

Práctica 2

CRIPTOSISTEMAS ASIMÉTRICOS



Manuel Ruiz Maldonado

Índice

Ejercicio 1.....	3
Ejercicio 2.....	3
Ejercicio 3.....	4
Ejercicio 4.....	5
Ejercicio 5.....	5
Ejercicio 6.....	7
Ejercicio 7.....	9
Ejercicio 8.....	9
Ejercicio 9.....	10
Ejercicio 10.....	10
Conclusión.....	11
Bibliografía.....	13

En esta práctica voy a seguir con el uso de OpenSSL, esta vez para estudiar el comportamiento de los criptosistemas asimétricos, más concretamente, RSA y criptosistemas basados en curvas elípticas.

Ejercicio 1

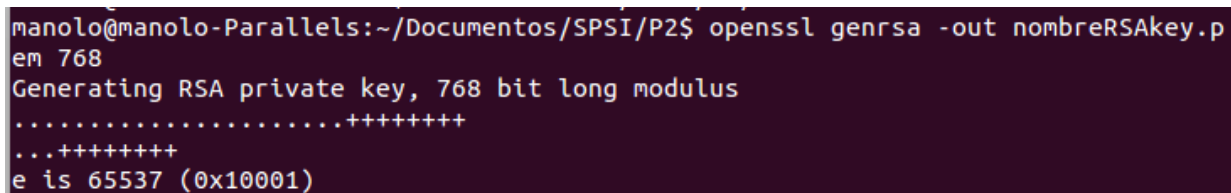
Generad, cada uno de vosotros, una clave RSA (que contiene el par de claves) de 768 bits. Para referirnos a ella supondré que se llama nombreRSAkey.pem. Esta clave no es necesario que esté protegida por contraseña.

En este ejercicio he utilizado la documentación de OpenSSL [1] relativa a la generación de claves RSA para conocer la sintaxis que debía utilizar. El comando usado ha sido:

```
$$ openssl genrsa -out nombreRSAkey.pem 768
```

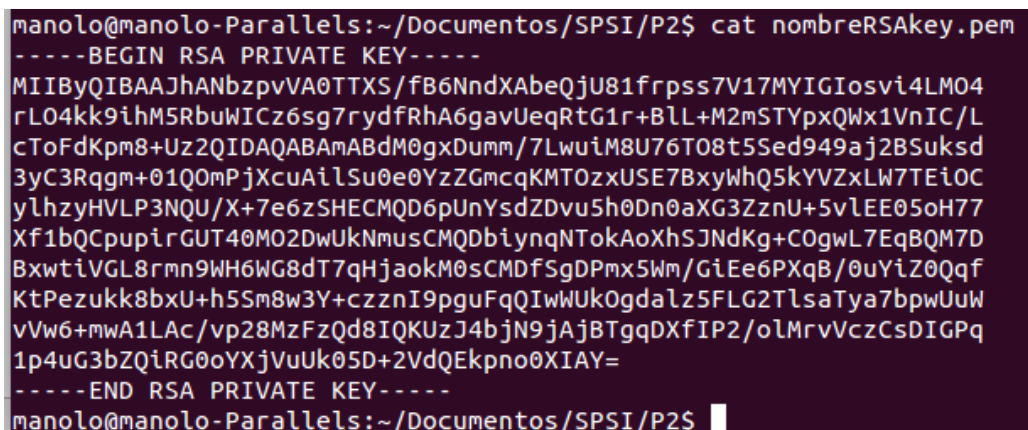
Con `openssl genrsa` se genera una clave para el criptosistema RSA. Con `-out` se indica el nombre del fichero en el que se quiere almacenar la clave. Finalmente se indica el tamaño en bits de la clave.

El resultado obtenido se puede ver en las Imágenes 1 y 2.



```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ openssl genrsa -out nombreRSAkey.p
em 768
Generating RSA private key, 768 bit long modulus
.....++++++
...++++++
e is 65537 (0x10001)
```

