JavierGaleraECkey.pem -genkey

~/MyGitHub/SPSI/practica2 on ◀ master! @ 18:22:39
$ cat JavierGaleraECkey.pem
-----BEGIN EC PARAMETERS-----
BggqhkJOPQMBBw==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIUTBiCedXQJiMi8SVznHIlv85Fneh8Vwe7yehUfrKn7oAoGCCqGSM49
AwEHoUQDQgAEzVWo1NBbr0G/HjzAeY/9LmYbbLQmwpjA9lqKw8/hAbR46VND4LE8
k+5URUVtxdkbddBq3gBKhtluZw2xWhl0tw==
-----END EC PRIVATE KEY-----
```

"Extraer" la clave privada contenida en el archivo ECkey.pem a otro archivo que tenga por nombre ECpriv.pem. Este archivo deberá estar protegido por contraseña. Mostrar los valores

Como en los ejercicios anteriores con RSA extraeremos y protegeremos con contraseña utilizaremos el comando:

```
openssl ec -in nombreEKey.pem -des -out nombreECpriv.pem
```

NOTA: Como contraseña por recomendación se ha utilizado: 0123456789

Podemos comprobar ahora el archivo:

```
~/MyGitHub/SPSI/practica2 on  $\angle$  master! @ 18:40:36
$ openssl ec -in JavierGaleraEKey.pem -des -out JavierGaleraECpriv.pem
read EC key
writing EC key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

~/MyGitHub/SPSI/practica2 on  $\angle$  master! @ 18:41:38
$ cat JavierGaleraECpriv.pem
-----BEGIN EC PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-CBC,7F8E5E8373C589D6

uS3903tubp93ok/LBTAS2ipqujmSS2qCiCNrdH0zAjqkl7U9MLVrNFyf0n8jLR+V
RCMRFowu42tTki81z1ULTBwn70W59tWd2Dg2q3/e3H5Ibg5cqVZnEnS2WehNC/K/
LsI7m5G6a2vLWj4Bx21o1/igDHwH1Hv6C10lul2VCu0=
-----END EC PRIVATE KEY-----
```

En la información comprobamos que se ha cifrado con "DES" en modo "CBC" el resto es la información necesaria para que, junto a la contraseña, se descifre el archivo.

Extraer en ECpub.pem la clave pública contenida en el archivo EKey.pem. Como antes, no debe de estar cifrada ni protegida. Mostrar sus valores.

Para ello utilizamos el comando:

```
openssl ec -in nombreEKey.pem -pubout -out nombreECpub.pem
```

Comprobamos ahora el archivo:

```
~/MyGitHub/SPSI/practica2 on   master!   18:44:18
$ openssl ec -in JavierGaleraECkey.pem -pubout -out JavierGaleraECpub.pem
read EC key
writing EC key
```

```
~/MyGitHub/SPSI/practica2 on   master!   18:47:23
$ cat JavierGaleraECpub.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEzVWo1NBbr0G/HjzAeY/9LmYbbLQm
wpjA9lqKw8/hAbR46VND4LE8k+5URUVtxdkbBq3gBKhtluZw2xWhl tw==
-----END PUBLIC KEY-----
```