

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

# Práctica 3: OpenSSL y Criptografía Pública

## 1. OpenSSL

### A. Cifrados simétricos

Ejecutamos el comando openssl help y obtenemos lo siguiente:

```
laura@Universidad:~$ openssl help
openssl:Error: 'help' is an invalid command.

Standard commands
asn1parse      ca          ciphers       cms
crl           crl2pkcs7   dgst         dh
dhparam        dsa          dsaparam     ec
ecparam        enc          engine       errstr
gendh          gendsa      genpkey     genrsa
nseq           ocsp         passwd      pkcs12
pkcs7          pkcs8        pkey        pkeyparam
pkeyutl        prime       rand         req
rsa            rsautl      s_client    s_server
s_time         sess_id     smime       speed
spkac          srp          ts          verify
version        x509

Message Digest commands (see the 'dgst' command for more details)
md4            md5          rmd160      sha
sha1

Cipher commands (see the 'enc' command for more details)
aes-128-cbc   aes-128-ecb   aes-192-cbc   aes-192-ecb
aes-256-cbc   aes-256-ecb   base64       bf
bf-cbc         bf-cfb      bf-ecb      bf-ofb
camellia-128-cbc camellia-128-ecb camellia-192-cbc camellia-192-ecb
camellia-256-cbc camellia-256-ecb cast         cast-cbc
cast5-cbc      casts-cfb   cast5-ecb   cast5-ofb
des            des-cbc     des-cfb     des-ecb
des-edc        des-edc-cbc  des-edc-ecb  des-edc-ofb
des-ed3        des-ed3-cbc  des-ed3-ecb  des-ed3-ofb
des-ofb        des3         desx        rc2
rc2-40-cbc    rc2-64-cbc   rc2-cbc    rc2-cfb
rc2-ecb        rc2-ofb     rc4         rc4-40
seed           seed-cbc    seed-cfb   seed-ecb
seed-ofb       seed-ofb
```

Podemos observar que OpenSSL ofrece una gran variedad de comandos. En Cipher Commands aparece una serie de algoritmos de cifrado, de los cuales los simétricos son: des y aes, con sus múltiples versiones; y otros que no hemos dado en teoría como rc, camellia, seed, bf, y cast (con varias versiones también).

Cabe añadir que en Message Digest aparecen algunas de las funciones Hash y MAC más importantes, como el sha.

Vamos a hablar resumidamente de los 7 nombrados anteriormente:

DES: Cifrado simétrico por bloques de 64 bits con una clave de 56 bits (modo simple). Consiste en pasar el bloque por 16 rondas Feistel.

AES: Cifrado simétrico por bloques de 128 bits y la clave puede ser de 128, 192 o 256 bits. Consiste en operar sobre una matriz state (bloque), realizando distintas operaciones como desplazamientos, sustituciones, etc.

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

**RC4:** Cifrado de flujo (no basado en bloques) simétrico. Parte de una clave secreta compartida por emisor y receptor, ambos generan un flujo de cifrado pseudoaleatorio (keystream). Este flujo de cifrado se mezcla con los datos que se van a transmitir mediante una operación de combinación, habitualmente XOR.

**CAMELLIA:** Cifrado simétrico por bloques con un tamaño de bloque de 128 bits y tamaños de clave de 128, 192 y 256 bits. Consiste en pasar el bloque por 18 rondas Feistel (cuando se usan claves de 128 bits) o 24 rondas Feistel (cuando se usan claves de 192 o 256 bits), y cada seis rondas se aplica una transformación lógica, llamada "función FL" o su inversa. Además, utiliza cuatro cajas S de  $8 \times 8$  bits con transformaciones afines y operaciones lógicas; el cifrado también utiliza blanqueamiento de claves de entrada y salida; y la capa de difusión utiliza una transformación lineal basada en una matriz.

**SEED:** Cifrado simétrico por bloques de 128 bits y claves de 128 bits. Consiste en pasar el bloque por 16 rondas Feistel, utilizando dos cajas S de  $8 \times 8$  que se derivan de una exponenciación discreta. Es importante que posee recursividad en su estructura, ya que el cifrado completo de 128 bits es una red Feistel con una función F que opera en mitades de 64 bits, mientras que la función F en sí es una red Feistel compuesta por una función G que opera en mitades de 32 bits. Sin embargo, la recursividad no se extiende más porque la función G no es una red de Feistel. En la función G, los bytes pasan a través de una de las cajas S, luego se combina en un conjunto complejo de funciones booleanas, donde cada bit de salida depende de 3 de los 4 bytes de entrada.

SEED tiene un programa de claves bastante complejo, generando sus treinta y dos subclaves de 32 bits mediante la aplicación de su función G en una serie de rotaciones de la clave sin procesar, combinadas con constantes redondas derivadas (como en TEA) de la proporción áurea.

**BF(Blowfish):** Cifrado simétrico por bloques de 64 bits y tamaños de clave desde los 32 hasta los 448 bits. Consiste en pasar el bloque por 16 rondas Feistel, aplicando llaves que dependen de las cajas de sustitución.

**CAST:** Cifrado simétrico por bloques de 64 bits y tamaños de clave entre 40 y 128 bits. Consiste en pasar el bloque por 12 o 16 rondas Feistel, con cajas de sustitución de  $8 \times 32$  bits basadas en funciones bent. Además, aplica rotaciones que dependen de la clave, sumas y restas modulares y operaciones XOR.

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

Ejecutamos el comando openssl enc -h y obtenemos lo siguiente:

```
Universidad: ~
laura@Universidad:~$ openssl enc -h
unknown option '-h'
options are
-in <file>      input file
-out <file>      output file
-pass <arg>      pass phrase source
-e              encrypt
-d              decrypt
-a/-base64     base64 encode/decode, depending on encryption flag
-k              passphrase is the next argument
-kfile         passphrase is the first line of the file argument
-md              the next argument is the md to use to create a key
                  from a passphrase. One of md2, md5, sha or sha1
-S              salt in hex is the next argument
-K/-iv          key/iv in hex is the next argument
-[pP]           print the iv/key (then exit if -P)
-bufsize <n>    buffer size
-nopad          disable standard block padding
-engine e       use engine e, possibly a hardware device.
Cipher Types
-aes-128-cbc      -aes-128-cbc-hmac-sha1      -aes-128-cbc-hmac-sha256
-aes-128-ccm      -aes-128-cfb      -aes-128-cfb1
-aes-128-cfb8     -aes-128-ctr      -aes-128-ecb
-aes-128-gcm      -aes-128-ofb      -aes-128-xts
-aes-192-cbc      -aes-192-ccm      -aes-192-cfb
-aes-192-cfb1     -aes-192-cfb8     -aes-192-ctr
-aes-192-ecb      -aes-192-gcm      -aes-192-ofb
-aes-256-cbc      -aes-256-cbc-hmac-sha1      -aes-256-cbc-hmac-sha256
-aes-256-ccm      -aes-256-cfb      -aes-256-cfb1
-aes-256-cfb8     -aes-256-ctr      -aes-256-ecb
-aes-256-gcm      -aes-256-ofb      -aes-256-xts
-aes128          -aes192          -aes256
-bf              -bf-cbc          -bf-cfb
-bf-ecb          -bf-ofb          -blowfish
-camellia-128-cbc -camellia-128-cfb      -camellia-128-cfb1
-camellia-128-ccm -camellia-128-ecb      -camellia-128-ofb
-camellia-192-cbc -camellia-192-cfb      -camellia-192-cfb1
-camellia-192-cfb8 -camellia-192-ecb      -camellia-192-ofb
-camellia-256-cbc -camellia-256-cfb      -camellia-256-cfb1
-camellia-256-cfb8 -camellia-256-ecb      -camellia-256-ofb
-camellia128      -camellia192      -camellia256
-cast            -cast-cbc          -cast5-cbc
-cast5-cfb        -cast5-ecb          -cast5-ofb
-des             -des-cbc          -des-cfb
-des-cfb1        -des-cfb8         -des-ecb
-des-edede       -des-edede-cbc      -des-edede-cfb
-des-edede-ofb   -des-edede3        -des-edede3-cbc
-des-edede3-cfb  -des-edede3-cfb1      -des-edede3-cfb8
-des-edede3-ofb  -des-ofb          -des3
-desx            -desx-cbc          -id-aes128-CCM
-id-aes128-GCM   -id-aes128-wrap      -id-aes192-CCM
-id-aes192-GCM   -id-aes192-wrap      -id-aes256-CCM
-id-aes256-GCM   -id-aes256-wrap      -id-smime-alg-CMS3DESWrap
-rc2             -rc2-40-cbc        -rc2-64-cbc
                                         -rc2-cfb          -rc2-ecb
                                         -rc2-ofb          -rc4              -rc4-40
                                         -rc4-hmac-md5      -seed            -seed-cbc
                                         -seed-cfb          -seed-ecb          -seed-ofb
laura@Universidad:~$
```

En la imagen anterior, podemos ver que se muestra ayuda adicional sobre los posibles argumentos de entrada de los cifradores, como por ejemplo -e para cifrar, -d para descifrar, -K/-iv para la clave y el vector de inicialización, etc...

Observamos una ayuda más detallada, los algoritmos tienen más versiones y están mejor estructurados.