Imagen 1: Generación de clave RSA usando OpenSSL.



```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat nombreRSAkey.pem
-----BEGIN RSA PRIVATE KEY-----
MIIBYQIBAAJhANbZpvVA0TTXS/fB6NndXAbeQjU81frpss7V17MYIGIosvi4LM04
rLO4kk9ihM5RbuWICz6sg7rydfRhA6gavUeqRtG1r+BLL+M2mSTYpxQWx1VnIC/L
cToFdKpm8+Uz2QIDAQABAMABdM0gxDumm/7LwuiM8U76T08t5Sed949aj2BSuksd
3yC3Rqgm+01Q0mPjXcuAilSu0e0YzZGmcqKMT0zxUSE7BxyWhQ5kYVZxLW7TEiOC
ylhzyHVL3P3NQU/X+7e6zSHECMQD6pUnYsdZDvu5h0Dn0aXG3ZznU+5vLEE05oH77
Xf1bQCpupirGUT40M02DwUkNmuscMQDblyngNTokAoXhSJNdKg+COgwL7EqBQM7D
BxwtiVGL8rmn9WH6WG8dT7qHjaokM0sCMDfSgDPmx5Wm/GiEe6PXqB/0uYiZ0Qqf
KtPezukk8bxU+h5Sm8w3Y+czznI9pguFqQIwwUk0gdalZ5FLG2TlsaTya7bpwUuW
vVw6+mWA1LAc/vp28MzFzQd8IQKUzJ4bjN9jAjbTgqDXfIP2/oLMrvVczCsDIGPq
1p4uG3bZQiRG0oYXjVvuUk05D+2VdQEkpno0XIAY=
-----END RSA PRIVATE KEY-----
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

Imagen 2: Salida por pantalla del contenido de la clave en modo texto.

Ejercicio 2

“Extraed” la clave privada contenida en el archivo nombreRSAkey.pem a otro archivo que tenga por nombre nombreRSAPriv.pem. Este archivo deberá estar protegido por contraseña cifrándolo con AES-128. Mostrad sus valores.

En este caso se utiliza el comando `openssl rsa` que sirve para manipular claves. La sintaxis la he extraído de [2]. La orden usada ha sido:

```
$$ openssl rsa -in nombreRSAkey.pem -out nombreRSAPriv.pem -aes128
```

La opción *-in* permite indicar el archivo que contiene la clave RSA. La opción *-out* permite indicar el archivo de salida en el que guardar la clave privada. Con *-aes128* se indica el método con el que cifrar la clave privada.

Hay que saber que el comando *openssl rsa* trabaja por defecto con archivos tipo *.pem*. En caso de usar otro tipo de archivos es necesario indicarlo.

Como se puede ver en la Imagen 3, al introducir la orden se ha requerido una contraseña. Desde teclado he introducido la que se sugería en el enunciado: “0123456789”.

```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ openssl rsa -in nombreRSAkey.pem -out
nombreRSPriv.pem -aes128
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Imagen 3: Introducción de contraseña para proteger la clave privada.

El resultado se puede ver en la Imagen 4. A simple vista se puede apreciar que el contenido de la clave está encriptado y además te dice el método que ha sido utilizado para realizar dicho cifrado.

```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat nombreRSPriv.pem
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,AD3759BC214D3EE069E6839ECC89EBEA

kNWhfp1lDCudulR9xXRDkgESwzN3xk3In5HNvMS0CC2eQrNjU+5kqjuuWHypz1Gf
vpu2oqpQ2lC/5adEjUoy+Xb0tYqGrmqIy0WfbFvS6tGE5C/lm3Trvw0ACsRWVIF
+6KdEYlwcIfLPrrA7LKIPpq0CgUz/J7dDPVeFEGDgL/ko385HYP8pEW0AvRmvg/b
VCp3YCjx1fDPy68ZG/pJVAhmVRnkKNFqyIfSxMzmzgyGn9203Cid3tW2ZIEJ3sSd
73XKXQ82QKAUxfsBt3GwoRxQEghmIY6GUC50xCsLaFL7lwKs7VapKlv5be2mqDRp
XvbE++batFKpxvqCdezEljrg1Hyer2j/2H/0agFMUZpk2/Cx3sJpUbAbDaSmGjQK
iZncvbuJwmDPx1P4JMCxHfiWLhJcsAmQ1n38BRm8w02vKVXL0f3I6c0BYJOFAkSE
9lkseLdjb96x0EbV/xjD35gzFqawrIhyRSAHF7v/+ZQh5cQmJQEi4UWBSrae4uiv
REZceawws0fxl+ZrBNcB0rVLE/Y074adoffbe9s2IDLEohE3mrIorlpeWvwekI4
UG8wzbZdIBf7QqK9MhQ51lT600HN7oYbaJDUQq1VmhQ=
-----END RSA PRIVATE KEY-----
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

Imagen 4: Valor cifrado de la clave privada.

Ejercicio 3

Extraed en *nombreRSAPub.pem* la clave pública contenida en el archivo *nombreRSAkey.pem*. Evidentemente *nombreRSAPub.pem* no debe estar cifrado ni protegido. Mostrad sus valores.

Para realizar este ejercicio voy a utilizar la orden del ejercicio anterior añadiendo una opción nueva.

