

UNIVERSIDAD DE GRANADA  
E.T.S.I INFORMÁTICA Y TELECOMUNICACIONES  
GRADO EN INGENIERÍA INFORMÁTICA (GII)

---

## Memoria Práctica 2: Criptosistemas Asimétricos

---

Seguridad y Protección de Sistemas Informáticos  
CURSO 2017-2018

Arthur Rodríguez Nesterenko - DNI Y1680851W

5 de noviembre de 2017

# Índice

1. Tarea 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.	3
2. Tarea 2: “Extraed”la clave privada contenida en el archivo nombreRSA-key.pem a otro archivo que tenga por nombre nombreRSApriv.pem. Este archivo deberá estar protegido por contraseña cifrándolo con AES-128. Mostrad sus valores.	4
3. Tarea 3: Extraed en nombreRSApub.pem la clave pública contenida en el archivo nombreRSAkey.pem. Mostrad sus valores.	4
4. Tarea 4-5: Reutilizar el fichero input.bin de 1024 bits de la práctica anterior e intentad cifrar input.bin con vuestras claves publicas. Explicad el resultado.	5
5. Tarea 6: Diseñad un cifrado híbrido, con RSA como criptosistema asimétrico. El modo de proceder será el siguiente	6
5.1. Tarea 6.1: El emisor debe seleccionar un sistema simétrico con su correspondiente modo de operación . . . . .	6
5.2. Tarea 6.2: El archivo sessionkey se cifrará con la clave pública del receptor.	6
5.3. El mensaje se cifrará utilizando el criptosistema simétrico, la clave se generará a partir del archivo anterior mediante la opción -pass file:sessionkey.	6
6. Tarea 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 eleccion de las disponibles en OpenSSL. Mostrad los valores.	8
6.1. Tarea 7.1: 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. . . . .	9
6.2. Tarea 7.2: Extraed la clave privada contenida en el archivo nombreEC-key.pem a otro archivo que tenga por nombre nombreECpriv.pem. Este archivo deber a estar protegido por contraseña cifrándolo con 3DES. Mostrad sus valores. . . . .	10
6.3. Tarea 7.3: Extraed en nombreECpub.pem la clave publica contenida en el archivo nombreECkey.pem . . . . .	10

1. Tarea 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.

La generación de claves RSA se realiza a través del comando `openssl genrsa` donde además se le indica el fichero de salida junto con la extensión (por defecto, PEM) y el número de bits de la misma. El comando y la salida del mismo se muestra a continuación.

Comando: `openssl genrsa -out nombreRSAkey.pem 768`

```
[cvi075019:ficheros Arthur18$ openssl genrsa -out nombreRSAkey.pem 768
Generating RSA private key, 768 bit long modulus
.....++++++
.....++++++
e is 65537 (0x10001)
```

Figura 1.1: Generación del par de claves pública-privada con RSA

Para comprobar que se ha generado correctamente, mostramos el fichero `nombreRSAkey.pem` que contiene nuestro par de claves pública-privada, que es **almacenado de forma conjunta**, lo que significa que a partir de este fichero podremos separar la clave pública de la privada, acciones que realizaremos a continuación.

