

Práctica 2. Criptosistemas Asimétricos

Patricia Maldonado Mancilla

Índice

1. Generad, cada uno de vosotros, una clave RSA (que contiene el par de claves) de 901 bits. Para referirnos a ella supondré que se llama <nombre>RSAkey.pem. Esta clave no es necesario que esté protegida por contraseña. 4
2. Extraed la clave privada contenida en el archivo nombre RSAkey.pem a otro archivo que tenga por nombre nombre RSApriv.pem. Este archivo deberá estar protegido por contraseña cifrándolo con AES-128. Mostrad sus valores 5
3. Extraed en <nombre>RSAPub.pem la clave publica contenida en el archivo <nombre>RSAkey.pem. Evidentemente <nombre>RSAPub.pem no debe estar cifrado ni protegido. Mostrad sus valores. 6
4. Reutilizaremos el archivo binario input.bin de 1024 bits, todos ellos con valor 0, de la práctica anterior. 6
5. Intentad cifrar input.bin con vuestras claves públicas. Explicad el mensaje de error obtenido. 7
6. Diseñad un cifrado híbrido, con RSA como criptosistema asimétrico. 7
7. 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. 10
8. 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. 11
9. Generad cada uno de vosotros una clave para los parámetros anteriores. La clave se almacenará en <nombre>ECkey.pem y no es necesario protegerla por contraseña. 13
10. Extraed la clave privada contenida en el archivo <nombre>ECkey.pem a otro archivo que tenga por nombre <nombre>ECpriv.pem. Este archivo deberá estar protegido por contraseña. Mostrad sus valores. 13
11. Extraed en <nombre>ECpub.pem la clave pública contenida en el archivo <nombre>ECkey.pem. Como antes <nombre>ECpub.pem no debe estar cifrado ni protegido. Mostrad sus valores. 14

Índice de figuras

1.1. Comando para generar clave RSA	4
1.2. Clave RSA generada	4
2.1. Comando para la extracción de la clave privada RSA generada	5
2.2. Clave privada RSA extraída	5
3.1. Comando para extraer la clave publica RSA generada	6
3.2. Clave pública RSA extraída	6
4.1. Archivo input.bin	6
5.1. Comando para cifrar input.bin con clave pública	7
6.1. Comando para el cifrado híbrido	8
6.2. Contenido sessionkey	8
6.3. Comando para cifrar con la clave pública	9
6.4. Contenido sessionkey cifrado	9
6.5. Contenido sessionkey cifrado	9
6.6. Comando para cifrar el mensaje	9
6.7. Contenido inputs.bin	10
7.1. Contenido inputs.bin	10
7.2. Descifrado del archivo sessionkeycifrado con la clave privada	10
7.3. Contenido sessionkeydescifrado	11
7.4. Descifrado inputs	11
7.5. Contenido de descifrar inputs.bin	11
8.1. Listado de curvas OpenSSL	12
8.2. Comando para curva P-192	12
8.3. Contenido generado curva P-192	13
9.1. Comando para generar la clave EC	13
9.2. Contenido patriciaECkey.pem	13
10.1. Comando para extraer la clave privada EC	14
10.2. Contenido de extraer la clave privada	14
11.1. Comando para extraer la clave pública	14
11.2. Contenido de la clave pública	15

1. **Generad, cada uno de vosotros, una clave RSA (que contiene el par de claves) de 901 bits. Para referirnos a ella supondré que se llama <nombre>RSAkey.pem. Esta clave no es necesario que esté protegida por contraseña.**

Para generar las claves RSA utilizamos el comando `openssl genrsa`. Se emplea para generar un par de claves pública/privada que se almacenan de forma conjunta.

Además indicamos con el argumento `-out` el archivo de salida para almacenar la clave generada con extensión `.pem` y le indicamos el tamaño que tendrá la clave, en este caso 901 bits.