```
$$ openssl rsa -in nombreRSAkey.pem -out nombreRSAPub.pem -pubout
```

En este caso no añado *-aes128* porque no se requiere que la clave pública esté cifrada. La opción *-pubout* hace que se extraiga de la clave RSA la parte pública, ya que por defecto, como se veía en el Ejercicio 1 - Imagen 2, se mostraba la parte privada de la clave como salida.

Se puede ver el resultado obtenido en la Imagen 5.

```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat nombreRSApub.pem
-----BEGIN PUBLIC KEY-----
MHwwDQYJKoZIhvcNAQEBBQADAwAwAJhANbZpvVA0TTXS/fB6NndXAbeQjU81frp
ss7V17MYIGIosvi4LM04rL04kk9ihM5RbuWICz6sg7rydfRhA6gavUeqRtG1r+B1
L+M2mSTYpxQWx1VnIC/LcToFdKpm8+Uz2QIDAQAB
-----END PUBLIC KEY-----
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

Imagen 5: Extracción de clave pública RSA.

Ejercicio 4

Reutilizaremos el archivo binario **input.bin** de 1024 bits, todos ellos con valor 0, de la práctica anterior. Intentad cifrar **input.bin** con vuestras claves pública. Explicad el resultado.

Para realizar este ejercicio voy a utilizar la sintaxis descrita en [3] que habla sobre las utilidades de RSA. El comando usado es el siguiente:

```
$$ openssl rsautl -in input.bin -out input-cifrado.bin -inkey nombreRSApub.pem -pubin -encrypt
```

Con *openssl rsautl* estamos indicando que vamos a realizar algún proceso con el criptosistema RSA. Las opciones *-in* y *-out* indican el fichero de entrada y de salida que se quiere cifrar. Con *-inkey* se indica el fichero del que se cogerá la clave para cifrar. Con *-pubin* se indica que se utilizará una clave RSA pública para encriptar. Y con la opción *-encrypt* se indica que el proceso que se quiere realizar es encriptación, el cual requiere el uso de una clave pública RSA.

El resultado obtenido se puede observar en la Imagen 6.

```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ openssl rsautl -in input.bin -out
input-cifrado.bin -inkey nombreRSApub.pem -pubin -encrypt
RSA operation error
139953188632224:error:0406D06E:rsa routines:RSA_padding_add_PKCS1_type_2:data to
o large for key size:rsa_pk1.c:151:
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

Imagen 6: Error al intentar cifrar usando RSA.

Como se puede deducir, se ha producido un error en el proceso de cifrado.

Según se puede leer en ese mensaje de error, esto ha ocurrido porque el tamaño del mensaje que se quiere cifrar es demasiado grande para la clave utilizada, por lo que el proceso de cifrado no se puede completar. He realizado varias pruebas en las que utilizando tamaños de mensajes menores que los de la clave sí permite realizar el cifrado, luego se puede extraer que para realizar ese proceso es necesario que el mensaje tenga un tamaño limitado. Además, en [4] hay varios apartados en los que se define dicho límite.

Ejercicio 5

Diseñad 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 operación.

El sistema que voy a elegir en este ejercicio es AES-256-CBC porque vi en la práctica anterior que era un buen método de cifrado simétrico.

2. El emisor generará un archivo de texto, llamado por ejemplo *sessionkey* con dos líneas. La primera línea contendrá una cadena aleatoria hexadecimal cuya longitud sea la requerida por la clave. OpenSSL permite generar cadenas aleatorias con el comando `openssl rand`. La segunda línea contendrá la información del criptosistema simétrico seleccionado. Por ejemplo, si hemos decidido emplear el algoritmo Blowfish en modo ECB, la segunda línea deberá contener `-bf-ecb`.

Lo primero que he hecho ha sido generar el fichero *sessionkey*. Para ello he usado el comando `openssl rand` cuya sintaxis viene explicada en [5]. La orden usada ha sido:

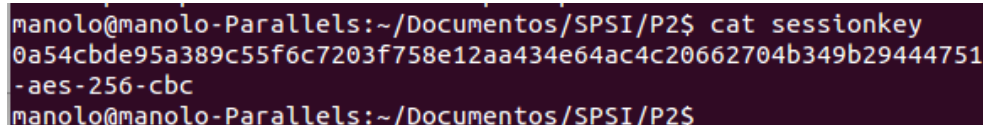
```
$$ openssl rand -out sessionkey -hex 32
```

Las opciones usadas han sido:

`-hex` indica que el resultado se pueda leer como una cadena hexadecimal.

`32` indica el número de bytes que debe tener la clave. Como para AES-256 se necesita una clave de 256 bits, eso corresponde a 32 bytes.

Después he usado un editor de texto para añadir el contenido del criptosistema simétrico que he decidido utilizar para este criptosistema híbrido. El contenido del fichero *sessionkey* se puede ver en la Imagen 7.