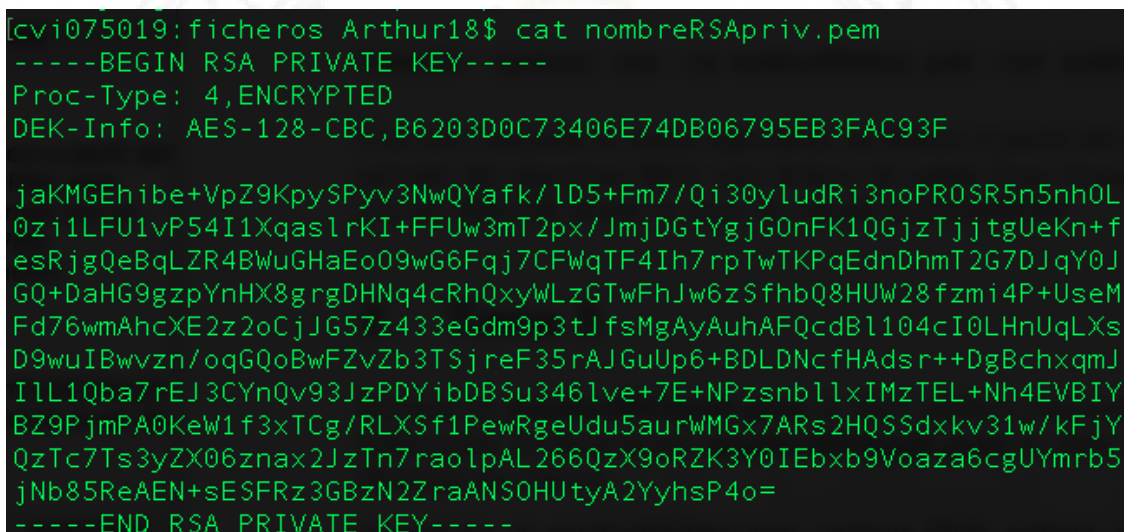
```
[cvi075019:ficheros Arthur18$ cat nombreRSAkey.pem
-----BEGIN RSA PRIVATE KEY-----
MIIBYQIBAAJhA0sk+iq/YdWVfH+6BMuZTDLni7X2AE+t4pTlpmrdruIhyry9ZbAI
7kEar1/NhQ0PnGTsXs5WrPSaz/TsGywsovGMPcJyXYMG30K1J9NCLu0Fo3I1PZEI
lnaWvdWkgyThHwIDAQABAmAKA5Yf0eyxst8NdXeU0B99n3zEsAhNXGNk4e7XaD91
Y+kFZq948rhqzk0puc/rwXlbr35Qpr2yeDpg2AEBWh7cG1Jy0EYAssk5qoxQpqN0
K7k300gtvTb19+s2HAYAw9ECMQD44aTdTkB4egT+hsCA5TSJh+mk0fNuaKX+2zqL
sd/1uVTsPRc0GgeXAQVloTyhh8kCMQDx3r+dd753XYcUoSvy/sRUjgvhN0Tml54J
ggR3oY5rReWA//X84027o4PMtiEbZacCMEB0x+rc0kuhZlu48HAXbErWHUenbEFC
7wKzK1V1isgr6HeKVsc7LI4KIKiyem5nKQIwfPlymuPPh4/NB0zMD6+yNFp9Ku2Q
xNFCuF70/fd9R1KEZqZbLUMp3CV+KZ2F98kLAjBVIbvBksXRamXeq9AyMpBCPc+I
LxZe0XDtgZp93qlEry/TJwXh3+KIVZmZTDjWZ9c=
-----END RSA PRIVATE KEY-----
```

Figura 1.2: Fichero `nombreRSAkey.pem` conteniendo el par de claves

2. Tarea 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.

Comando: `openssl rsa -in nombreRSAkey.pem -out nombreRSApriv.pem -outform PEM -aes128`

Con este comando se busca especificar un fichero a partir del cual se pueda extraer la clave privada del algoritmo RSA; este fichero de salida tiene formato PEM y estará cifrado con un **AES-128** donde la clave de cifrado es **0123456789**. Mostramos el resultado del comando a continuación.



```
[cvi075019:ficheros Arthur18$ cat nombreRSApriv.pem
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,B6203D0C73406E74DB06795EB3FAC93F

jaKMGEhibe+VpZ9KpySPyv3NwQYafk/1D5+Fm7/Qi30yludRi3noPR0SR5n5nh0L
0zi1LFU1vP54I1XqaslRKi+FFUw3mT2px/JmjDGtYgjG0nFK1QGjzTjjtgUeKn+f
esRjgQeBqLZR4BWuGHaeo09wG6Fqj7CFWqTF4Ih7rpTwTKPqEdnDhmT2G7DJqY0J
GQ+DaHG9gzpYnHX8grgDHNq4cRhQxyWLzGTwFhJw6zSfhbQ8HUW28fzmi4P+UseM
Fd76wmAhcXE2z2oCjJG57z433eGdm9p3tJfsMgAyAuhAFQcdB1104cI0LHnUqLXs
D9wuIBwvzn/oqGQoBwFZvZb3TSjreF35rAJGuUp6+BDLDNcfHAdsr++DgBchxqmJ
I1L1Qba7rEJ3CYnQv93JzPDYibDBSu346lve+7E+NPzsnb1lxIMzTEL+Nh4EVBiy
BZ9PjmPA0Kew1f3xTCg/RLXSf1PewRgeUdu5aurWMGx7ARs2HQSSdxkv31w/kFjY
QzTc7Ts3yZX06znax2JzTn7raolpAL266QzX9oRZK3Y0IEbxb9Voaza6cgUYmrb5
jNb85ReAEN+sESFRz3GBzN2ZraANS0HutyA2YyhsP4o=
-----END RSA PRIVATE KEY-----
```

Figura 2.1: Fichero nombreRSApriv conteniendo la clave privada RSA

- 
3. Tarea 3: Extraed en nombreRSApub.pem la clave pública contenida en el archivo nombreRSAkey.pem. Mostrad sus valores.