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

Tras ejecutar el comando openssl ciphers -v obtenemos:

```
laura@Universidad:~$ openssl ciphers -v
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=RSA Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=RSA Enc=AES(256) Mac=SHA384
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(256) Mac=SHA384
ECDHE-RSA-AES256-SHA SSLv3 Kx=ECDH     Au=RSA Enc=AES(256) Mac=SHA1
ECDHE-ECDSA-AES256-SHA SSLv3 Kx=ECDH     Au=ECDSA Enc=AES(256) Mac=SHA1
SRP-DSS-AES-256-CBC-SHA SSLv3 Kx=SRP     Au=DSS Enc=AES(256) Mac=SHA1
SRP-RSA-AES-256-CBC-SHA SSLv3 Kx=SRP     Au=RSA Enc=AES(256) Mac=SHA1
SRP-AES-256-CBC-SHA SSLv3 Kx=SRP     Au=SRP Enc=AES(256) Mac=SHA1
DHE-DSS-AES256-GCM-SHA384 TLSv1.2 Kx=DH     Au=DSS Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH     Au=RSA Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-SHA256 TLSv1.2 Kx=DH     Au=RSA Enc=AES(256) Mac=SHA256
DHE-DSS-AES256-SHA256 TLSv1.2 Kx=DH     Au=DSS Enc=AES(256) Mac=SHA256
DHE-RSA-AES256-SHA SSLv3 Kx=DH     Au=RSA Enc=AES(256) Mac=SHA1
DHE-DSS-AES256-SHA SSLv3 Kx=DH     Au=DSS Enc=AES(256) Mac=SHA1
DHE-RSA-CAMELLIA256-SHA SSLv3 Kx=DH     Au=RSA Enc=Camellia(256) Mac=SHA1
DHE-DSS-CAMELLIA256-SHA SSLv3 Kx=DH     Au=DSS Enc=Camellia(256) Mac=SHA1
ECDH-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH/RSA Au=ECDH Enc=AESGCM(256) Mac=AEAD
ECDH-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH/ECDSA Au=ECDH Enc=AESGCM(256) Mac=AEAD
ECDH-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH/RSA Au=ECDH Enc=AES(256) Mac=SHA384
ECDH-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH/ECDSA Au=ECDH Enc=AES(256) Mac=SHA384
ECDH-RSA-AES256-SHA SSLv3 Kx=ECDH/RSA Au=ECDH Enc=AES(256) Mac=SHA1
ECDH-ECDSA-AES256-SHA SSLv3 Kx=ECDH/ECDSA Au=ECDH Enc=AES(256) Mac=SHA1
AES256-GCM-SHA384 TLSv1.2 Kx=RSA     Au=RSA Enc=AESGCM(256) Mac=AEAD
AES256-SHA256 TLSv1.2 Kx=RSA     Au=RSA Enc=AES(256) Mac=SHA256
AES256-SHA SSLv3 Kx=RSA     Au=RSA Enc=AES(256) Mac=SHA1
CAMELLIA256-SHA SSLv3 Kx=RSA     Au=RSA Enc=Camellia(256) Mac=SHA1
PSK-AES256-CBC-SHA SSLv3 Kx=PSK     Au=PSK Enc=AES(256) Mac=SHA1
ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=RSA Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=RSA Enc=AES(128) Mac=SHA256
ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(128) Mac=SHA256
ECDHE-RSA-AES128-SHA SSLv3 Kx=ECDH     Au=RSA Enc=AES(128) Mac=SHA1
ECDHE-ECDSA-AES128-SHA SSLv3 Kx=ECDH     Au=ECDSA Enc=AES(128) Mac=SHA1
SRP-DSS-AES-128-CBC-SHA SSLv3 Kx=SRP     Au=DSS Enc=AES(128) Mac=SHA1
SRP-RSA-AES-128-CBC-SHA SSLv3 Kx=SRP     Au=RSA Enc=AES(128) Mac=SHA1
SRP-AES-128-CBC-SHA SSLv3 Kx=SRP     Au=SRP Enc=AES(128) Mac=SHA1
DHE-DSS-AES128-GCM-SHA256 TLSv1.2 Kx=DH     Au=DSS Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=DH     Au=RSA Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-SHA256 TLSv1.2 Kx=DH     Au=RSA Enc=AES(128) Mac=SHA256
DHE-DSS-AES128-SHA256 TLSv1.2 Kx=DH     Au=DSS Enc=AES(128) Mac=SHA256
DHE-RSA-AES128-SHA SSLv3 Kx=DH     Au=RSA Enc=AES(128) Mac=SHA1
DHE-DSS-AES128-SHA SSLv3 Kx=DH     Au=DSS Enc=AES(128) Mac=SHA1
DHE-RSA-SEED-SHA SSLv3 Kx=DH     Au=RSA Enc=SEED(128) Mac=SHA1
DHE-DSS-SEED-SHA SSLv3 Kx=DH     Au=DSS Enc=SEED(128) Mac=SHA1
DHE-RSA-CAMELLIA128-SHA SSLv3 Kx=DH     Au=RSA Enc=Camellia(128) Mac=SHA1
DHE-DSS-CAMELLIA128-SHA SSLv3 Kx=DH     Au=DSS Enc=Camellia(128) Mac=SHA1
ECDH-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH/RSA Au=ECDH Enc=AESGCM(128) Mac=AEAD
ECDH-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH/ECDSA Au=ECDH Enc=AESGCM(128) Mac=AEAD
ECDH-RSA-AES128-SHA SSLv3 Kx=RSA     Au=RSA Enc=AES(128) Mac=SHA1
ECDH-ECDSA-AES128-SHA SSLv3 Kx=RSA     Au=ECDSA Enc=AES(128) Mac=SHA1
AES128-GCM-SHA256 TLSv1.2 Kx=RSA     Au=RSA Enc=AESGCM(128) Mac=AEAD
AES128-SHA256 TLSv1.2 Kx=RSA     Au=RSA Enc=AES(128) Mac=SHA256
AES128-SHA SSLv3 Kx=RSA     Au=RSA Enc=AES(128) Mac=SHA1
SEED-SHA SSLv3 Kx=RSA     Au=RSA Enc=SEED(128) Mac=SHA1
CAMELLIA128-SHA SSLv3 Kx=RSA     Au=RSA Enc=Camellia(128) Mac=SHA1
PSK-AES128-CBC-SHA SSLv3 Kx=PSK     Au=PSK Enc=AES(128) Mac=SHA1
ECDHE-RSA-RC4-SHA SSLv3 Kx=ECDH     Au=ECDSA Enc=RC4(128) Mac=SHA1
ECDHE-ECDSA-RC4-SHA SSLv3 Kx=ECDH     Au=ECDSA Enc=RC4(128) Mac=SHA1
ECDH-RSA-RC4-SHA SSLv3 Kx=ECDH/RSA Au=ECDH Enc=RC4(128) Mac=SHA1
ECDH-ECDSA-RC4-SHA SSLv3 Kx=ECDH/ECDSA Au=ECDH Enc=RC4(128) Mac=SHA1
RC4-SHA SSLv3 Kx=RSA     Au=RSA Enc=RC4(128) Mac=SHA1
RC4-MD5 SSLv3 Kx=RSA     Au=RSA Enc=RC4(128) Mac=MD5
PSK-RC4-SHA SSLv3 Kx=PSK     Au=PSK Enc=RC4(128) Mac=SHA1
ECDHE-RSA-DES-CBC3-SHA SSLv3 Kx=ECDH     Au=RSA Enc=3DES(168) Mac=SHA1
ECDHE-ECDSA-DES-CBC3-SHA SSLv3 Kx=ECDH     Au=ECDSA Enc=3DES(168) Mac=SHA1
SRP-DSS-3DES-EDE-CBC-SHA SSLv3 Kx=SRP     Au=DSS Enc=3DES(168) Mac=SHA1
SRP-RSA-3DES-EDE-CBC-SHA SSLv3 Kx=SRP     Au=RSA Enc=3DES(168) Mac=SHA1
SRP-3DES-EDE-CBC-SHA SSLv3 Kx=SRP     Au=SRP Enc=3DES(168) Mac=SHA1
EDH-RSA-DES-CBC3-SHA SSLv3 Kx=DH     Au=RSA Enc=3DES(168) Mac=SHA1
EDH-DSS-DES-CBC3-SHA SSLv3 Kx=DH     Au=DSS Enc=3DES(168) Mac=SHA1
ECDH-RSA-DES-CBC3-SHA SSLv3 Kx=ECDH/RSA Au=ECDH Enc=3DES(168) Mac=SHA1
ECDH-ECDSA-DES-CBC3-SHA SSLv3 Kx=ECDH/ECDSA Au=ECDH Enc=3DES(168) Mac=SHA1
DES-CBC3-SHA SSLv3 Kx=RSA     Au=RSA Enc=3DES(168) Mac=SHA1
PSK-3DES-EDE-CBC-SHA SSLv3 Kx=PSK     Au=PSK Enc=3DES(168) Mac=SHA1
laura@Universidad:~$
```

Obtenemos los cifrados soportados por la herramienta OpenSSL, están tanto los cifrado simétricos como los asimétricos.

A continuación vamos a probar algunos de ellos:

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

DES:

Probamos el des:

```
2$ openssl des -e -K 1122334455667788 -iv 1010101010101010 -in don_quixote.txt -out cifrado.txt  
2$ openssl des -d -K 1122334455667788 -iv 1010101010101010 -in cifrado.txt -out descifrado.txt  
2$
```

Comprobamos el resultado del descifrado:

```
2$ diff descifrado.txt don_quixote.txt  
2$
```

Claves débiles que tiene el DES:

- **0101010101010101h**
- 1F1F1F1F0E0E0E0Eh
- E0E0E0E0F1F1F1F1h
- FEEFEFEFEFEFEFEh

Comprobamos que es cierto que la clave débil cumple que  $E_k(E_k(x)) = x$ , por lo tanto probamos a cifrar el texto cifrado con la misma clave débil:

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ diff don_quixote.txt descifrado-cd.txt  
208a209  
> [REDACTED]B("w6f♦  
\ No hay ningún carácter de nueva línea al final del archivo  
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```

Sí se cumple, es decir, se ha obtenido el texto original, pero de la línea 208 al 209 posee bytes basura. Y al final del archivo no hay carácter de nueva línea.

Claves semidébiles que tiene el DES:

- **01FE01FE01FE01FEh**
- **FE01FE01FE01FE01h**
- 1FE01FE00EF10EF1h
- E01FE01FF10EF10Eh
- ...

Comprobamos que es cierto que la clave semidébil cumple que  $E_k(E_k(x)) = x$ , por lo tanto probamos a cifrar el texto cifrado con la primera clave y luego cifrar de nuevo con la segunda:

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ diff desdes.txt don_quixote.txt  
$ openssl des-ecb -e -K 01FE01FE01FE01FE -in don_quixote.txt -out cifrado-cd  
$ openssl des-ecb -e -K FE01FE01FE01FE01 -in cifrado-cd -out des-csd.txt  
$
```

Laura Sánchez Herrera

# Daniel Mateo Moreno

## Pareja: 5

## Comprobamos el resultado:

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ diff des-csd.txt don_quixote.txt  
209d208  
< \n\n\n*d#9+5#  
\ No hay ningún carácter de nueva línea al final del archivo  
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```

De nuevo, se cumple, se ha obtenido el texto original, pero de la línea 208 al 209 posee bytes basura. Y al final del archivo no hay carácter de nueva línea.

AES:

## Probamos el aes-128-cb:

Comprobamos el resultado del descifrado:

```
2$ diff desaes.txt don_quixote.txt  
2$
```

## B. Cifrados asimétricos

Los cifrados asimétricos que soporta OpenSSL los podemos observar en la salida del comando `openssl ciphers -v`, hemos capturado su salida en el apartado a).

Podemos encontrar el cifrado asimétrico RSA, con el que hemos hecho una prueba de cifrado y de descifrado.

Para esto, necesitamos una clave privada y otra pública, que hemos generado y explicamos en el siguiente apartado. Entonces, ciframos un archivo con esta clave pública, simulando que alguien ajeno nos está enviando un archivo cifrado con nuestra clave pública. Y, tras esto, procedemos a descifrar ese archivo cifrado aplicando nuestra clave privada, que solo la conocemos nosotros, obteniendo el mensaje original.

Probamos su funcionamiento, pero primero con un archivo grande que supera el tamaño de la clave:

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ openssl rsa -encrypt -pubin -inkey pub-key.pem -in don_quixote.txt -out cifrado.enc
RSA operation error
14018923325720:error:04060D6E:rsa routines:RSA_padding_add_PKCS1_type_2:data too large for key size:rsa_pk1.c:153:
laura@Universidad:~/Escritorio/ CUARTO/cripto/P2$
```

Se observa que se muestra un error al intentar cifrar el archivo, y es debido a que la clave no es lo suficientemente grande para cifrar el archivo por completo.

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

Ciframos y desciframos un archivo más pequeño:

```
$ openssl rsautl -encrypt -pubin -inkey pub-key.pem -in texto.txt -out cifrado.enc  
$ openssl rsautl -decrypt -inkey key.pem -in cifrado.enc -out descifradora.txt
```

Comprobamos su funcionalidad:

```
$ diff texto.txt descifradora.txt  
$
```

### C. Generación de claves privadas y públicas

El esquema de cifrado público basado en RSA es válido tanto para cifrar como para firmar digitalmente. La seguridad de este algoritmo radica en el problema de la factorización de números enteros grandes. El funcionamiento se basa en la generación de dos números primos muy grandes, p y q, que con su producto se obtiene el módulo n. Con esto, ya se puede aplicar el algoritmo de generación de un par de claves para este cifrado. Este consiste en obtener un número exponente e y su inverso d en módulo  $\phi(n)$ , obteniendo la clave pública (e, n) y la clave privada (d, n).

En este apartado nos centramos en la generación de este par de claves.