```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat sessionkey
0a54cbde95a389c55f6c7203f758e12aa434e64ac4c20662704b349b29444751
-aes-256-cbc
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

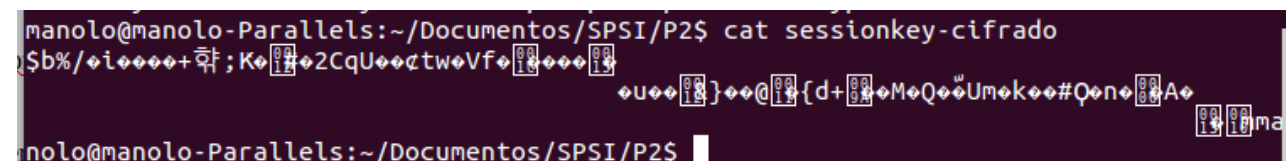
Imagen 7: Contenido del fichero *sessionkey*.

3. El archivo *sessionkey* se cifrará con la clave pública del receptor.

Para cifrar *sessionkey* voy a utilizar la orden del Ejercicio 4.

```
$$ openssl rsautl -in sessionkey -out sessionkey-cifrado -inkey nombreRSAPub.pem -pubin
-encrypt
```

Así se ha obtenido el fichero *sessionkey-cifrado* (ver Imagen 8). En este caso sí se puede cifrar este fichero usando RSA porque el tamaño del fichero es menor que el de la clave.



```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat sessionkey-cifrado
$B%/i+++++;K2CqU+ctwVf+
+u++}++@{d++M+Q+Umk++#Qon+A+
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

Imagen 8: Fichero *sessionkey-cifrado*.

4. El mensaje se cifrará utilizando el criptosistema simétrico, la clave se generará a partir del archivo anterior mediante la opción `-pass file:sessionkey`.

El mensaje que quiero cifrar va a ser el contenido del fichero *input.bin*. Para ello voy a utilizar los comandos aprendidos en la práctica anterior. El comando a usar es el siguiente:

```
$$ openssl enc -aes-256-cbc -in input.bin -out input-aes256cbc.bin -pass file:sessionkey
```

Así he realizado el cifrado del fichero *input.bin* utilizando el sistema AES-256 en modo CBC (Imagen 9). La contraseña utilizada para cifrar se coge automáticamente del fichero *sessionkey* según se indica en la documentación en [6]. Además, no le tengo que indicar el vector de inicialización ya que solo es necesario hacerlo cuando la clave se le pasa con la opción *-K*. En cualquier otro caso, como se la estoy pasando en esta ocasión, el vector de inicialización se genera automáticamente a partir de la clave.

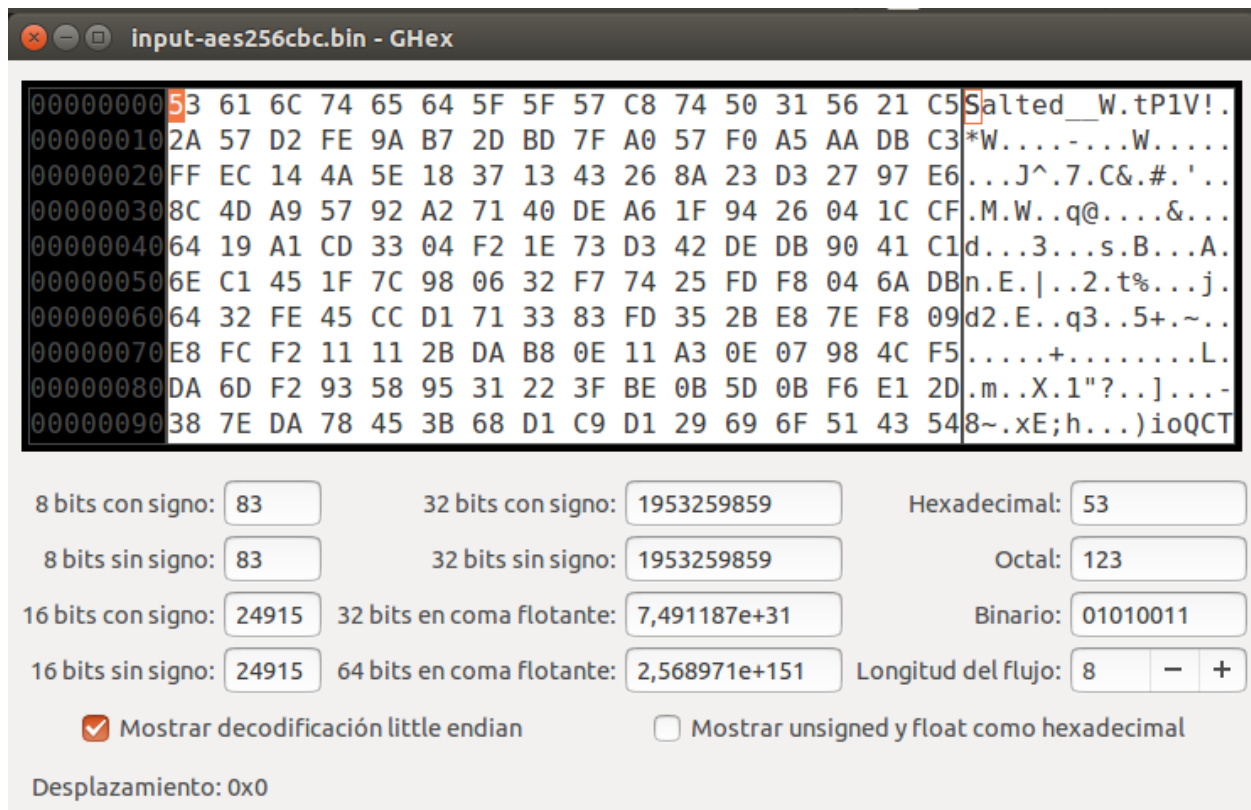


Imagen 9: Contenido del fichero *input-aes256cbc.bin*, resultado obtenido tras cifrar el fichero *input.bin* con AES-256 en modo CBC.

Ejercicio 6

Utilizando el criptosistema híbrido diseñado, cada uno debe cifrar el archivo *input.bin* con su clave pública para, a continuación, descifrarlo con la clave privada. Comparad el resultado con el archivo original.

Como en el ejercicio anterior ya he realizado el cifrado de *input.bin*, en este caso voy a descifrarlo. Para ello voy a suponer que he recibido únicamente el fichero *input* cifrado y el fichero *sessionkey* cifrado con la clave pública RSA.

Utilizando mi clave privada voy a descifrar primero el archivo *sessionkey-cifrado*. Para realizar este proceso voy a utilizar la orden *openssl rsautl* la cual permite realizar todas las operaciones de cifrado y descifrado de RSA, en este caso descifrado ya que utilizo la opción *-decrypt*.

```
$$ openssl rsautl -in sessionkey-cifrado -out sessionkey-descifrado -inkey nombreRSApriv.pem -decrypt
```

Como la clave privada estaba cifrada con contraseña para descifrar me ha pedido la contraseña (Imagen 10). El resultado obtenido tras descifrar el fichero se puede ver en Imagen 11.

```

manolo@manolo-Parallels:~/Documentos/SPSI/P2$ openssl rsautl -in sessionkey-cifrado
-out sessionkey-descifrado -inkey nombreRSApriv.pem -decrypt
Enter pass phrase for nombreRSApriv.pem:
manolo@manolo-Parallels:~/Documentos/SPSI/P2$

```

Imagen 10: Solicitud de contraseña para poder utilizar la clave privada que estaba protegida.

```

manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat sessionkey-descifrado
0a54cbde95a389c55f6c7203f758e12aa434e64ac4c20662704b349b29444751
-aes-256-cbc
manolo@manolo-Parallels:~/Documentos/SPSI/P2$

```

Imagen 11: Contenido del fichero sessionkey-descifrado.

Ahora que ya conozco la clave del sistema simétrico utilizado así como el propio sistema, voy a intentar descifrar el mensaje. Para ello utilizo de nuevo los conocimientos adquiridos en la práctica anterior.