El comando utilizado para generar la clave RSA es el siguiente:

```
patri@patri:~/Escritorio$ openssl genrsa -out patriciaRSAkey.pem 901
Generating RSA private key, 901 bit long modulus
.+++++
.....+++++
e is 65537 (0x10001)
```

Figura 1.1: Comando para generar clave RSA

En la imagen siguiente observamos la clave generada RSA de 901 bits en el archivo `patriciaRSAkey.pem`. Contiene nuestro par de claves pública/privada.

```
patriciaRSAkey.pem x
1 |-----BEGIN RSA PRIVATE KEY-----
2 MIICFAIBAAJxF1lt/P+zMOaQJzsNj8A5VaOr10Zvysp9qy5WtjCLq9nupW8KgNsz
3 JLVqXUwbzAA5NPRBJpwGOpVqGW1FH60PZ8BeFkSckG1t19ZB0zPTtLXF5hJTqZNk
4 6flaQs5nuhiLct8cQ2M3isfxZLBKZ4AcgOsCAwEAAQJxAnFn8aWrUP1+rawfEiBR
5 n2U9kIudHMGs/mEdz3gq1PM7ZfHFTpAgv00I8PQKoICRe7nmqyx99lk0pGyTbjBf
6 BBhdN12n+ohcQMiaoyGjfSHKQZiIyekVWSMfa0/5cUmj+4KFqt9CTLS7wyu8M0De
7 7VECOQdNsAOSiqC4xey7hULB3jwCFA5DUvJQtM9wzlzBBCOBBQHhy3jrAUXguvvt
8 QgQgyncNkasdl3fRcwI5AzJv8Uqma8XQBQVefkZ2v7MBgIVm2XJxEAR8qR/32yTD
9 yluVau9hvdw0q75eiMxm0ToJk1sxMNSpAjkCODw8ebckLcQJtNxFc8JlHoSV6SQ2
10 fQzW1wQA8LCfUCeB12wNre7PHAP9FfiwZH7k0/1/shis070CODIArnLGuaboQqZF
11 U6myb5vcJ6H+2qInTjc39o1Bc14XqDFY06RMCHExtrZ2yBzU20uUcjbMHROhAjke
12 1sa80Pas5MNCvqcX3FfD9ocS3AeESCFncg50l+y2Drt+8F/WX3jzy4YlfgZUJbsp
13 HZQBLKsfQvo=
14 -----END RSA PRIVATE KEY-----
```

Figura 1.2: Clave RSA generada

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

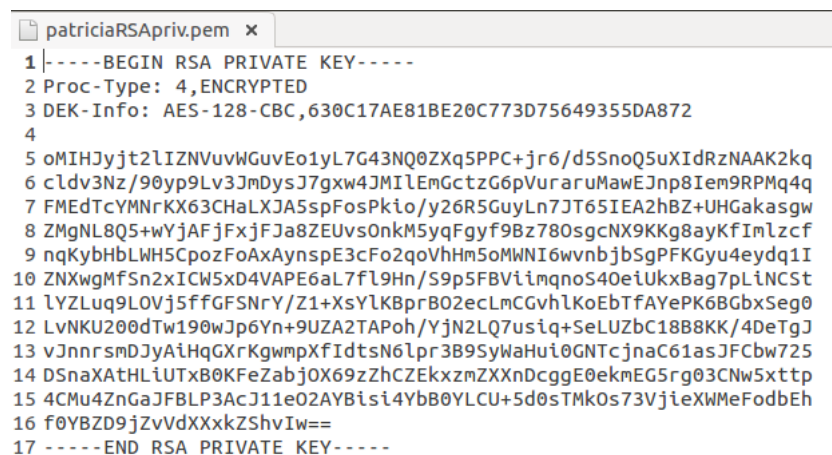
Para manipular claves generadas usamos el siguiente comando con el que vamos a extraer del archivo patriciaRSAkey.pem la parte privada.

```
patri@patri:~/Escritorio$ openssl rsa -in patriciaRSAkey.pem -out patriciaRSApriv.pem -aes128
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Figura 2.1: Comando para la extracción de la clave privada RSA generada

- **rsa**: manipula claves generadas, con él que extraemos la parte privada.
- **-in**: especificamos el nombre de archivo de entrada que contiene la clave RSA.
- **-out**: especificamos el archivo de salida para almacenar la clave privada.
- **aes-128**: lo añadimos para cifrar la clave privada.
- Nos pide una contraseña y utilizaremos la que nos proporciona el pdf: 0123456789.

En la imagen siguiente visualizamos el contenido del cifrado de la clave privada en archivo patriciaRSApriv.pem, también nos muestra el método que hemos utilizado para cifrarla.