En un esquema real, una entidad genera una clave privada para nosotros y que debemos guardar con cautela. A partir de ella, también se genera la clave pública que todo el mundo podrá ver. Esta clave pública la utilizarán para cifrar sus mensajes que vayan dirigidos a nosotros, para que nosotros seamos los únicos que podamos descifrar el mensaje, aplicando nuestra clave privada (que descifra el mensaje cifrado).

A continuación, generamos la clave privada y la clave pública, de tamaño 1024 bits de módulo n:

**Clave privada:**

```
openssl genrsa -out key.pem 1024
```

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ openssl genrsa -out key.pem 1024  
Generating RSA private key, 1024 bit long modulus  
.....+++++  
.....+++++  
e is 65537 (0x10001)  
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ cat key.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDs+jp7fkSEKSRy6l9Qnp2i7NF0vz5wcaWrhS1SIgCPw1ikATSN
+2wclVqHMzllrW8PkC/ab50K43lltMKZwZfd574Euz22qsbm+g5t0AVFco/TJQmi
WV2GJ4idvIEuQEgt1UHhm4LC45BqYICNGzwKk39pVhClbB4UJF+b3AyQrwIDAQAB
AoGAAnZjLy9Tt0/cghkvXl4Xkz7UufXAx1hW+V5L500VL+k+vY2Jp0QKjo16wFbm
7MiNk+LW/xUT064jb8weDYz2b7yVSJEKWSaQqu7bvSkP03uJYgB+X8uY+Ud/II5B
XAKVJGrryW29bXgtqQCYS1BcmjTTyPprmD8MmVolbL7P6/kCQQD3+xUYksxwE0lw
FwACkKiSOCIVdmQLqMot4v0HpBU16R1iSOaGwhFsGzFWTdE+lj+A4V1hGWUmB5Cv
lxAKEHCLAKEA9KQN9oeBwbLxw9oZFDokSydfDj6Nof+WlR/HoLmpLyxQtQ6nDaoP
N+ELFP1jRraa+1B0Co2kVw/7SCR/s6kg7QJBAKIutndh2w8Zrhf/EuSak4U97S25
cmklUUZhX8v0woG3E7hetN7rUVU64mUT1ttjI7P1fChWUPsAra0R6stYU0CQDxH
P/mCogItwzSypINLYC30YYOM/jDwXRYZYoLDvHm3PZufJKJFSMKn/KcLWC70rZdg
zjPjep7JHU1FUQHfpnECQFqLTP3LkNdbLW16t8/T/xL3fXEQqZvgakwt0ck+i8eD
NBeZJ7buYbMODuRCcFU838FsQjVGrgG1TDQKPN51EL4=
-----END RSA PRIVATE KEY-----
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```

**Clave pública:**

```
openssl rsa -in key.pem -pubout -out pub-key.pem
```

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ openssl rsa -in key.pem -pubout -out pub-key.pem
writing RSA key
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```

```
-----END RSA PRIVATE KEY-----
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ cat pub-key.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDs+jp7fkSEKSRy6l9Qnp2i7NF0
vz5wcaWrhS1SIgCPw1ikATSN+2wclVqHMzllrW8PkC/ab50K43lltMKZwZfd574E
uz22qsbm+g5t0AVFco/TJQmiWV2GJ4idvIEuQEgt1UHhm4LC45BqYICNGzwKk39p
VhClbB4UJF+b3AyQrwIDAQAB
-----END PUBLIC KEY-----
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```

## D. Diferencias entre velocidades de cifrados simétricos y asimétricos

Ejecutamos en la terminal el comando openssl speed, que ejecuta todos los cifradores y muestra cuántas veces cifran un bloque (con su tamaño correspondiente) en el tiempo indicado (3 o 10 segundos). Se puede ver en las capturas:

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

```
laura@Universidad:~$ openssl speed
Doing md4 for 3s on 16 size blocks: 11377882 md4's in 2.96s
Doing md4 for 3s on 64 size blocks: 9947960 md4's in 2.96s
Doing md4 for 3s on 256 size blocks: 6916101 md4's in 2.98s
Doing md4 for 3s on 1024 size blocks: 2800462 md4's in 2.92s
Doing md4 for 3s on 8192 size blocks: 4266669 md4's in 2.92s
Doing md5 for 3s on 16 size blocks: 8627345 md5's in 2.94s
Doing md5 for 3s on 64 size blocks: 6878400 md5's in 2.88s
Doing md5 for 3s on 256 size blocks: 4432393 md5's in 2.95s
Doing md5 for 3s on 1024 size blocks: 1611666 md5's in 2.96s
Doing md5 for 3s on 8192 size blocks: 168917 md5's in 2.80s
Doing hmac(md5) for 3s on 16 size blocks: 6623416 hmac(md5)'s in 2.96s
Doing hmac(md5) for 3s on 64 size blocks: 6082750 hmac(md5)'s in 2.96s
Doing hmac(md5) for 3s on 256 size blocks: 4051955 hmac(md5)'s in 2.96s
Doing hmac(md5) for 3s on 1024 size blocks: 1616727 hmac(md5)'s in 2.95s
Doing hmac(md5) for 3s on 8192 size blocks: 246972 hmac(md5)'s in 2.97s
Doing sha1 for 3s on 16 size blocks: 9275686 sha1's in 2.91s
Doing sha1 for 3s on 64 size blocks: 7013484 sha1's in 2.84s
Doing sha1 for 3s on 256 size blocks: 4359101 sha1's in 2.80s
Doing sha1 for 3s on 1024 size blocks: 1402229 sha1's in 2.51s
Doing sha1 for 3s on 8192 size blocks: 248853 sha1's in 2.74s
Doing sha256 for 3s on 16 size blocks: 11219299 sha256's in 2.83s
Doing sha256 for 3s on 64 size blocks: 6622292 sha256's in 2.86s
Doing sha256 for 3s on 256 size blocks: 2980201 sha256's in 2.86s
Doing sha256 for 3s on 1024 size blocks: 1018389 sha256's in 2.91s
Doing sha256 for 3s on 8192 size blocks: 128012 sha256's in 2.71s
Doing sha512 for 3s on 16 size blocks: 9255980 sha512's in 2.93s
Doing sha512 for 3s on 64 size blocks: 9750056 sha512's in 2.97s
Doing sha512 for 3s on 256 size blocks: 3954721 sha512's in 2.97s
Doing sha512 for 3s on 1024 size blocks: 1400416 sha512's in 2.95s
Doing sha512 for 3s on 8192 size blocks: 203910 sha512's in 2.95s
Doing whirlpool for 3s on 16 size blocks: 6685672 whirlpool's in 2.97s
Doing whirlpool for 3s on 64 size blocks: 3551117 whirlpool's in 2.96s
Doing whirlpool for 3s on 256 size blocks: 1502399 whirlpool's in 2.86s
Doing whirlpool for 3s on 1024 size blocks: 401731 whirlpool's in 2.79s
Doing whirlpool for 3s on 8192 size blocks: 59780 whirlpool's in 2.92s
Doing rmd160 for 3s on 16 size blocks: 6553612 rmd160's in 2.96s
Doing rmd160 for 3s on 64 size blocks: 4298170 rmd160's in 2.89s
Doing rmd160 for 3s on 256 size blocks: 2097183 rmd160's in 2.94s
Doing rmd160 for 3s on 1024 size blocks: 695209 rmd160's in 2.90s
Doing rmd160 for 3s on 8192 size blocks: 93232 rmd160's in 2.86s
Doing rc4 for 3s on 16 size blocks: 99064009 rc4's in 2.96s
Doing rc4 for 3s on 64 size blocks: 31174225 rc4's in 2.99s
Doing rc4 for 3s on 256 size blocks: 7019428 rc4's in 2.98s
Doing rc4 for 3s on 1024 size blocks: 1777681 rc4's in 3.00s
Doing rc4 for 3s on 8192 size blocks: 206697 rc4's in 2.99s
Doing des cbc for 3s on 16 size blocks: 14790651 des cbc's in 2.98s
Doing des cbc for 3s on 64 size blocks: 3775100 des cbc's in 2.90s
Doing des cbc for 3s on 256 size blocks: 965237 des cbc's in 2.97s
Doing des cbc for 3s on 1024 size blocks: 245695 des cbc's in 2.90s
Doing des cbc for 3s on 8192 size blocks: 31041 des cbc's in 2.95s
Doing des ede3 for 3s on 16 size blocks: 5811932 des ede3's in 2.96s
Doing des ede3 for 3s on 64 size blocks: 1444566 des ede3's in 2.97s
Doing des ede3 for 3s on 256 size blocks: 358039 des ede3's in 2.96s
```

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

```
Doing des ede3 for 3s on 1024 size blocks: 90330 des ede3's in 2.98s
Doing des ede3 for 3s on 8192 size blocks: 11381 des ede3's in 2.97s
Doing aes-128 cbc for 3s on 16 size blocks: 24775840 aes-128 cbc's in 2.94s
Doing aes-128 cbc for 3s on 64 size blocks: 6823907 aes-128 cbc's in 2.84s
Doing aes-128 cbc for 3s on 256 size blocks: 1630705 aes-128 cbc's in 2.78s
Doing aes-128 cbc for 3s on 1024 size blocks: 932045 aes-128 cbc's in 2.90s
Doing aes-128 cbc for 3s on 8192 size blocks: 97141 aes-128 cbc's in 2.62s
Doing aes-192 cbc for 3s on 16 size blocks: 16308702 aes-192 cbc's in 2.66s
Doing aes-192 cbc for 3s on 64 size blocks: 5291950 aes-192 cbc's in 2.83s
Doing aes-192 cbc for 3s on 256 size blocks: 1292667 aes-192 cbc's in 2.80s
Doing aes-192 cbc for 3s on 1024 size blocks: 723442 aes-192 cbc's in 2.82s
Doing aes-192 cbc for 3s on 8192 size blocks: 96996 aes-192 cbc's in 2.79s
Doing aes-256 cbc for 3s on 16 size blocks: 17539481 aes-256 cbc's in 2.93s
Doing aes-256 cbc for 3s on 64 size blocks: 5286820 aes-256 cbc's in 2.98s
Doing aes-256 cbc for 3s on 256 size blocks: 1334695 aes-256 cbc's in 2.98s
Doing aes-256 cbc for 3s on 1024 size blocks: 739294 aes-256 cbc's in 2.98s
Doing aes-256 cbc for 3s on 8192 size blocks: 92944 aes-256 cbc's in 2.98s
Doing aes-128 ige for 3s on 16 size blocks: 24951779 aes-128 ige's in 2.95s
Doing aes-128 ige for 3s on 64 size blocks: 6978209 aes-128 ige's in 2.97s
Doing aes-128 ige for 3s on 256 size blocks: 1719662 aes-128 ige's in 2.84s
Doing aes-128 ige for 3s on 1024 size blocks: 422025 aes-128 ige's in 2.78s
Doing aes-128 ige for 3s on 8192 size blocks: 48458 aes-128 ige's in 2.66s
Doing aes-192 ige for 3s on 16 size blocks: 17710320 aes-192 ige's in 2.79s
Doing aes-192 ige for 3s on 64 size blocks: 5030949 aes-192 ige's in 2.84s
Doing aes-192 ige for 3s on 256 size blocks: 1320897 aes-192 ige's in 2.81s
Doing aes-192 ige for 3s on 1024 size blocks: 364516 aes-192 ige's in 2.98s
Doing aes-192 ige for 3s on 8192 size blocks: 46764 aes-192 ige's in 2.96s
Doing aes-256 ige for 3s on 16 size blocks: 17974662 aes-256 ige's in 2.95s
Doing aes-256 ige for 3s on 64 size blocks: 4530499 aes-256 ige's in 2.88s
Doing aes-256 ige for 3s on 256 size blocks: 1223919 aes-256 ige's in 2.88s
Doing aes-256 ige for 3s on 1024 size blocks: 294577 aes-256 ige's in 2.91s
Doing aes-256 ige for 3s on 8192 size blocks: 36817 aes-256 ige's in 2.85s
Doing ghash for 3s on 16 size blocks: 111924878 ghash's in 2.98s
Doing ghash for 3s on 64 size blocks: 105760463 ghash's in 2.97s
Doing ghash for 3s on 256 size blocks: 64204339 ghash's in 2.81s
Doing ghash for 3s on 1024 size blocks: 21798483 ghash's in 2.82s
Doing ghash for 3s on 8192 size blocks: 3265871 ghash's in 2.81s
Doing camellia-128 cbc for 3s on 16 size blocks: 20449928 camellia-128 cbc's in 2.90s
Doing camellia-128 cbc for 3s on 64 size blocks: 7892751 camellia-128 cbc's in 2.91s
Doing camellia-128 cbc for 3s on 256 size blocks: 2087062 camellia-128 cbc's in 2.75s
Doing camellia-128 cbc for 3s on 1024 size blocks: 617843 camellia-128 cbc's in 2.95s
Doing camellia-128 cbc for 3s on 8192 size blocks: 70865 camellia-128 cbc's in 2.80s
Doing camellia-192 cbc for 3s on 16 size blocks: 17968337 camellia-192 cbc's in 2.90s
Doing camellia-192 cbc for 3s on 64 size blocks: 6590076 camellia-192 cbc's in 2.93s
Doing camellia-192 cbc for 3s on 256 size blocks: 1848253 camellia-192 cbc's in 2.94s
Doing camellia-192 cbc for 3s on 1024 size blocks: 464222 camellia-192 cbc's in 2.90s
Doing camellia-192 cbc for 3s on 8192 size blocks: 61071 camellia-192 cbc's in 2.93s
Doing camellia-256 cbc for 3s on 16 size blocks: 18778875 camellia-256 cbc's in 2.93s
Doing camellia-256 cbc for 3s on 64 size blocks: 6622379 camellia-256 cbc's in 2.92s
Doing camellia-256 cbc for 3s on 256 size blocks: 1927303 camellia-256 cbc's in 2.96s
Doing camellia-256 cbc for 3s on 1024 size blocks: 498645 camellia-256 cbc's in 2.97s
Doing camellia-256 cbc for 3s on 8192 size blocks: 63064 camellia-256 cbc's in 2.98s
Doing seed cbc for 3s on 16 size blocks: 17710507 seed cbc's in 2.96s
Doing seed cbc for 3s on 64 size blocks: 4537077 seed cbc's in 2.97s
```

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