Realizamos un paso equivalente pero esta vez le indicamos al comando que queremos extraer una clave pública con la opción **-pubout** ya que por defecto extrae la clave

privada. La salida del comando junto con la clave pública se muestra a continuación.

Comando: `openssl rsa -in nombreRSAkey.pem -outform PEM -pubout -out nombreRSAPub.pem`

```
[cvi075019:archivos Arthur18$ openssl rsa -in nombreRSAkey.pem -outform PEM -pubout -out nombreRSAPub.pem
writing RSA key
[cvi075019:archivos Arthur18$ cat nombreRSAPub.pem
-----BEGIN PUBLIC KEY-----
MHwwDQYJKoZIhvcNAQEBBQADAwAwAAJhA0sk+iq/YdWVfH+6BMuZTDLnI7X2AE+t
4pTlpmrdruIhyry9ZbAI7kEAR1/NhQ0PnGTSXs5WrPSaz/TsGywsovGMPcJyXYMG
30K1J9NCLu0Fo3IlPZEIlNaWVdWKgyThHwIDAQAB
-----END PUBLIC KEY-----
```

Figura 3.1: Fichero nombreRSAPub conteniendo la clave pública RSA

---

#### 4. Tarea 4-5: Reutilizar el fichero input.bin de 1024 bits de la práctica anterior e intentad cifrar input.bin con vuestras claves publicas. Explicad el resultado.

Comando: `openssl rsautl -encrypt -pubin -inkey nombreRSAPub.pem -in input.bin -out inputEnc.bin`

El resultado es la salida siguiente:

```
RSA operation error
140735739995016:error:0406D06E:rsa routines:RSA\_padding\_add\_PKCS1\_type\_2:data
too large for key size:rsa\_pk1.c:153:
```

Este error es la salida esperada al intentar cifrar un fichero de 1024 bits con una clave de 768. El algoritmo RSA, al ser un algoritmo de cifrado de flujo, solo contempla la posibilidad de cifrar un fichero cuyo tamaño sea como máximo el del tamaño de la clave, razón por la cual la salida del comando incluye la línea **data too large for key size**; RSA no se puede utilizar directamente con ficheros grandes, para ello se utiliza lo que se denomina un criptosistema híbrido, que veremos a continuación.

---

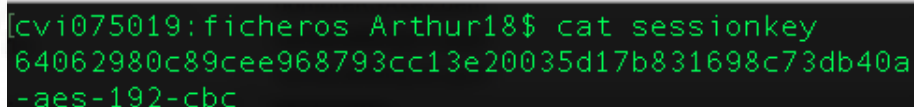
## 5. Tarea 6: Diseñad un cifrado híbrido, con RSA como criptosistema asimétrico. El modo de proceder será el siguiente

### 5.1. Tarea 6.1: El emisor debe seleccionar un sistema simétrico con su correspondiente modo de operación

El cifrado híbrido que diseñaremos hará uso de un cifrado simétrico **AES-192 en modo CBC**, por tanto nuestro fichero **sessionkey** tendrá en la segunda línea la cadena siguiente: **-aes-192-cbc**. Generamos el fichero **sessionkey** con el comando

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

La clave de AES en este caso es 192 bits (24 bytes), que son los que genera este comando; además en la segunda línea se especifica el criptosistema simétrico con su modo de operación. La salida del fichero **sessionKey** se ve a continuación.



```
[cvi075019:ficheros Arthur18$ cat sessionkey  
64062980c89cee968793cc13e20035d17b831698c73db40a  
-aes-192-cbc
```

Figura 5.1: Clave de sesión

### 5.2. Tarea 6.2: El archivo **sessionkey** se cifrará con la clave pública del receptor.

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

Utilizando este criptosistema híbrido simularemos la existencia de un emisor y receptor, aunque realmente sea la misma persona y se utilice un único par de claves pública-privada. El procedimiento a seguir será el siguiente:

1. **Paso 1:** Se cifrará el fichero **input.bin** con la clave y criptosistema simétrico contenido en **sessionkey**. En nuestro caso no utilizaremos vector de inicialización especificado de forma explícita dado que al especificar la clave con **-pass file:<fichero>**, el IV se genera a partir de la clave especificada en el fichero.