```
patriciaRSApriv.pem x
1 |-----BEGIN RSA PRIVATE KEY-----
2 Proc-Type: 4, ENCRYPTED
3 DEK-Info: AES-128-CBC,630C17AE81BE20C773D75649355DA872
4
5 oMIHJyjt2LIzNVuvWGuVeo1yL7G43NQ0ZXq5PPC+jr6/d5SnoQ5uXIIdRzNAAK2kq
6 cldv3Nz/90yp9Lv3JmDysJ7gxw4JMIEmGctzG6pVurarumawEJnp8Iem9RPMq4q
7 FMEdTcYMNrKX63ChALXJA5spFosPkio/y26R5GuyLn7JT65IEA2hBZ+UHGakasgw
8 ZMgNL8Q5+wYjAFjFxfJJa8ZEUvsOnkM5yqFgyf9Bz780sgcNX9KKg8ayKfImLzcf
9 nqKybHbLWH5CpozFoAxAynspE3cFo2qoVhHm5oMwNI6wvnbjbsgPFGyu4eydq1I
10 ZNXwgMfSn2xICW5xD4VAPE6aL7fL9Hn/S9p5FBVimqnoS40eiUkxBag7pLiNCSt
11 lYZLqu9LOVj5ffGFSNrY/Z1+XsYlKBprB02ecLmCGvhlKoEbTfAYePK6BGbxSeg0
12 LvNKU200dTw190wJp6Yn+9UZA2TAPoh/YjN2LQ7usiq+SeLUZbC18B8KK/4DeTgJ
13 vJnnrsmDJyAiHqGXRkgwmpXfIdtsN6lpr3B9SyWaHui0GNTcjnaC61asJFCbw725
14 DSnaXAtHLiUTxB0KFeZabj0X69zZhCEkxzmZXXnDcggE0ekmEG5rg03CNw5xttp
15 4CMu4ZnGaJFBLP3AcJ11e02AYBisi4YbB0YLCU+5d0sTMKos73VjieXWMeFodBEh
16 f0YBZD9jZvVdXXxkZShvIw==
17 -----END RSA PRIVATE KEY-----
```

Figura 2.2: Clave privada RSA extraida

3. **Extraed en <nombre>RSAPub.pem la clave publica contenida en el archivo <nombre>RSAkey.pem. Evidentemente <nombre>RSAPub.pem no debe estar cifrado ni protegido. Mostrad sus valores.**

Vamos a extraer la clave pública del archivo patriciaRSAkey.pem con el siguiente comando:

```
patri@patri:~/Escritorio$ openssl rsa -in patriciaRSAkey.pem -pubout -out patriciaRSAPub.pem
writing RSA key
```

Figura 3.1: Comando para extraer la clave publica RSA generada

Como en el caso anterior añadimos el archivo de entrada para extraer la clave pública y el de salida para guardarla. Además hemos añadido la opción **-pubout** para extraer de la clave RSA la parte pública, ya que por defecto extrae la clave privada. La clave pública no tiene que estar protegida por contraseña por lo que no usamos el sistema de cifrado.

Salida generada de la extracción de la parte pública en el archivo patriciaRSAPub.pem:

```
patriciaRSAPub.pem x
1|-----BEGIN PUBLIC KEY-----
2 MIGMMA0GCSqGSIb3DQEBAQUAA3sAMHgCcrDZbfz/szDmkCc7DY/A0VWjq9dGb8rK
3 fasuVrYwi6vZ7qVvCoDbMyZVal1MG8wA0TT0QSacBjqVahltrr+jj2fAXhZEnJBt
4 bdfWQTsz07S1xeYSU6mTZOn5WkLOZ7oYiwrFHENjN4rH8WZQSmeAHIDrAgMBAAE=
5 -----END PUBLIC KEY-----
```

Figura 3.2: Clave pública RSA extraida

4. **Reutilizaremos el archivo binario input.bin de 1024 bits, todos ellos con valor 0, de la práctica anterior.**

Contenido del archivo input.bin relleno de ceros de 1024 bits.



Figura 4.1: Archivo input.bin

5. Intentad cifrar input.bin con vuestras claves públicas. Explicad el mensaje de error obtenido.

Para cifrar input.bin con mi clave pública uso el siguiente comando:

```
patri@patri:~/Escritorio$ openssl rsautl -encrypt -in input.bin -out inputcp.bin -inkey patrici
aRSApub.pem -pubin
RSA operation error
140331315177120:error:0406D06E:rsa routines:RSA_padding_add_PKCS1_type_2:data too large for key
size:rsa_pk1.c:151:
```

Figura 5.1: Comando para cifrar input.bin con clave pública

Los argumentos para realizarlo son los siguientes:

- **rsautl**: se puede usar para firmar, verificar, cifrar y descifrar datos usando el algoritmo RSA. En este caso lo usaremos para cifrar input.bin
- **-encrypt**: con esta opción indicamos que queremos encriptar y lo haremos con la clave pública RSA.
- **-in**: le pasamos input.bin para cifrar con la clave pública.
- **-out**: salida para guardar el cifrado de input.bin.
- **-inkey**: el archivo de entrada que le pasamos con la clave.
- **-pubin**: indicamos que queremos cifrar con la clave pública.

Podemos observar que no se ha realizado correctamente, y nos muestra un mensaje de error, "los datos son demasiado grandes para el tamaño de clave". Esto ocurre porque estamos intentando cifrar un archivo de 1024 bits mayor que el que utilizamos con la clave pública de 901 bits. RSA no permite cifrar mensajes mayores que su clave, ya que es un cifrado de flujo y no por bloques.

6. Diseñad un cifrado híbrido, con RSA como criptosistema asimétrico.

El cifrado híbrido es un método criptográfico que usa tanto un cifrado simétrico como un asimétrico.

Solventa los problemas de privacidad que podría suponer el uso del cifrado simétrico y el tiempo de procesamiento del uso del cifrado asimétrico, de esta forma se combinan ambas para su uso.

Cuando por ejemplo, deseamos mandar un mensaje, este mensaje lo cifraremos con una clave simétrica, conocida en este proceso como la clave de sesión.

Después, cifraremos con la clave pública del receptor la clave de sesión. Cuando el receptor reciba el mensaje, primero descifrará con su clave privada la clave de sesión. Una vez posea la clave de sesión ya puede acceder al contenido del mensaje.

El modo de proceder ser a el siguiente:

1. **El emisor debe seleccionar un sistema simétrico con su correspondiente modo de operación.**

Voy a seleccionar el cifrado simétrico AES-256-ecb, con tamaño de clave 256 y modo ECB.

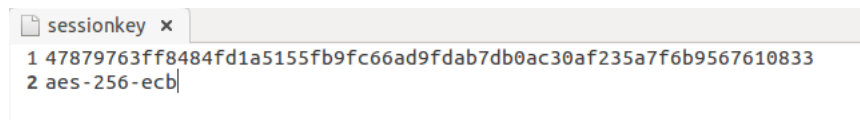
2. **El emisor generará un archivo de texto, llamado por ejemplo sessionkey con dos líneas. La primera línea contendrá una cadena aleatoria hexadecimal cuya longitud sea la requerida para la clave del criptosistema simétrico. OpenSSL permite generar cadenas aleatorias con el comando openssl rand. La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

- Generamos la clave aleatoria mediante **rand**.
- **-hex**, con esta opción indicamos que genere una cadena en hexadecimal. Seguido de 32 bytes (256 bits) que es el tamaño de clave que utilizaremos para el cifrado simétrico que hemos elegido aes-256. El archivo de salida se llamará sessionkey.

```
patri@patri:~/Escritorio$ openssl rand -hex 32 -out sessionkey
```

Figura 6.1: Comando para el cifrado híbrido

En el archivo que hemos generado sessionkey se ha guardado la cadena aleatoria con la longitud de la clave y añadimos una segunda linea indicando el tipo de cifrado elegido aes-256-ecb.



```
1 47879763ff8484fd1a5155fb9fc66ad9fdab7db0ac30af235a7f6b9567610833
2 aes-256-ecb
```

Figura 6.2: Contenido sessionkey

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

Utilizamos el archivo patriciaRSAPub.pem que contiene nuestra clave pública. El comando para cifrar con la clave pública es el que hemos usado en el ejercicio anterior en el que nos dió error, en este caso si es posible cifrar usando RSA ya que el tamaño del archivo es menor que el de la clave. Vamos a cifrar el archivo sessionkey con la clave publica almacenada en patriciaRSA.pub, y generaremos el archivo de salida sessionkeycifrado.