```
Doing seed cbc for 3s on 256 size blocks: 1081589 seed cbc's in 2.93s
Doing seed cbc for 3s on 1024 size blocks: 245812 seed cbc's in 2.73s
Doing seed cbc for 3s on 8192 size blocks: 31752 seed cbc's in 2.91s
Doing rc2 cbc for 3s on 16 size blocks: 9550805 rc2 cbc's in 2.88s
Doing rc2 cbc for 3s on 64 size blocks: 2597816 rc2 cbc's in 2.90s
Doing rc2 cbc for 3s on 256 size blocks: 672591 rc2 cbc's in 2.94s
Doing rc2 cbc for 3s on 1024 size blocks: 173657 rc2 cbc's in 2.95s
Doing rc2 cbc for 3s on 8192 size blocks: 21881 rc2 cbc's in 2.98s
Doing blowfish cbc for 3s on 16 size blocks: 25703308 blowfish cbc's in 2.89s
Doing blowfish cbc for 3s on 64 size blocks: 7021857 blowfish cbc's in 2.94s
Doing blowfish cbc for 3s on 256 size blocks: 1724267 blowfish cbc's in 2.95s
Doing blowfish cbc for 3s on 1024 size blocks: 455237 blowfish cbc's in 2.95s
Doing blowfish cbc for 3s on 8192 size blocks: 52988 blowfish cbc's in 2.94s
Doing cast cbc for 3s on 16 size blocks: 22269934 cast cbc's in 2.95s
Doing cast cbc for 3s on 64 size blocks: 5839502 cast cbc's in 2.90s
Doing cast cbc for 3s on 256 size blocks: 1453349 cast cbc's in 2.95s
Doing cast cbc for 3s on 1024 size blocks: 389211 cast cbc's in 2.95s
Doing cast cbc for 3s on 8192 size blocks: 46505 cast cbc's in 2.88s
Doing 512 bit private rsa's for 10s: 198742 512 bit private RSA's in 9.76s
Doing 512 bit public rsa's for 10s: 2254431 512 bit public RSA's in 9.11s
Doing 1024 bit private rsa's for 10s: 64754 1024 bit private RSA's in 9.27s
Doing 1024 bit public rsa's for 10s: 1035439 1024 bit public RSA's in 9.63s
Doing 2048 bit private rsa's for 10s: 14657 2048 bit private RSA's in 9.69s
Doing 2048 bit public rsa's for 10s: 348441 2048 bit public RSA's in 9.62s
Doing 4096 bit private rsa's for 10s: 1519 4096 bit private RSA's in 9.52s
Doing 4096 bit public rsa's for 10s: 92297 4096 bit public RSA's in 9.57s
Doing 512 bit sign dsa's for 10s: 178254 512 bit DSA signs in 9.64s
Doing 512 bit verify dsa's for 10s: 211215 512 bit DSA verify in 9.89s
Doing 1024 bit sign dsa's for 10s: 89761 1024 bit DSA signs in 9.58s
Doing 1024 bit verify dsa's for 10s: 85750 1024 bit DSA verify in 9.56s
Doing 2048 bit sign dsa's for 10s: 27404 2048 bit DSA signs in 9.28s
Doing 2048 bit verify dsa's for 10s: 28030 2048 bit DSA verify in 9.69s
Doing 160 bit sign ecdsa's for 10s: 24123 160 bit ECDSA signs in 9.52s
Doing 160 bit verify ecdsa's for 10s: 32020 160 bit ECDSA verify in 9.16s
Doing 192 bit sign ecdsa's for 10s: 21408 192 bit ECDSA signs in 9.63s
Doing 192 bit verify ecdsa's for 10s: 33773 192 bit ECDSA verify in 9.68s
Doing 224 bit sign ecdsa's for 10s: 103236 224 bit ECDSA signs in 9.75s
Doing 224 bit verify ecdsa's for 10s: 69284 224 bit ECDSA verify in 9.55s
Doing 256 bit sign ecdsa's for 10s: 111580 256 bit ECDSA signs in 9.65s
Doing 256 bit verify ecdsa's for 10s: 80882 256 bit ECDSA verify in 9.37s
Doing 384 bit sign ecdsa's for 10s: 5767 384 bit ECDSA signs in 9.41s
Doing 384 bit verify ecdsa's for 10s: 10016 384 bit ECDSA verify in 9.41s
Doing 521 bit sign ecdsa's for 10s: 23874 521 bit ECDSA signs in 9.68s
Doing 521 bit verify ecdsa's for 10s: 16003 521 bit ECDSA verify in 9.78s
Doing 163 bit sign ecdsa's for 10s: 6462 163 bit ECDSA signs in 9.87s
Doing 163 bit verify ecdsa's for 10s: 18002 163 bit ECDSA verify in 9.68s
Doing 233 bit sign ecdsa's for 10s: 2842 233 bit ECDSA signs in 9.90s
Doing 233 bit verify ecdsa's for 10s: 14742 233 bit ECDSA verify in 9.85s
Doing 283 bit sign ecdsa's for 10s: 1996 283 bit ECDSA signs in 9.99s
Doing 283 bit verify ecdsa's for 10s: 9668 283 bit ECDSA verify in 9.98s
Doing 409 bit sign ecdsa's for 10s: 852 409 bit ECDSA signs in 10.00s
Doing 409 bit verify ecdsa's for 10s: 5801 409 bit ECDSA verify in 9.97s
Doing 571 bit sign ecdsa's for 10s: 377 571 bit ECDSA signs in 9.73s
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
Doing 571 bit verify ecdsa's for 10s: 2295 571 bit ECDSA verify in 9.56s
Doing 163 bit sign ecdsa's for 10s: 6157 163 bit ECDSA signs in 9.52s
Doing 163 bit verify ecdsa's for 10s: 17342 163 bit ECDSA verify in 9.63s
Doing 233 bit sign ecdsa's for 10s: 2552 233 bit ECDSA signs in 9.56s
Doing 233 bit verify ecdsa's for 10s: 14214 233 bit ECDSA verify in 9.63s
Doing 283 bit sign ecdsa's for 10s: 1887 283 bit ECDSA signs in 9.69s
Doing 283 bit verify ecdsa's for 10s: 8380 283 bit ECDSA verify in 9.83s
Doing 409 bit sign ecdsa's for 10s: 843 409 bit ECDSA signs in 9.65s
Doing 409 bit verify ecdsa's for 10s: 5406 409 bit ECDSA verify in 9.76s
Doing 571 bit sign ecdsa's for 10s: 386 571 bit ECDSA signs in 9.85s
Doing 571 bit verify ecdsa's for 10s: 2483 571 bit ECDSA verify in 9.90s
Doing 160 bit ecdh's for 10s: 27689 160-bit ECDH ops in 9.75s
Doing 192 bit ecdh's for 10s: 19474 192-bit ECDH ops in 9.23s
Doing 224 bit ecdh's for 10s: 67799 224-bit ECDH ops in 7.16s
Doing 256 bit ecdh's for 10s: 83085 256-bit ECDH ops in 7.11s
Doing 384 bit ecdh's for 10s: 4247 384-bit ECDH ops in 7.03s
Doing 521 bit ecdh's for 10s: 15502 521-bit ECDH ops in 7.02s
Doing 163 bit ecdh's for 10s: 30228 163-bit ECDH ops in 8.19s
Doing 233 bit ecdh's for 10s: 25536 233-bit ECDH ops in 8.96s
Doing 283 bit ecdh's for 10s: 15938 283-bit ECDH ops in 9.11s
Doing 409 bit ecdh's for 10s: 10466 409-bit ECDH ops in 9.31s
Doing 571 bit ecdh's for 10s: 4520 571-bit ECDH ops in 9.27s
Doing 163 bit ecdh's for 10s: 30641 163-bit ECDH ops in 9.18s
Doing 233 bit ecdh's for 10s: 25194 233-bit ECDH ops in 9.29s
Doing 283 bit ecdh's for 10s: 14688 283-bit ECDH ops in 9.05s
Doing 409 bit ecdh's for 10s: 9588 409-bit ECDH ops in 9.46s
Doing 571 bit ecdh's for 10s: 3965 571-bit ECDH ops in 8.98s
OpenSSL 1.0.2g 1 Mar 2016
built on: reproducible build, date unspecified
options:bn(64,64) rc4(16x,int) des(idx,cisc,16,int) aes(partial) blowfish(idx)
compiler: cc -I. -I.. -I./include -fPIC -DOPENSSL_PIC -DOPENSSL_THREADS -D_REENTRANT -c -fPIC -m64 -mtune=generic -O3 -D_FORTIFY_SOURCE=2 -Wl,-Bsymbolic-functions -Wl,-z,relro -Wa,--noexecstack
A256_ASM -DSHA512_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM
The 'numbers' are in 1000s of bytes per second processed.
type          16 bytes      64 bytes     256 bytes    1024 bytes     8192 bytes
md2            0.00         0.00        0.00        0.00        0.00
mdc2           0.00         0.00        0.00        0.00        0.00
md4       61502.06k   215091.03k   594134.85k   982079.82k   1197011.11k
md5       46951.54k   152853.33k   384641.56k   557549.32k   494202.88k
hmac(md5)    35802.25k   131518.92k   350439.35k   561196.08k   681210.31k
sha1       51000.34k   158050.34k   398546.38k   572064.74k   744015.98k
rmd160     35424.93k   95184.39k   182611.85k   245480.70k   267047.74k
rc4        535481.13k   667274.38k   603011.26k   606781.78k   566308.30k
des cbc    79412.89k   83312.55k   83198.88k   86755.75k   86199.28k
des ede3    31415.85k   31128.69k   30965.54k   31039.57k   31391.63k
idea cbc    0.00         0.00        0.00        0.00        0.00
seed cbc    95732.47k   97768.66k   94500.61k   92202.01k   89385.70k
rc2 cbc    53060.03k   57331.11k   58565.75k   60279.58k   60150.72k
rc5-32/12 cbc  0.00         0.00        0.00        0.00        0.00
blowfish cbc 142302.05k   152856.75k   149631.31k   158021.25k   147645.47k
cast cbc    120786.08k   128871.77k   126121.13k   135102.39k   132280.89k
aes-128 cbc 134834.50k   153778.19k   150165.64k   329108.30k   303732.47k
aes-192 cbc  98097.46k   119676.61k   118186.70k   262696.67k   284799.72k
aes-256 cbc  95778.74k   113542.44k   114658.36k   254039.28k   255502.43k
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