Comando: `openssl enc -aes-192-cbc -pass file:sessionkey -in input.bin -out mensajeEnc.bin`



La salida de este comando se ve a continuación:

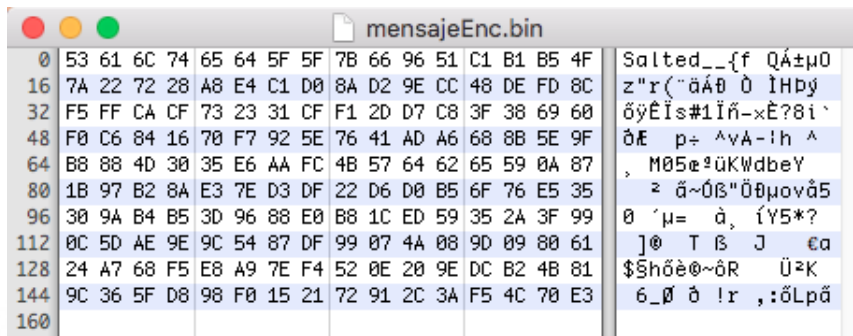


Figura 5.2: Fichero input.bin cifrado con AES-192.

- Paso 2:** El siguiente paso consiste en cifrar el fichero de sesión con la clave pública del criptosistema asimétrico, en nuestro caso RSA. Con el siguiente comando procederemos a cifrar el fichero de sesión con nuestra clave pública de RSA.

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

- Paso 3:** Tras simular el proceso de envío, se descifrará el fichero de sesión con la clave privada (en un caso real, sería una segunda persona quien descifraría con su clave privada) y posteriormente se utilizará la clave del criptosistema simétrico elegido (AES-192) contenido en este (el mismo que en el paso 1) para descifrar el mensaje y comprobar que la salida es la misma que **input.bin**

Primero desciframos la clave de sesión y comprobamos que es la misma que antes.

```
Comando: openssl rsautl -decrypt -inkey nombreRSAPriv.pem -in sessionKeyEnc
-out sessionKeyDec
```

Comprobamos que la salida del comando descifra el fichero de sesión original.

```
lcv1075019:ficheros Arthur18$ cat sessionKeyDec
64062980c89cee968793cc13e20035d17b831698c73db40a
-aes-192-cbc
```

Figura 5.3: Fichero de sesión descifrado utilizando la clave privada

Por último desciframos el mensaje con la clave de sesión descifrada en el paso anterior y comprobamos que es igual que el fichero input.bin

```
Comando: openssl aes-192-cbc -d -in mensajeEnc.bin -out mensajeDec.bin
-pass file:sessionKeyDec
```

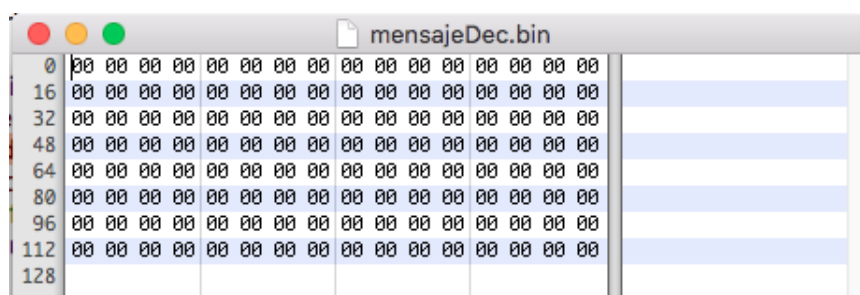


Figura 5.4: Mensaje descifrado es igual que el input.bin

Hasta aquí las tareas correspondientes a los criptosistemas asimétricos, en nuestro caso RSA, y un ejemplo real de comunicación en el que se hace uso de un criptosistema híbrido, en cuyo interior están integrados tanto un cifrado simétrico, en nuestro ejemplo AES-192-CBC, que se utiliza para cifrar el mensaje, y el cifrado asimétrico (RSA) que nos permite cifrar la clave de sesión con la que estableceremos una comunicación, a día de hoy, segura con una segunda persona. Pasamos con las tareas correspondientes a curvas elípticas.