```
patri@patri:~/Escritorio$ openssl rsautl -encrypt -in sessionkey -out sessionkey
cifrado -inkey patriciaRSApub.pem -pubin
```

Figura 6.3: Comando para cifrar con la clave pública

Contenido de sessionkey cifrado por la clave pública:

sessionkeycifrado

1 [09][09][09][09]äS«6<Đsí[09][14]Dò[09][94]Sø[09][09][09]oi[09][7f]Äěj[09][y]:[09][09][09]û#[09][14]ô[09][1A]_ā[09][N][09]
lávK+äk*KÊæÜ«LÛPëpâ(çóð[09]c --[09]ý[09]H-[09][09][09]Nw[nôž[09][09][09]«ŠpBZt[09]±sz,[09]i
[09][94]€jšĚŸ+ÁÑÚ[09]Qø

Figura 6.4: Contenido sessionkey cifrado

sessionkeycifrado - GHex

Archivo Editar Ver Ventanas Ayuda

00000000	01 9C 14 E5 53 AB 36 3C D0 73 ED 1A 44 D2 94 53	...S.6<.s..D..S
00000010	AE 9E 6F EC 7F C0 EB 6A 83 79 ED 3A 01 89 FB 23	...o...j.y:...#
00000020	14 D4 1A 5F E3 0B D1 03 20 A0 EF E1 76 4B B9 E3	..._...vK...
00000030	4B B3 4B C8 61 EA DC AB 6C D9 DE BD B5 E2 28 90	K.K.a...l...(.c...H~...Nw
00000040	D4 1C 63 1D AF 2D 7F FD 91 48 7E 93 0C 05 4E 77	[.....Z...s.
00000050	5B F1 D6 BF 8F 8E AB A6 FE DF 5A EF 01 B1 73 BF	[...j...+...Q
00000060	2C 0B EE 5B 94 A4 6A A7 CB BE 2B C1 D1 FA 1B 51	
00000070	F8	.

Figura 6.5: Contenido 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`.

Ciframos el archivo `input.bin` con el método `aes-256-ecb`, con la opción `-pass file:sessionkey` con el que contiene la clave de 32 bits generadada anteriormente.

```
patri@patri:~/Escritorio$ openssl enc -aes-256-ecb -in input.bin -pass file:sessionkey -out inputs.bin
```

Figura 6.6: Comando para cifrar el mensaje

El archivo de salida que hemos generado es el siguiente:

```

00000000 53 61 6C 74 65 64 5F 5F 66 18 2B FC 7D 29 E6 79 Salted__f.+.)}.y
00000010 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000020 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000030 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000040 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000050 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000060 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000070 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000080 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000090 71 C1 A9 74 A0 87 AD EE BB C5 20 B0 08 C4 5C FB q..t.....\..

```

Figura 6.7: Contenido inputs.bin

7. 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.

En el ejercicio anterior ciframos el mensaje con nuestra clave pública con el siguiente resultado:

```

00000000 53 61 6C 74 65 64 5F 5F 66 18 2B FC 7D 29 E6 79 Salted__f.+.)}.y
00000010 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000020 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000030 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000040 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000050 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000060 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000070 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000080 DD 36 32 B8 F6 25 CC F4 88 D1 C2 CB DB DF 72 2C .62..%.....r,
00000090 71 C1 A9 74 A0 87 AD EE BB C5 20 B0 08 C4 5C FB q..t.....\..

```

Figura 7.1: Contenido inputs.bin

A continuación realizamos el descifrado de sessionkeycifrado.

- **rsault** se puede usar para firmar, verificar, cifrar y descifrar datos usando el algoritmo RSA. En este caso lo usaremos para descifrar.
- Mediante **-decrypt** indicamos que queremos descifrar.
- **-inkey**: Le pasamos el archivo que contiene la clave privada.


```

patri@patri:~/Escritorio$ openssl rsautl -decrypt -in sessionkeycifrado -inkey patricia
RSApriv.pem -out sessionkey_descifrado
Enter pass phrase for patriciaRSApriv.pem:

```

Figura 7.2: Descifrado del archivo sessionkeycifrado con la clave privada

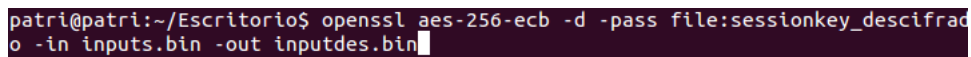
Se puede observar en la siguiente imagen como se ha descifrado correctamente, ya que hemos vuelto al archivo original donde teníamos la clave de 32 bits y el método de cifrado.