	0.00	0.00	0.00	0.00	0.00
rc5-32/12 cbc					
blowfish cbc	142302.05k	152856.75k	149631.31k	158021.25k	147645.47k
cast cbc	120786.08k	128871.77k	126121.13k	135102.39k	132280.89k
aes-128 cbc	134834.50k	153778.19k	150165.64k	329108.30k	303732.47k
aes-192 cbc	98097.46k	119676.61k	118186.70k	262696.67k	284799.72k
aes-256 cbc	95778.74k	113542.44k	114658.36k	254039.28k	255502.43k
camellia-128 cbc	112827.19k	173586.28k	194286.50k	214464.82k	207330.74k
camellia-192 cbc	99135.65k	143947.05k	160936.32k	163918.39k	170748.68k
camellia-256 cbc	102546.76k	145148.03k	166685.66k	171923.39k	173362.51k
sha256	63430.67k	148191.15k	266759.25k	358360.94k	386964.69k
sha512	50544.60k	210102.22k	340878.31k	486110.50k	566247.70k
whirlpool	36017.09k	76780.91k	134480.47k	147445.36k	167711.56k
aes-128 ige	135331.68k	150372.18k	155011.79k	155450.94k	149236.07k
aes-192 ige	101564.56k	113373.50k	120337.95k	125256.50k	129422.53k
aes-256 ige	97489.69k	100677.76k	108792.80k	103658.71k	105826.27k
ghash	600938.94k	2279013.34k	5849220.92k	7915477.51k	9521001.86k
	sign	verify	sign/s	verify/s	
rsa 512 bits	0.000049s	0.000004s	20362.9	247467.7	
rsa 1024 bits	0.000143s	0.000009s	6985.3	107522.2	
rsa 2048 bits	0.000661s	0.000028s	1512.6	36220.5	
rsa 4096 bits	0.006267s	0.000104s	159.6	9644.4	
	sign	verify	sign/s	verify/s	
dsa 512 bits	0.000054s	0.000047s	18491.1	21356.4	
dsa 1024 bits	0.000107s	0.000111s	9369.6	8969.7	
dsa 2048 bits	0.000339s	0.000346s	2953.0	2892.7	
	sign	verify	sign/s	verify/s	
160 bit ecdsa (secp160r1)	0.0004s	0.0003s	2533.9	3495.6	
192 bit ecdsa (nistp192)	0.0004s	0.0003s	2223.1	3488.9	
224 bit ecdsa (nistp224)	0.0001s	0.0001s	10588.3	7254.9	
256 bit ecdsa (nistp256)	0.0001s	0.0001s	11562.7	8632.0	
384 bit ecdsa (nistp384)	0.0016s	0.0009s	612.9	1064.4	
521 bit ecdsa (nistp521)	0.0004s	0.0006s	2466.3	1636.3	
163 bit ecdsa (nistk163)	0.0015s	0.0005s	654.7	1859.7	
233 bit ecdsa (nistk233)	0.0035s	0.0007s	287.1	1496.6	
283 bit ecdsa (nistk283)	0.0050s	0.0010s	199.8	968.7	
409 bit ecdsa (nistk409)	0.0117s	0.0017s	85.2	581.8	
571 bit ecdsa (nistk571)	0.0258s	0.0042s	38.7	240.1	
163 bit ecdsa (nistb163)	0.0015s	0.0006s	646.7	1800.8	
233 bit ecdsa (nistb233)	0.0037s	0.0007s	266.9	1476.0	
283 bit ecdsa (nistb283)	0.0051s	0.0012s	194.7	852.5	
409 bit ecdsa (nistb409)	0.0114s	0.0018s	87.4	553.9	
571 bit ecdsa (nistb571)	0.0255s	0.0040s	39.2	250.8	
	op	op/s			
160 bit ecdh (secp160r1)	0.0004s	2839.9			
192 bit ecdh (nistp192)	0.0005s	2109.9			
224 bit ecdh (nistp224)	0.0001s	9469.1			
256 bit ecdh (nistp256)	0.0001s	11685.7			
384 bit ecdh (nistp384)	0.0017s	604.1			
521 bit ecdh (nistp521)	0.0005s	2208.3			
163 bit ecdh (nistk163)	0.0003s	3690.8			
233 bit ecdh (nistk233)	0.0004s	2850.0			
283 bit ecdh (nistk283)	0.0006s	1749.5			
409 bit ecdh (nistk409)	0.0009s	1124.2			
571 bit ecdh (nistk571)	0.0021s	487.6			
163 bit ecdh (nistb163)	0.0003s	3337.8			
233 bit ecdh (nistb233)	0.0004s	2711.9			
283 bit ecdh (nistb283)	0.0006s	1623.0			
409 bit ecdh (nistb409)	0.0010s	1013.5			
571 bit ecdh (nistb571)	0.0023s	441.5			

Vamos a comparar un cifrador simétrico, como el AES, con un cifrador asimétrico, como el RSA. Los mostramos en las siguientes capturas:

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Resultados del AES modo CBC:

```
Doing aes-128 cbc for 3s on 16 size blocks: 24775840 aes-128 cbc's in 2.94s
Doing aes-128 cbc for 3s on 64 size blocks: 6823907 aes-128 cbc's in 2.84s
Doing aes-128 cbc for 3s on 256 size blocks: 1630705 aes-128 cbc's in 2.78s
Doing aes-128 cbc for 3s on 1024 size blocks: 932045 aes-128 cbc's in 2.90s
Doing aes-128 cbc for 3s on 8192 size blocks: 97141 aes-128 cbc's in 2.62s
Doing aes-192 cbc for 3s on 16 size blocks: 16308702 aes-192 cbc's in 2.66s
Doing aes-192 cbc for 3s on 64 size blocks: 5291950 aes-192 cbc's in 2.83s
Doing aes-192 cbc for 3s on 256 size blocks: 1292667 aes-192 cbc's in 2.80s
Doing aes-192 cbc for 3s on 1024 size blocks: 723442 aes-192 cbc's in 2.82s
Doing aes-192 cbc for 3s on 8192 size blocks: 96996 aes-192 cbc's in 2.79s
Doing aes-256 cbc for 3s on 16 size blocks: 17539481 aes-256 cbc's in 2.93s
Doing aes-256 cbc for 3s on 64 size blocks: 5286820 aes-256 cbc's in 2.98s
Doing aes-256 cbc for 3s on 256 size blocks: 1334695 aes-256 cbc's in 2.98s
Doing aes-256 cbc for 3s on 1024 size blocks: 739294 aes-256 cbc's in 2.98s
Doing aes-256 cbc for 3s on 8192 size blocks: 92944 aes-256 cbc's in 2.98s
```

Resultados del RSA:

```
Doing 512 bit private rsa's for 10s: 198742 512 bit private RSA's in 9.76s
Doing 512 bit public rsa's for 10s: 2254431 512 bit public RSA's in 9.11s
Doing 1024 bit private rsa's for 10s: 64754 1024 bit private RSA's in 9.27s
Doing 1024 bit public rsa's for 10s: 1035439 1024 bit public RSA's in 9.63s
Doing 2048 bit private rsa's for 10s: 14657 2048 bit private RSA's in 9.69s
Doing 2048 bit public rsa's for 10s: 348441 2048 bit public RSA's in 9.62s
Doing 4096 bit private rsa's for 10s: 1519 4096 bit private RSA's in 9.52s
Doing 4096 bit public rsa's for 10s: 92297 4096 bit public RSA's in 9.57s
```

Se puede ver que los bloques que cifra el algoritmo AES en modo CBC van desde 16 bits hasta 8192 bits (por cada tamaño de clave), y el RSA cifra bloques desde 512 bits hasta 4096 bits (por cada tipo de clave). Además, los tiempos de ejecución de la prueba son distintos, ya que para el AES se dejan 3 segundos mientras que para el RSA se dejan 10 segundos.

A continuación, hemos tabulado los datos anteriores para hacer una gráfica comparativa de todas las ejecuciones de ambos algoritmos. Y, además, hemos generado una tabla equivalente pero con los valores de ejecuciones en 1 segundo (dividiendo cada valor por el tiempo de ejecución de la prueba anterior).

	3s	3s	3s	10s	10s		1s	1s	1s	1s	1s	
	AES-128	AES-192	AES-256	RSA PRIV	RSA PUB		AES-128	AES-192	AES-256	RSA PRIV	RSA PUB	
16	24775840	16308702	17539481	0	0		16	8258613	5436234	5846494	0	0
64	6823907	5291950	5286820	0	0		64	2274636	1763983	1762273	0	0
256	1630705	1292667	1334695	0	0		256	543568,3	430889	444898,3	0	0
512	0	0	0	198742	2254431		512	0	0	0	19874,2	225443,1
1024	932045	723442	739294	64754	1035439		1024	310681,7	241147,3	246431,3	6475,4	103543,9
2048	0	0	0	14657	348441		2048	0	0	0	1465,7	34844,1
4096	0	0	0	1519	92297		4096	0	0	0	151,9	9229,7
8192	97141	96996	92944	0	0		8192	32380,33	32332	30981,33	0	0

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

El gráfico comparativo es el siguiente:



Se puede ver que según va aumentando el tamaño del bloque van disminuyendo el número de ejecuciones por segundo. La verdadera comparación se puede hacer para tamaño de bloque 1024, donde vemos que el AES (cifrado simétrico) tiene muchas más ejecuciones por segundo que el RSA (cifrado asimétrico), apenas se ven las barras. En conclusión, podemos ver que el cifrado simétrico es mucho más rápido que el cifrado asimétrico.

## E. Certificados X.509

Estos certificados proporcionan seguridad en el caso de estar firmados por una autoridad certificadora (CA) de confianza. Todos los mensajes que contengan cualquier certificado firmado pueden ser considerados seguros en la capa de transporte. El certificado te asocia a una clave privada, y el resto de personas pueden confiar en que cifrar sus mensajes con la clave pública asociada a esa clave privada firmada solo lo puede descifrar el receptor al que va dirigido el mensaje.