```

$$ openssl aes-256-cbc -d -in input-aes256cbc.bin -out input-descifrado.bin -pass file:sessionkey-
descifrado

```

Como se puede observar en la Imagen 12, el mensaje ha sido descifrado correctamente. Esto ha sido posible realizarlo únicamente porque tengo la clave privada. Si alguien intercepta el mensaje no podrá descifrarlo ya que la clave privada no corresponderá con la clave pública utilizada para cifrar la clave usada en el sistema simétrico.

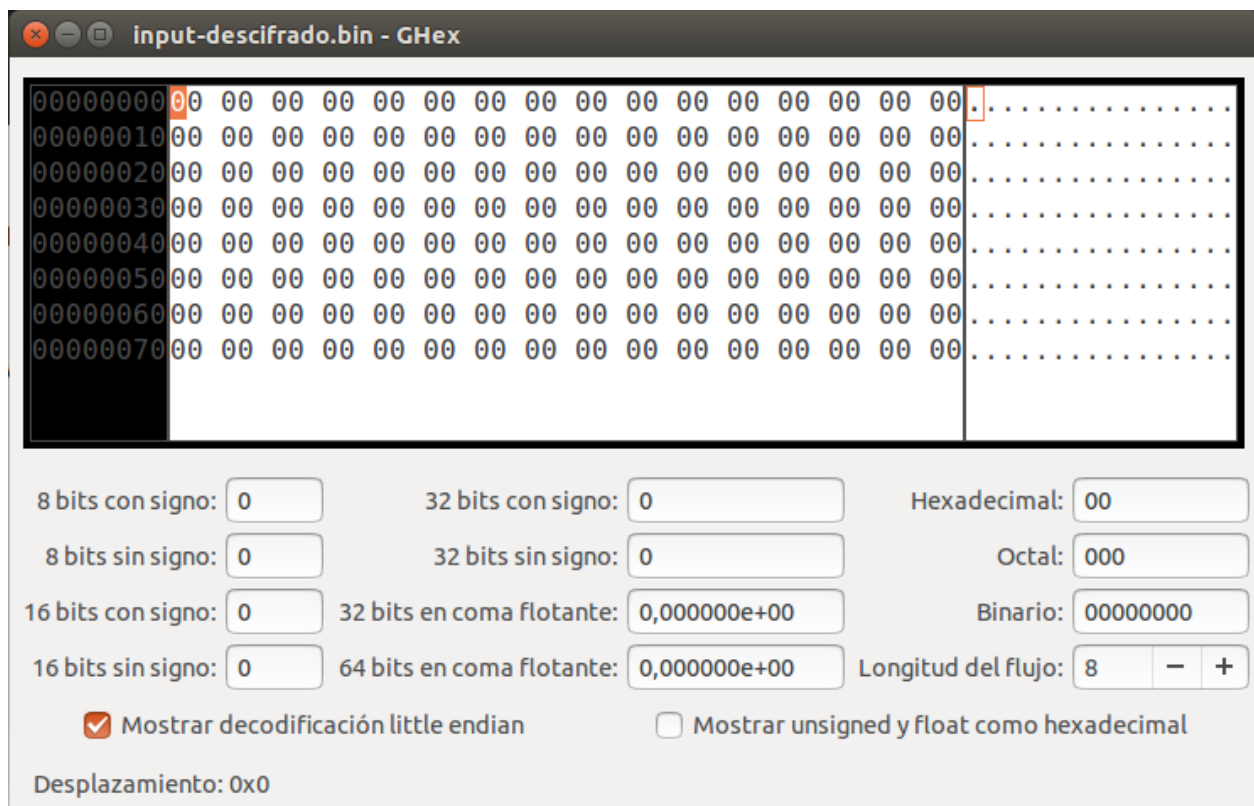


Imagen 12: Contenido del fichero input-descifrado.bin.

Ejercicio 7

Generad un archivo `stdECparam.pem` que contenga los parámetros públicos de una de las curvas elípticas contenidas en las transparencias de teoría. Si no lográis localizarlas haced el resto de la práctica con una curva cualquiera a vuestra elección de las disponibles en OpenSSL. Mostrad los valores.

Una de las curvas utilizadas en los apuntes de clase es la P-192. Para conocer las curvas soportadas por mi versión de OpenSSL utilizo la siguiente orden:

```
$$ openssl ecparam -list_curves
```

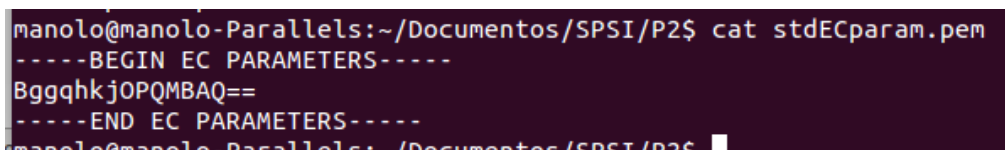
Según he podido observar, esta curva corresponde con la definida por OpenSSL como *prime192v1*:

prime192v1: NIST/X9.62/SECG curve over a 192 bit prime field

Para generar el archivo con los parámetros de la curva utilizo la siguiente orden cuya sintaxis se puede ver en [7]. Estos parámetros definen la curva y es justo lo que se necesita para poder cifrar un mensaje utilizando curvas elípticas.

```
$$ openssl ecparam -name prime192v1 -out stdECparam.pem
```

El archivo generado se puede ver en la Imagen 13.



```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat stdECparam.pem
-----BEGIN EC PARAMETERS-----
BggqhkJOPQMBAQ==
-----END EC PARAMETERS-----
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

Imagen 13: Contenido del fichero `stdECparam.pem`.

Ejercicio 8

Generad cada uno de vosotros una clave para los parámetros anteriores. La clave se almacenará en `nombreECkey.pem` y no es necesario protegerla por contraseña.

Como ya he generado en el apartado anterior los parámetros de la curva, en este ejercicio voy a utilizarlo para crear una clave. Para ello se tiene que volver a utilizar la orden `openssl ecparam` ya que también permite manipular los parámetros y poder extraer a partir de ellos la clave que se solicita.

```
$$ openssl ecparam -in stdECparam.pem -genkey -out nombreECkey.pem -noout
```

Con esta orden completa se genera la clave. Las opciones `-in` y `-out` indican el fichero desde el que se extraen los parámetros de la curva para ser utilizados y el fichero en el que guardar la clave generada, respectivamente. Con `-genkey` se indica que se quiere generar una clave. Finalmente, la opción `-noout` hace que en el fichero de salida no se vuelvan a guardar los parámetros de la curva. Así obtenemos finalmente el fichero únicamente con la clave (Imagen 14).

```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat nombreECkey.pem
-----BEGIN EC PRIVATE KEY-----
MF8CAQEGLT1TCfmgEyFp2Jg8zzvgdCpC108vaxqX6AKBggqhkjOPQMBAAE0AzIA
BLq6N9nbICRX+qxXIEMUKaAwFREgNt+eTrZ3B7102Thh/CsWMhst9MC3AX1Y//f3
ew==
-----END EC PRIVATE KEY-----
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

Imagen 14: Clave privada de la curva elíptica.

Ejercicio 9

“Extraed” la clave privada contenida en el archivo `nombreECkey.pem` a otro archivo que tenga por nombre `nombreECpriv.pem`. Este archivo deberá estar protegido por contraseña cifrándolo con 3DES. Mostrad sus valores.

Para realizar este ejercicio voy a utilizar la orden `openssl ec` la cual permite realizar procesamiento sobre claves de curvas elípticas [8]. Con ello se puede extraer la clave pública y privada por separado de la clave generada. La orden a usar es la siguiente:

```
$$ openssl ec -in nombreECkey.pem -out nombreECpriv.pem -des3
```

De nuevo, las opciones `-in` y `-out` sirven para indicar de qué fichero se lee la clave y en cual se almacena la parte privada de la misma. Con la opción `-des3` se indica que se quiere realizar un cifrado de la clave utilizando 3DES. Para ello se solicita introducir una contraseña dos veces, utilizando de nuevo la sugerida en el enunciado (“0123456789”). Finalmente, se puede observar en la Imagen 15 el resultado del fichero, el cual contiene la clave privada cifrada con triple DES.

```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat nombreECpriv.pem
-----BEGIN EC PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,504ADCF1BCBB2128

lvysIpmB6CkKxvi0Bg2ydhAHLNETeUG+IrIUtydAbdi/6/95lxu60X9+UjgrEe3
5c05E9IKXFP22z97IWNkL3RmeB0hbNsrqLmDelorMJ56oIo0LUpujW0ZngsH2+mN
eOgnjQ5sRiw=
-----END EC PRIVATE KEY-----
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

Imagen 15: Clave privada cifrada de la curva elíptica.

Ejercicio 10

Extraed en `nombreECpub.pem` la clave pública contenida en el archivo `nombreECkey.pem`. Como antes, `nombreECpub.pem` no debe estar cifrado ni protegido. Mostrad sus valores.