```
sessionkey_descifrado x
1 47879763ff8484fd1a5155fb9fc66ad9fdab7db0ac30af235a7f6b9567610833
2 aes-256-ecb
```

Figura 7.3: Contenido sessionkeydescifrado

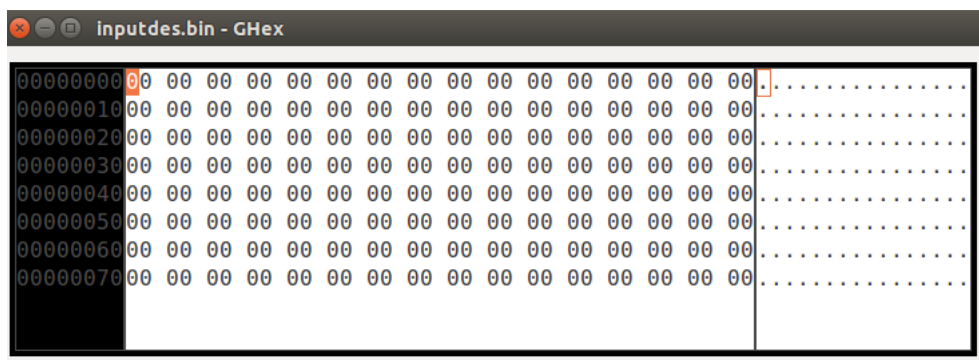
Finalmente para descifrar el archivo inputs.bin usaremos el siguiente comando:



```
patri@patri:~/Escritorio$ openssl aes-256-ecb -d -pass file:sessionkey_descifrado -in inputs.bin -out inputdes.bin
```

Figura 7.4: Descifrado inputs

El archivo se ha descifrado correctamente y obtenemos el mismo contenido que el archivo input.bin de 1024 bits relleno de ceros.



The screenshot shows a hex editor window titled 'inputdes.bin - GHex'. The main area displays a grid of hex values, all of which are '00'. The grid is organized into rows and columns, with the first column showing the offset (e.g., 00000000, 00000010, etc.) and the subsequent columns showing the hex data. The right side of the window shows the corresponding ASCII representation, which consists of a series of dots, indicating that the data is non-printable (zeros).

Figura 7.5: Contenido de descifrar inputs.bin

8. 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.

Listado de curvas OpenSSL:

```
patri@patri: ~  
patri@patri:~$ 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  
sect131r2 : SECG curve over a 131 bit binary field  
sect163k1 : NIST/SECG/WTLS curve over a 163 bit binary field  
sect163r1 : SECG curve over a 163 bit binary field  
sect163r2 : NIST/SECG curve over a 163 bit binary field  
sect193r1 : SECG curve over a 193 bit binary field  
sect193r2 : SECG curve over a 193 bit binary field  
sect233k1 : NIST/SECG/WTLS curve over a 233 bit binary field  
sect233r1 : NIST/SECG/WTLS curve over a 233 bit binary field  
sect239k1 : SECG curve over a 239 bit binary field  
sect283k1 : NIST/SECG curve over a 283 bit binary field  
sect283r1 : NIST/SECG curve over a 283 bit binary field  
sect409k1 : NIST/SECG curve over a 409 bit binary field  
sect409r1 : NIST/SECG curve over a 409 bit binary field
```

Figura 8.1: Listado de curvas OpenSSL

Una de las curvas de las transparencias es la curva P-192 correspondiente a prime192v1.

Para manipular y generar curvas elípticas se emplea el comando openssl ecparam. Generamos el archivo stdECparam.pem que se definen con los parámetros de prime192v1.

```
patri@patri:~/Escritorio$ openssl ecparam -name prime192v1 -out stdECparam.pem
```

Figura 8.2: Comando para curva P-192

En el archivo general se muestran los parámetros de prime192v1.

```

1 |-----BEGIN EC PARAMETERS-----
2 BggqhkjOPQMBAQ==
3 -----END EC PARAMETERS-----

```

Figura 8.3: Contenido generado curva P-192

9. Generad cada uno de vosotros una clave para los parámetros anteriores. La clave se almacenará en <nombre>ECkey.pem y no es necesario protegerla por contraseña.

Comando para generar la clave con los parámetros creados anteriormente:

```

patri@patri:~/Escritorio$ openssl ecparam -genkey -in stdECparam.pem -out patriciaECkey.pem

```

Figura 9.1: Comando para generar la clave EC

Le pasamos un fichero de entrada mediante **-in**, el fichero que contiene los parámetros de la curva para generar la clave, además con **-out** le indicamos como se llamará el archivo de salida. Con esta opción **-genkey** generamos una clave privada EC usando los parámetros especificados.

Contenido del archivo patriciaECkey.pem:

```

patriciaECkey.pem x
1 |-----BEGIN EC PARAMETERS-----
2 BggqhkjOPQMBAQ==
3 -----END EC PARAMETERS-----
4 -----BEGIN EC PRIVATE KEY-----
5 MF8CAQEEGM99s/d3eubLJE0heiJHulIaqXEorV+P36AKBggqhkjOPQMBAAE0AzIA
6 BEICss9mERcFaGUsxk8Ic193xVypcNYE3DbmU5pl+q3inUxL2D87JAIU5afsB73
7 NA==
8 -----END EC PRIVATE KEY-----

```

Figura 9.2: Contenido patriciaECkey.pem

10. Extraed la clave privada contenida en el archivo <nombre>ECkey.pem a otro archivo que tenga por nombre <nombre>ECpriv.pem. Este archivo deber a estar protegido por contraseña. Mostrad sus valores.

Comando para extraer la clave privada del archivo patriciaECkey.pem:

```
patri@patri:~/Escritorio$ openssl ec -in patriciaECkey.pem -out patriciaECpriv.p
em -aes128
read EC key
writing EC key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Figura 10.1: Comando para extraer la clave privada EC

En este caso usamos `openssl ec` para procesar claves de curvas elípticas. Con lo que podemos obtener la clave pública y la privada. Se le pasa el archivo `patriciaECkey.pem` que contiene la clave, y obtendremos el archivo de salida `patriciaECpriv.pem` que contendrá la clave privada que extraeremos. Además le añadimos el método con el que cifraremos la clave, `-aes`. Nos pide una contraseña e introduciremos la que nos proporciona el pdf: 0123456789.

Contenido de la clave privada que hemos cifrado con `-aes128`:

```
patriciaECpriv.pem x
1 |-----BEGIN EC PRIVATE KEY-----
2 Proc-Type: 4,ENCRYPTED
3 DEK-Info: AES-128-CBC,2951D750B8C54112C94783D7DFD548D6
4
5 ASE2Lo0GeVJCF5Xv5AE404pv2hxrdo2Je/346zE+7whIUYqAnw0T8tla6vGirAb
6 0f80/UeS4TCwgZ4Il6rgRF2lobFeTWw47NuRYoMCn5/+0cBn4lIYnp9D1S/rFvQi
7 m8wZ0URi5JwtoVWKxeuNmg==
8 -----END EC PRIVATE KEY-----
```

Figura 10.2: Contenido de extraer la clave privada

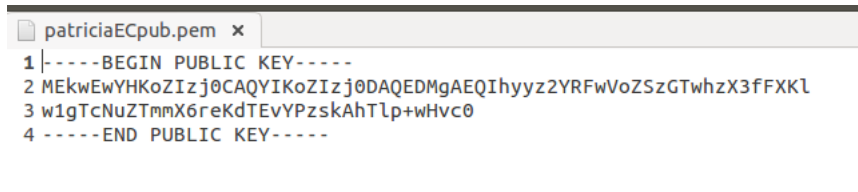
11. Extraed en `<nombre>ECpub.pem` la clave pública contenida en el archivo `<nombre>ECkey.pem`. Como antes `<nombre>ECpub.pem` no debe estar cifrado ni protegido. Mostrad sus valores.

Usaremos el comando `openssl ec` como en el ejercicio anterior para procesar curvas elípticas. En este caso vamos a extraer la clave pública, que indicamos con el argumento `-pubout`, indicamos el archivo de entrada `patriciaECkey.pem` con los parámetros de la curva. Y generamos el archivo de salida con la clave pública `patriciaECpub.pem`.

```
patri@patri:~/Escritorio$ openssl ec -pubout -in patriciaECkey.pem -out patricia
ECpub.pem
read EC key
writing EC key
```

Figura 11.1: Comando para extraer la clave pública

Contenido de la clave pública de la curva elíptica:



```
patriciaECpub.pem x
1 -----BEGIN PUBLIC KEY-----
2 MEkwEwYHKoZIZj0CAQYIKoZIZj0DAQEDMgAEQIhyyz2YRFwVoZSzGTwhzX3fFXKl
3 w1gTcNuZTmmX6reKdTEvYPzskAhTlp+wHvc0
4 -----END PUBLIC KEY-----
```

Figura 11.2: Contenido de la clave pública