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

A continuación, hemos generado mediante OpenSSL un certificado y lo hemos firmado con nuestros datos:

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:4096 -keyout private.key -out certificate.crt
Generating a 4096 bit RSA private key
+++
.....+
writing new private key to 'private.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:SP
State or Province Name (full name) [Some-State]:Madrid
Locality Name (eg, city) []:Alcobendas
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:Laura Sanchez Herrera
Email Address []:laura.sanchezherrera@estudiante.uam.es
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```

En las siguientes capturas podemos ver el contenido del certificado, como la información personal y la firma Hash:

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ openssl x509 -in certificate.crt -text -noout
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 16558409840300959134 (0xe5cb4a050078a59e)
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=SP, ST=Madrid, L=Alcobendas, O=Internet Widgits Pty Ltd, CN=Laura Sanchez Herrera/emailAddress=laura.sanchezherrera@estudiante.uam.es
Validity
    Not Before: Dec 13 15:15:44 2020 GMT
    Not After : Dec 13 15:15:44 2021 GMT
Subject: C=SP, ST=Madrid, L=Alcobendas, O=Internet Widgits Pty Ltd, CN=Laura Sanchez Herrera/emailAddress=laura.sanchezherrera@estudiante.uam.es
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (4096 bit)
        Modulus:
            00:ad:5b:c5:59:c8:d9:5e:1d:c0:9a:dc:92:5a:10:
            41:9f:3f:41:bc:0c:7d:0d:3c:9e:7:c4:b1:34:8b:
            88:2a:02:ef:b5:a:fb:3e:45:08:5a:93:e3:b1:25:
            ff:6c:f3:fa:ef:77:be:0d:7e:ca:2a:7f:e:1:94:ca:
            96:98:05:39:86:7e:72:b0:0f:90:0c:dd:be:a1:1c:
            ab:13:a0:56:9a:d1:f8:e5:66:95:6b:25:33:d7:72:
            6a:89:65:b7:31:9f:72:61:4a:9f:8a:a0:af:75:b5:
            c7:b5:b5:ab:e8:1f:1f:84:67:94:50:99:c4:c1:38:
            70:2e:fe:cb:40:c0:00:b4:4e:da:a6:1a:d2:8f:83:
            46:f5:ef:d1:a4:37:67:5e:9e:31:3d:32:6b:b9:07:
            79:b4:03:b6:a1:62:fc:e8:36:93:66:b6:99:2d:b3:
            a8:f8:Bb:c1:d6:3c:7b:02:60:08:12:7e:9e:6d:c0:
            64:26:00:59:f0:24:42:7e:37:0e:ba:85:85:8d:02:
            aa:50:e6:a9:fe:38:48:91:ff:e2:2d:2c:3c:bc:eb:
            64:c8:f0:97:bb:9c:ff:0e:90:39:eb:10:b6:7c:2e:
            a5:a4:1c:8a:ea:cf:f0:a2:b9:19:27:1a:40:ec:dd:
            42:81:5c:8c:3d:12:cc:13:13:50:be:a:f:a5:d4:57:
            18:4f:fa:e5:0d:00:c2:fc:03:ba:99:e0:6:a:ee:13:
            a7:c8:51:25:59:fa:34:2f:f1:fe:f7:fd:e2:c6:d5:
            52:07:99:06:53:2a:1c:ab:c9:b3:09:35:71:49:50:
            95:e2:c6:0e:81:8a:cc:1e:ae:39:ee:44:1c:93:6c:
            cd:91:97:f1:96:08:a0:6c:75:f3:bf:b1:f6:62:66:
            da:6a:0d:42:be:a0:f6:e4:07:e8:2f:5a:d0:bb:66:
            99:92:10:c9:f5:d9:6e:3e:7a:38:f9:d4:08:81:18:
            65:56:3c:0a:0f:09:15:12:a9:48:a7:13:90:0f:c3:
            99:32:a8:46:fd:e5:fb:13:3e:89:37:3c:c4:68:0a:
            37:b5:a4:7f:7f:8f:48:4a:0e:92:88:ce:4f:97:c0:
            f6:42:7f:c7:8f:f2:98:69:3e:61:f6:e2:87:e2:63:
            bc:c5:78:d4:a5:6f:b9:43:dd:0c:2a:d3:e3:3e:6b:
            9e:24:cb:0c:0f:2f:71:84:96:24:40:70:30:2c:38:
            37:68:28:1b:2d:6d:24:96:82:7b:cf:4d:61:3b:ef:
            bf:59:0a:da:2c:1e:7:53:10:83:18:22:4e:ab:b6:c7:
            de:99:77:9d:21:0b:27:d4:df:09:37:86:08:cc:f2:
            a1:7c:8f:d9:f1:dd:0a:63:82:0a:6e:cd:c3:67:a1:
            78:ca:99
        Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Subject Key Identifier:
39:A0:6B:70:AD:C9:F5:F7:08:03:5F:CE:7E:58:35:E8:72:A6:B8:1E
```

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

```
cd:91:97:f1:96:08:a0:6c:75:f3:bf:b1:f6:62:66:  
da:6a:0d:42:8e:a0:f6:e4:07:e8:2f:5a:d0:bb:66:  
99:92:10:c9:f5:d9:6e:3e:7a:38:f9:d4:08:81:18:  
65:56:3c:0a:0f:09:15:12:89:48:a7:13:90:0f:c3:  
99:32:a8:46:fd:e5:fb:13:36:89:37:3c:c4:68:0a:  
37:b5:a4:7f:7f:8f:48:4a:0e:92:88:ce:4f:97:c0:  
f6:42:7f:c7:78:f2:98:69:3e:61:f6:e2:87:e2:63:  
bc:c5:78:4d:a5:6f:b9:43:dd:0c:2a:d3:e3:3e:6b:  
9e:24:cb:0c:0f:2f:71:84:96:24:40:70:30:2c:38:  
37:68:28:1b:2d:6d:24:96:82:7b:cf:4d:61:3b:ef:  
bf:59:0a:da:2c:e7:53:10:83:18:22:4e:ab:b6:c7:  
de:99:77:9d:21:0b:27:d4:df:09:37:86:08:cc:f2:  
a1:7c:8f:d9:f1:dd:0a:63:82:0a:6e:cd:c3:67:a1:  
78:ca:99  
Exponent: 65537 (0x10001)  
X509v3 extensions:  
X509v3 Subject Key Identifier:  
39:A0:6B:70:AD:C9:F5:F7:08:03:5F:CE:7E:58:35:E8:72:A6:B8:1E  
X509v3 Authority Key Identifier:  
keyid:39:A0:6B:70:AD:C9:F5:F7:08:03:5F:CE:7E:58:35:E8:72:A6:B8:1E  
X509v3 Basic Constraints:  
CA:TRUE  
Signature Algorithm: sha256WithRSAEncryption  
9e:10:f7:1a:c2:cb:e1:f6:21:b9:65:eb:11:4f:af:d8:67:30:  
5f:97:ec:63:5d:43:0b:e7:f0:e7:53:fc:76:23:31:1e:36:b2:  
e4:78:92:0c:42:92:b1:e1:a6:15:fc:0a:a3:27:c0:cf:d5:12:  
f6:cb:64:f4:08:88:cf:7d:5a:6e:48:c4:64:2c:ed:6c:8b:63:  
66:2a:02:b5:fc:7b:09:da:40:03:d4:74:5c:46:5a:5e:72:e6:  
57:69:c6:2e:d3:bf:d2:3a:a7:cf:46:fe:5e:cd:59:35:f8:27:  
0f:46:dc:fc:2f:34:5f:e0:d3:3a:67:56:1b:26:ca:fc:ba:84:  
a2:c2:19:9f:75:76:e9:8f:cf:60:62:6c:a3:fe:fd:fc:09:cd:  
6c:f6:75:53:eb:d7:ce:05:da:28:0f:8e:c9:d6:09:5f:4f:ad:  
56:e0:ed:61:7c:10:6a:86:d7:84:0c:4a:f1:74:f3:9c:00:e7:  
52:ac:77:8b:09:a4:ea:07:01:88:24:88:a7:ae:e0:1c:d7:5f:  
75:d9:40:96:a9:10:71:22:3f:32:7a:f7:d6:b9:48:fc:c6:32:  
97:2a:5e:73:3c:76:c2:57:53:15:f8:7b:63:93:8b:81:c4:4a:  
14:c9:c4:46:c1:70:ed:b0:b6:49:6c:c6:4f:fb:32:13:e4:1e:  
99:22:a5:c5:a4:43:3f:37:47:58:fb:58:46:df:af:6c:83:b6:  
84:59:b7:ea:d3:aa:09:5d:87:fc:0c:75:a4:b7:c8:86:f5:d6:  
f0:ea:8b:ae:69:c5:b6:4a:38:70:68:49:d0:16:e3:48:1e:ad:  
e6:1c:83:dd:42:8f:a4:6e:74:96:d4:6b:50:8f:dd:64:13:a6:  
63:5d:88:70:1a:55:e9:63:5e:c0:2b:d1:56:82:35:a7:00:bf:  
a3:0d:f6:0c:a3:af:84:bb:20:e3:ad:f2:d1:f6:8e:3d:7e:9b:  
26:ad:46:e0:38:5e:5f:69:a4:9f:8b:77:bc:b3:bb:8b:ee:cd:  
1d:e6:fa:35:31:ad:b3:06:1b:e7:29:14:6c:60:2e:14:02:87:  
ef:9a:08:2d:b8:4c:b2:c7:14:f4:8a:40:27:0e:82:5d:5d:4b:  
b8:ec:40:2d:43:c4:41:f5:e5:15:85:5a:50:13:ad:2b:5b:d0:  
fc:03:1e:6e:8d:51:fd:72:9b:83:a4:ed:60:02:cd:79:6e:35:  
87:ed:a9:23:6f:8e:fd:be:09:d9:a1:31:e3:5b:2d:25:61:25:  
01:dd:ef:40:76:62:ef:0e:82:de:f8:c8:42:6e:48:d3:71:9a:  
55:60:44:8f:6b:39:0c:90:8f:e1:c7:99:5e:a8:eb:65:99:ab:  
ad:c7:00:98:8a:9a:c9:2b  
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ █
```

Mostramos la clave privada de RSA generada y firmada con el certificado anterior:

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ cat private.key
-----BEGIN PRIVATE KEY-----
MIJQQIBADNBgkqhkiG9w0BAQEFAASCCSswggknAgEAAoICAQCTw2xZyNleHcCa
3JJaEEGfP0G8DH0NPJ7nxLE0i4gqAu+1Svs+RQhak+0xJf9s8/rvd74Nfsoqf+GU
ypaYBTmGfnKwD5AM3b6hHKsToFaa0fjZpVrJTPXcmqJZbcxn3JhSp+KoK91tce1
tavoHx+EZ5RqmCTB0HAu/stAwAC0TtqmGtKPg0b179GkN2denjE9Mmu5B3m0A7ah
YvzoNpNmtpkts6j4i8HWPHsCbbgSfp5twQ0mAfnwJE3+Nw66hYWNAqp05qn+0EiR
/+ItLDy862TI8JeLnP8OnTnrELZ8LqWkHIRqz/CiuRknGkDs3UKBXiW9EswTE1C+
r6XUVxhP+uUNAML8A7qZ4GruE6fIUSVz+jQv8f73/eLG1V1HmQZTKhyrybMJNXFJ
UJXixg6BiswerjnuRByTbM2Rl/GWCKBsdf0/sfZiZtpqDUK0oPbkB+gvWtC7ZpmS
EMn12W4+ejj51AiBGGVWPAoPCRUSqUiinE5APw5kyqEb95fsTNok3PMR0Cje1pH9/
johKDpK1zk+xWPZCf8948phpPmH24ofiy7zFeE2lb7ld3Qwq0+M+a54kywwPL3GE
lirAcDaSODdoKBstbSSWgnvPTWE7779ZCtos51MQgxqiTqu2x96Zd50hCyfU3wk3
hgjM8qF8j9nx3QpjggpuzcNnoXjKmQIDAQABAoICAFOu0jr1Lzi1HAileTBosITC
8GRqoNU5BBVboVC5dQb3tzQ+T8x67ptsXg0M6LQk82NK/0vuz+Qw8Ajm1Z/DynR
WGOM36iWUJNLt3md6LCLs25peq9iIghafvscUrZtlbVJZIjy1sAno+Qfcob2paww
o/PUhp0POHg3Hr5+8NgZSIRuX75PTK3juntfE1MLYfd8o2uKJEG1mS+GrTcbQURP
2Nmq5GI2dLeKMocVl8m1vZyVRsLhYiJ/t+ldgoqfcbyEENTEvkQuQLnZn8DWnetE
Zj5phX4rpyPS2eaTBsQjcGosLRL0leC3IWey1yyAE16H4lqI/GIo7M1f1NkPwfFt
5/Tc2f0mOXw0eiWdgWaBLmqTuyJcn9IN04APMEvIMN/cbFztRePqgfhkXYCaYV+N
72/DZMzAK1o3l5qvewFPjthStu3gE3mGV802nh8FB20gvKbEBPHWlyjgZm5PJ
89SNqwZmLAX6HaGYOB6e98xDKBJg/Ulrngb+0Xgr6SJwsDJvj074UeYsc9WRt+gR
wmQFqk1Sw8sbkon2dTBUv0Aj1KIXtvVC9ULQnMTqr9VflMBm00N+U87gVKxDKXIS
3Ev/SWIqltz0Tuq6CAqfVXalYhF4JFu6QceZ+Bnm3bjzbdlbUpxuv2Y3QVXuc0YC
Vpoatb0ogfbFm6Mn80zBBAoIBAQDiWA7zGCRdK9s72t/pVoCq/R8aogunaD3YUNFR
gvZapGrGGT1IezzauYMruxCvCh+7Pq9LYFmdBJZTzQKQLjBvCVkEoeToJV95BHT3
Fz4t7zb3Sh8mHSz9xSEhTwyslcTitjGZ3LmSGCm3DW54RN+lm4Tsro8C/PKa25r
60NFgz0980TvI74D57VHSZ6xpa2k86uP+u5mu98/AcNjs/MlMFx+dbWNzfRxuV
p65MeBNAbmneaunUpZ0RoeZYK85QcUe4aZ8DQCFDFJwEYji7XKeK1zPIBci+NLT
AfPQhTRICkPXPEsCsAFTEqPCHsFnOnDM0DKOp5DaTmS48jttAoIBAQDEEhmwdM2+
hxSiKy3bSeGls6u8qgM2iQOEIy8TfQW3d27ak9SyQuTuBtJKaz6KSVTAgl2Ujjq
4Tw4yJFJftJElgyHPHvib9+zEw9LgTKQA5FXCs3Mhn+X8jJRM2trj8PE2l3ZSw15
Vf2oT9Z6dznTvqV1Lwc3XL0QGPbg8xbxLLPGJrn99zhVlrV0mGVnV9vf6S8ABFq/
NGtueqxjlruosUjQ8+46yo96RP1HXQ/QQKV6XWNZyGzQFpm0D8vN+hu1sXwrzMq/
VgvQot5Q0nXpYEWQyhLidAC7+JVFWgOvQIm4XxGQp7qpNT/1c69lpkvBnhVAZIrB
LBWm2zDECYRdAoIBAFcgRGjo0x6WQiayxt3DU9SlaLksy8WeB6hb6cfvUq43Wxx
x+Oh5JPhMw/CrYc34J8oCv906DwYhLR4fmz/erP81coJmAB+tUvlQKED0hLm6Hn
A0R9mJL6rb0PmTXSVN4zYDDnI0BnfeTdAUVS0Vk5+Jdwg38QLsCtR+z70Xgxg4J0
QQW4gW0pZQtThyTVirAvBzd15AwnB/Vr/JbgaL68SAxgXIVKTe1vhWH/TzE1lNd
hYU2x1tPAL4Hzw4DlsA65rw/dBsBNYyIRzFCXqnTcvpVHjkQjLdmhmi46yzRiYVt
hh201L7tGaXy2kV+5zzIN5lvTvbAo66U3Iqgm6UCggEAXKxeDW9CVAK0JuYEQ3a
js1e/s1ct2oLP2lh1tRA33aT7rvwzFDRfICIZQxNte+8tH8+5y0xGdnbo3spMjPA
zqodQc5Uv7IhcYL4t7dZLriA+aMhoonvx0G05qxnGW/aGDEeKpZ0yArc2bDirv+r
VTie+8Q+p0wwcGtdc3VyYou+b0ng/yjad5nHPNTLGQ1t32dXz6o/OiCMpqtu0Krhz
Z9pjF+70g6DwkbZb+Gml+H2VN/PFaClzbTg8TqGGdUdQ80f7RPewJDrBLH2YfU/
xg660o0IaYBq2G2mICFrnGnkkrkBBDR46G8APevQy8tlj73fb850SAbvUgAkaA458
AQKCAQB5B5G7RdnTqsLzLpsDHsN4joTfgMScqAUAnipqhkDt/Z+aECGqg6uuHI
68zbUNTLJYZstbf3iXk4FJ9wX4nDE7wT44eW0GxH6VYNbm4PqnSee/FXJPOQBqQF
KG1SJDgqQmbaubZHqbyB5N7GbZwn9smXY9fDylZltyqzSKifx9ZAKkoilFVBCDUD
Gy7yvAe6DeR1hzS2w4iks3GNSjJ+TtqbrD0GGtIPpuuMA7IeBaDPsAXGXqH6SILm
ANrAbw14EuZ8irm2wkU4GP+NuTplFpwsIfoscBlfG1l5fCQvjP50kyzD9hKUBwPV
Vyl7oMa5YOFk2Qy2SMSUsU09KPH0
-----END PRIVATE KEY-----
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ █
```