Para realizar este ejercicio hay que usar la orden del ejercicio anterior, la cual permitía manipular claves de curvas elípticas. En este caso se omite la opción `-des3` ya que no se requiere cifrar ni proteger la clave. En su lugar, se utiliza la opción `-pubout` ya que permite extraer la parte pública de la clave. La orden usada es:

```
$$ openssl ec -in nombreECkey.pem -out nombreECpub.pem -pubout
```

El valor de la clave pública de la curva elíptica se puede observar en la Imagen 16.

```
manolo@manolo-Parallels:~/Documentos/SPSI/P2$ cat nombreECpub.pem
-----BEGIN PUBLIC KEY-----
MEkwEwYHKoZIzj0CAQYIKoZIzj0DAQEDMgAEuro32dsgJFf6rFcgQxQpoDAVESA2
3550tnCHvU7ZOGH8KxYyGy30wLcBfVj/9/d7
-----END PUBLIC KEY-----
manolo@manolo-Parallels:~/Documentos/SPSI/P2$
```

Imagen 16: Clave pública de la curva elíptica.

Conclusión

En esta práctica nos hemos centrado en la generación y manipulación de claves de criptosistemas asimétricos. Con ello he podido apreciar el potencial de OpenSSL para este cometido. Esto ofrece un método simple y rápido de obtener claves (en RSA por ejemplo) que de otro modo serían computacionalmente más costosas de obtener de forma manual.

Además, con la parte de cifrado y descifrado RSA, junto con el error obtenido de que no se podían cifrar mensajes de tamaño mayor al de la clave, he observado que en muchos sitios se recomienda precisamente realizar un cifrado híbrido tal como el que se ha realizado en esta práctica. Esto permite cubrir las desventajas que ofrecen los sistemas de cifrado simétrico para intercambio de claves, ya que gracias a RSA se pueden enviar claves de forma segura y con garantías de que el único receptor posible es el propietario de la clave privada.

Índice de imágenes

Imagen 1: Generación de clave RSA usando OpenSSL.....	3
Imagen 2: Salida por pantalla del contenido de la clave en modo texto.....	3
Imagen 3: Introducción de contraseña para proteger la clave privada.....	4
Imagen 4: Valor cifrado de la clave privada.....	4
Imagen 5: Extracción de clave pública RSA.....	5
Imagen 6: Error al intentar cifrar usando RSA.....	5
Imagen 7: Contenido del fichero sessionkey.....	6
Imagen 8: Fichero sessionkey-cifrado.....	6
Imagen 9: Contenido del fichero input-aes256cbc.bin, resultado obtenido tras cifrar el fichero input.bin con AES-256 en modo CBC.....	7
Imagen 10: Solicitud de contraseña para poder utilizar la clave privada que estaba protegida.....	8
Imagen 11: Contenido del fichero sessionkey-descifrado.....	8
Imagen 12: Contenido del fichero input-descifrado.bin.....	8
Imagen 13: Contenido del fichero stdECparam.pem.....	9
Imagen 14: Clave privada de la curva elíptica.....	10
Imagen 15: Clave privada cifrada de la curva elíptica.....	10
Imagen 16: Clave pública de la curva elíptica.....	11

Bibliografía

- [1] Documentación OpenSSL para generar claves RSA.
<https://www.openssl.org/docs/man1.0.2/apps/genrsa.html>
- [2] Documentación OpenSSL para manipular claves RSA.
<https://www.openssl.org/docs/man1.0.2/apps/rsa.html>
- [3] Documentación OpenSSL sobre utilidades de RSA.
<https://www.openssl.org/docs/man1.0.2/apps/openssl-rsautl.html>
- [4] PKCS#1: Estándar criptográfico RSA
<https://tools.ietf.org/html/rfc3447>
- [5] Documentación OpenSSL para crear claves aleatorias.
<https://www.openssl.org/docs/man1.0.2/apps/openssl-rand.html>
- [6] Documentación OpenSSL sobre encriptado simétrico.
<https://www.openssl.org/docs/man1.0.2/apps/enc.html>
- [7] Documentación OpenSSL para generar y manipular parámetros de curvas elípticas.
<https://www.openssl.org/docs/man1.0.2/apps/openssl-ecparam.html>
- [8] Documentación OpenSSL sobre procesamiento de claves en curvas elípticas.
<https://www.openssl.org/docs/man1.0.2/apps/openssl-ec.html>