- 
6. **Tarea 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 eleccion de las disponibles en OpenSSL. Mostrad los valores.**

En la documentación de openssl indica que para listar las curvas elípticas se utilice el comando

```
openssl ecparam -list_curves
```

Una de las curvas que aparece en las transparencias de teoría es la P-192 y siguiendo el ejemplo que aparece en la documentación referente a **openssl ecparam** para la generación de parámetros de las curvas elípticas, usaremos la curva **prime192v1**. Para generar los parámetros públicos de la curva elíptica elegida utilizamos el comando



Comando: `openssl ecparam -name prime192v1 -out stdECparam.pem`

La salida del comando se muestra a continuación.

```
MacBook-Pro-de-Arthur:archivos Arthur18$ cat stdECparam.pem
-----BEGIN EC PARAMETERS-----
BgqhkhjOPQMBAQ==
-----END EC PARAMETERS-----
```

Figura 6.1: Parámetros públicos de la curva elíptica P-192

El siguiente paso consistirá en realizar un procedimiento similar al anterior, en donde generaremos un par de claves pública-privada en otro fichero a partir de los parámetros de la curva elíptica que hemos elegido.

---

**6.1. Tarea 7.1: 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.**

Para generar un par de claves pública/privada a partir de los parámetros públicos de la EC P-192, utilizamos el siguiente comando:

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

donde **-genkey** se encargará de generar el par de claves que posteriormente separaremos. El fichero resultado se muestra a continuación:

```
MacBook-Pro-de-Arthur:archivos Arthur18$ cat nombreECkey.pem
-----BEGIN EC PRIVATE KEY-----
MF8CAQEGBefYNli3aCXgPyFasSo0g6mknrCbLwTG6AKBgqhkhjOPQMBAaE0AzIA
BJcfnRH5UjjS0ajHTpzlVV2E1nchBstDV0Lp5qYmcUSkne4HIyXjNDUflpue47jT
Pg==
-----END EC PRIVATE KEY-----
```

Figura 6.2: Generación de la clave con los parámetros públicos de la EC

## 6.2. Tarea 7.2: Extraer la clave privada contenida en el archivo nombreECkey.pem a otro archivo que tenga por nombre nombreECpriv.pem. Este archivo deber a estar protegido por contraseña cifrándolo con 3DES. Mostrad sus valores.

El comando para extraer la clave privada es similar al que se utilizaba con RSA: ahora especificamos que este fichero estará cifraado con un 3DES con la misma contraseña que antes, 0123456789.

Comando: `openssl ec -in nombreECkey.pem -out nombreECpriv.pem -outform PEM -des3`

Donde la salida obtenida es la siguiente:

```
[MacBook-Pro-de-Arthur:archivos Arthur18$ cat nombreECpriv.pem
-----BEGIN EC PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,33190E43FE43623B

0UeN6Cmdek/HQCXEuzdcTFma35oezjyIrF6F9C+6e94E+k4XEWXBRBqP4WaNpU1J
X9DA/03FApWIqkYcy/r2f/B0SGCm0mB5M8HAGrDtUvBfy0hhJX33n9Up70I3U+v6
Q50UMNDFeZk=
-----END EC PRIVATE KEY-----
```

Figura 6.3: Clave privada generada con la EC

## 6.3. Tarea 7.3: Extraer en nombreECpub.pem la clave publica contenida en el archivo nombreECkey.pem

Por último se nos pide extraer la clave pública que será utilizada por otros usuarios para enviarnos mensajes cifrados. El comando para extraer la clave pública se asemeja al utilizado para RSA:

Comando: `openssl ec -in nombreECkey.pem -pubout -out nombreECpub.pem`

Donde la salida obtenida es la siguiente:

```
[MacBook-Pro-de-Arthur:archivos Arthur18$ cat nombreECpub.pem
-----BEGIN PUBLIC KEY-----
MEkwEwYHKOZIZj0CAQYIKoZIZj0DAQEDMgAE1x+dEf1S0NI5qMd0n0VVXYTWdyEG
y0NXQunmpiZxRKsd7gcjJeM0NR+Wm57juNM+
-----END PUBLIC KEY-----
```

Figura 6.4: Clave publica generada con la EC