El hecho de autofirmarse un certificado no sirve para nada, ya que no se trata de una CA. Por lo tanto, ese certificado no puede considerarse seguro por otra persona.

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

## 2. RSA

Todos los programas que hemos realizado en este segundo punto utilizan la librería de gmp, por lo que se requiere poseer **libgmp.a** y **gmp.h** en la misma carpeta donde se realice el **make** que genere los ejecutables de los apartados siguientes.

### A. Potenciación de grandes números

El programa **potencia.c**, con su librería **potencia.h**, realiza la potenciación modular de grandes números. Los argumentos que recibe por terminal son los siguientes:

**./potencia [base] [exponente] [modulo]**

base: número base en el cálculo, b

exponente: cifra exponente en el cálculo, e

modulo: espacio modular del cálculo, m

Lo primero que hacemos es comprobar que los argumentos son correctos y guardarlos en las variables correspondientes. Y a partir de aquí comienza el cálculo.

Luego, calculamos el orden de bits del exponente **L** (empezando a mirar desde el bit 9000, al no tener un indicador previo del tamaño del número) y, a continuación, vamos recorriendo cada bit del exponente, empezando por el más significativo (**L**) hasta el menos significativo. Para cada bit realizamos la operación **x = x^2 mod n**, y x está inicializado a 1 en la primera iteración. En el caso de que ese bit sea 1 en el exponente, realizamos también **x = x\*b mod n**.

Y tras recorrer todos los bits obtenemos en x el resultado de la potenciación modular, siendo equivalente a **x = b^e mod n**.

Para comprobar el correcto funcionamiento de nuestro programa, comparamos nuestro resultado con el obtenido por gmp dentro del código.

Ejemplo de ejecución:

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P3$ ./potencia 123456789 12345 1234567891
El orden del exponente es: 14
--- Nuestro resultado:
123456789 ^ 12345 mod 1234567891 = 685397720
--- Resultado GMP:
123456789 ^ 12345 mod 1234567891 = 685397720
Se ha liberado correctamente la memoria...
laura@Universidad:~/Escritorio/CUARTO/cripto/P3$
```

Ambos resultados dan el mismo resultado, por lo tanto, es correcto nuestro cálculo.

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

## B. Generacion de numeros primos: Miller-Rabin

En este apartado se pide generar un número primo de tamaño definido por el usuario, de tal manera que haya pasado los tests de Miller-Rabin para ser considerado primo.

Para esto, hemos creado **primo.c** y **primo.h**, que contienen el programa correspondiente y las funciones necesarias para su funcionamiento correcto. Los argumentos de la ejecución son los siguientes:

**./primo -b num\_bits -p prob\_error [-o fileout]**

-b indica el tamaño en bits del primo generado

-p indica la probabilidad de equivocación utilizada como límite para considerarlo primo

-o fichero de salida opcional, si no se indica se imprimen los resultados por pantalla

Al principio, realizamos la comprobación de la introducción correcta de argumentos de entrada, dentro del método **read\_args()**.

Ahora, vamos a explicar de forma teórica, indicando el código equivalente, qué es lo que hay que hacer para obtener un número primo de ese tamaño y que pase los tests sin superar la probabilidad de equivocación introducida.

Primero, **se genera aleatoriamente un número de tamaño num\_bits bits**. Esto se hace poniendo el bit de esa posición a 1 y los demás aleatoriamente. Y, por si el número sale par, se pone el bit menos significativo a 1 también. En nuestro programa, generamos el número grande aleatorio con las funciones de GMP, creando una semilla y aplicando un rand, y ponemos solo el último bit a 1 (ya que la función GMP ya pone el más significativo a 1).

Tras la generación del número que queremos **comprobar si es primo**, le hacemos pasar por **una serie de tests de primalidad** de modo que al final, si los ha cumplido y el error no supera el exigido, se pueda dar como probablemente primo resultante. Si no se cumple esto, simplemente se suma 2 al número con el que hemos probado y se repite la misma secuencia de pruebas (si llega al límite  $2^{\text{num\_bits}}$  se vuelve a generar otro número aleatorio). Por lo tanto, en nuestro programa, esto se resume en un bucle while infinito que no se sale de él hasta que se encuentra un número que pase los tests de primalidad con un error menor al deseado, aumentando de 2 en 2 el número a probar en los tests.

Concretamente, un primer test que ha de pasar el número consiste en **dividirlo por los primos menores de 2000**, que se encuentran recogidos en un fichero aparte llamado **primos\_2k.txt**. Si el número no es divisible por ninguno de ellos, se

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

procede a la siguiente prueba de primalidad. Se dice que este primer test elimina el 99.8% de los números grandes que son compuestos. En nuestro código, esta prueba se hace en la función **prueba\_tabla\_primos()**.

A continuación, si el número ha pasado el test de la tabla de primos, se introduce en un test de primalidad, concretamente **Miller-Rabin**, y se repite tantas veces como se vea necesario para aumentar la certeza de primalidad del número que se prueba. Si al finalizar las distintas pruebas, el porcentaje de error es mayor que la probabilidad de equivocación introducida por argumento, ese número no se considera lo suficientemente primo y se continúa probando con el siguiente número. Este test se realiza en nuestra función **test\_miller\_rabin()**.

Pero antes, se tiene que hallar dos números necesarios para este test de primalidad. Estos números son **k** y **m**, que cumplen que  $n - 1 = m * 2^k$ , siendo k mayor que 0 y m impar. Ambos se calculan en el método **calcula\_k\_m()**, donde se itera en busca del m impar y el exponente k que cumplan la equivalencia anterior. Como estos dos números son siempre los mismos (para ese número), se calculan antes del bucle que va a iterar un número de veces aplicando los tests de Miller-Rabin.

Por lo tanto, con k y m hallados, se pasa el número por **NTEST** tests de Miller-Rabin, siendo una variable que se puede editar en la librería y que indica cuántas veces se ha de repetir el test. Cuanto mayor sea su valor, más pruebas se harán y el error será más preciso (aumentando la seguridad de primalidad del número).

Entonces, el **test de Miller-Rabin** consiste en ver si, mediante la generación de un número aleatorio **a** que cumpla  $1 < a < n - 1$ , se cumple alguna de las dos propiedades vistas en teoría que cumple un número primo. La primera consiste en realizar la potencia modular del número aleatorio, de tal manera que sea  $x = a^m \text{ mod } n$ . Si se cumple que x es igual a 1 o es igual a n-1, entonces se puede decir que el número es probablemente primo. La segunda consiste en iterar, desde 1 hasta k-1, de manera que se vaya obteniendo el cuadrado del x anterior, cumpliendo que  $x = a^{(m * 2^i)} \text{ mod } n$ , siendo i la iteración del bucle y exponente del 2. De manera que si en algún momento se cumple que x es igual a 1, se puede afirmar que es compuesto y, por otro lado, si x es igual a n-1 se puede decir que es probablemente primo. En el caso de que termine el bucle sin haber cumplido ninguna de las dos propiedades, entonces el número se considera compuesto en ese test.

Al final, por lo tanto, se obtiene un número de veces que el número pasa el test de Miller-Rabin y un número de veces que lo falla. Con estos últimos obtenemos el porcentaje de error o **probabilidad de equivocación del número** que se prueba. Si

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

este porcentaje es menor que el indicado al inicio por argumento, se da por probablemente primo y se devuelve, junto a sus valores de las pruebas (y los resultados de GMP). Si no es así, como ya hemos dicho, se continúa probando con el siguiente número.

Vamos a realizar una serie de pruebas de generación de primos que pasen el umbral de una probabilidad de error del 10%, con un número de test de 50, desde un tamaño de 100 bits a 8096 bits. Estos son los resultados:

```
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto/p3
$ ./primo -b 100 -p 0.1
bits 100 sec 0.100000
### Primo a probar: 1267645764525059029375087808511
> El primo no pasa la prueba de la tabla de primos. Es compuesto

### Primo a probar: 1267645764525059029375087808513
> El primo no pasa la prueba de la tabla de primos. Es compuesto

### Primo a probar: 1267645764525059029375087808515
> El primo no pasa la prueba de la tabla de primos. Es compuesto

### Primo a probar: 1267645764525059029375087808517
> El primo no pasa la prueba de la tabla de primos. Es compuesto
```

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

```
### Primo a probar: 1267645764525059029375087808551
> El primo pasa la prueba de la tabla de primos.
> El primo pasa el test de Miller-Rabin!
```

```
### Tamano bits indicado: 100
### Probabilidad error indicada: 0.100000
### Numero generado: 1267645764525059029375087808551
### Nuestro resultado: Es probablemente primo
### Resultado GMP: Es probablemente primo
### Probabilidad equivocacion: 0.000000
### Pasa el test: 50 veces de 50 tests realizados

Se ha liberado correctamente la memoria...
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto/p3
c
```

A partir de aquí, hemos añadido un cálculo del tiempo de ejecución para comparar los rendimientos de los cálculos de primos en tamaños grandes:

**Laura Sánchez Herrera**  
**Daniel Mateo Moreno**  
**Pareja: 5**

```
## Tamaño bits indicado: 1024
## Probabilidad error indicada: 0.10000
## Número generado: 179769313486231596772930518933685978393264195667857366438478854582414296737200722990543256686085206220299374275925801172222643202991601957944608479
24806918985527081461896458224960347974442158555213437127278995177515494009542539275249252089991669390769117881765396253991392227065206022278935554174962009899543
## Nuestro resultado: Es probablemente primo
## Resultado GMP: Es probablemente primo
## Probabilidad equivocación: 0.00000
## Pasa el test: 50 veces de 50 tests realizados
T Ejecución: 0.051400 min
Se ha liberado correctamente la memoria...
x541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto/p3
$ -
```

```
## Tamaño bits indicado: 2048
## Probabilidad error indicada: 0.10000
## Número generado: 32317006071310073007148766886699519604441026697154840321303454275246551388678908931972014115209768523200331486569261780362828529757334530976717519
9218052334320603117028733205964111603097528215659067282020744650884562737013575472550531927711437082787036208991180738417472842912825
75583945853025246196410725656555062473716824266602169230994744646496779942504311219924424023883486455341008443957803859480269562492729882298210414179443284991783049
653552176467796163992994214404264289205834680254467064484223218245765878319432687198931506355731899347704623871703608592585153538359
## Nuestro resultado: Es probablemente primo
## Resultado GMP: Es probablemente primo
## Probabilidad equivocación: 0.00000
## Pasa el test: 50 veces de 50 tests realizados
T Ejecución: 0.382317 min
Se ha liberado correctamente la memoria...
x541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto/p3
```

```
## Tamaño bits indicado: 4096
## Probabilidad error indicada: 0.10000
## Número generado: 1044388814131525066917527107166243825799642490473878038423348328395390797155745684882681193499755834089010671443926283798757343818579360726323608
7813652779455697654370999834031590134383718314428070011855946226376318839397712745672334684344586617496807908705803704071284048740186091144679777835980290065939852
095586845694115358965738757506685878506110171455164429180064037653200186362559288094173725527982023342949293267706372354550048581291959857836521464933569006321728627758
96820282989763818409911918923844061396658487711650859365358101254595889315683714890223593731017019881032339758782043803631768165744757259272487452821422954140454161542
93233720261530035706717831419473236546400474621942297614344175972227054815493797598697989260482234338621685173134164772639320591636731316374132073899244774126000043279
7782360888782785965002361164536698513522234299584069483707672138916581876582759051231724209042337162067330507791093438389137170446046890014214413769680331253
3835861026776746840342402423546103540005274964726870107017027275064912443008539609091618732266332356476782345002842019738244579217043308085050892439078402844795580767
4822338377311062869955039269813447181497828743065839362784586649059091022649
## Nuestro resultado: Es probablemente primo
## Resultado GMP: Es probablemente primo
## Probabilidad equivocación: 0.080000
## Pasa el test: 46 veces de 50 tests realizados
T Ejecución: 2.965117 min
Se ha liberado correctamente la memoria...
x541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto/p3
```

Debido al excesivo tiempo de ejecución con la generación de un número de 8096 bits y 50 tests, hemos reducido el número de tests a 10 y vamos a probar cómo afecta el parámetro p en los resultados:

```
## Tamaño bits indicado: 8096
## Probabilidad error indicada: 0.90000
## Número generado: 13767176986125814406244959806100101138086925999656783974125653256882989462717828645821928596225616508720429751611910497941179138743555168121256241
161952102020069716966988602039975965587411150165161934020297964443756189668047156486062368146491296062572942703075642509739780313772183021466354042363133122208242048089
02037728060812498116441580546840194673957192408389412901204524365071892894664585288289360998451302621158359160094958849393177997475598465161021346651386877525864
8969808543739645327918677562254539978233803976185247935734572322992717555943161722427140293397532170542686369621192359802197387240
6701525890686869900938645986174904420284917539484068897663703829069131005408924551447161004195022646181619283006845490668180163688791951139203394620770549488520787571862
697764003957339717367618465338758869914683497482640944818802510365816194383317094120348573477724103955878606160475172531602146825751716185897752452238089738853518
6290671517998936555424481395695554314502672412328736906963784039960128376998076305304672564468051881832402644685086052936613313993952240194218078891568863374505510
8461651788002858943116728293659457961081315922966822532966914577781463920294273932511751736213665218302427981849615744963613681848183524744448714431854087915304074958
182529638472199379847962827849890390298454079543660138723495097739618417434636206684117257337939465842117259256258760812545589325457467477000173653313439185967152679996
886304141257264652415922939161706585969475474675547207996579554996033733991414961748970718599812132068249882553210804711312611229716216900540377411313251144664596378996
970361861820472732844275854504680648975351622707657979238580701840077841250178180507847298481525259069888370209159368705878659714901396085164286997588847508277178631646
54056728158002858943116728293659457961081315922966822532966914577781463920294273932511751736213665218302427981849615744963613681848183524744448714431854087915304074958
88638123898883299580964153056766965053811794550415889436548438906161207120018030738044285680506845820611797031853298535062946837870141770474613451675297630118789067490
498231262752594457141665310125707289138123336249263067009409937168582344112177215065471653407192364813244681346261410809180507144368685795196496446192920617612502970723
66950531552428240763452894457077720870800493925978873016511810373886337093791636900201486337562560709002129
## Nuestro resultado: Es probablemente primo
## Resultado GMP: Es probablemente primo
## Probabilidad equivocación: 0.60000
## Pasa el test: 4 veces de 10 tests realizados
T Ejecución: 4.255300 min
Se ha liberado correctamente la memoria...
x541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto/p3
```

**Laura Sánchez Herrera  
Daniel Mateo Moreno  
Pareja: 5**

```

### Tamano bits indicado: 8096
### Probabilidad error indicada: 0.00000
### Numero generado: 137671769860256450100269468575341814887791636896163173848570595574052315286898075577157367536624492302946721686595362947089264059793286521399123878
0372506147612063810887347278696216616128354878283316902496741495869128224534101658513694388163929044461461138500906232290119571646777991419442896221777632521132114417
089086216513241540846943863061952528533916766086209548783063685145706252741306474227058177497135860086773013988606084662322556441751579027452333804101105783612762893304
98022667107929660010133167387886718899711903871148549267580179454515187299419048807975338781614328504929118641360024233831091185305197474703032042826995894713171071360576011
167240811240438893138634928662752077568234084246609093677352783655513600756857020518904410905384177599977712041408917598379416001879607783494089768149047696225308547234
43787853233215953806739866888876151643068505008435507062248175305682688820480903579062335847016054840580814107522290753959165617306371762143551147196602451092635675
527706336984587975146462385500232988995070841960134807388515597934464396528865617095429420046255975983614745692694890155748981316726058976390887459477698403010485622759
585683308312498899674570611987131903765816933214850704356542566842186355941677520569155816981040422943020748657704042449133895927001266217414224132329638144539388254473
8216145138376322024830786399277401102299713606795468859264311399521740338744178352436203718795428370976391782028822638688953701107370906565233306469736975306174086
20705796546044597411966389519557756722837313974178669926116128027808168874287789269376352003860737765486498943388154976929087201760602182399543855951322278713664039
1339304583214174389055983081865066308796224307754464372633229577966769697392975715956652187907410006118766921242755598159434267928046535066187683076450185120614966408439
372274644265951742666069446148291804541237186343872761571756985534512909205317868172978734734375066768564431827437168495155853752434430219134395254983042424008214556
62105953266643859857173285982988761944281766949233424576247180785709749857154483245281415611727416757159754399022505534975791405998835826317350129353920890
444503249982662255093083403536600981828011477088695413748577200059185104488409072931
### Nuestro resultado: Es probablemente primo
### Resultado GMP: Es probablemente primo
### Probabilidad equivocacion: 0.00000

```

```

## Tamanio bits indicado: 8096
## Probabilidad error indicada: 0.400000
## Numero generador: 137671769861258144062449598061001011380869259996567839741256532568882989462717828645821928596225616508720429751611910497941179138743555168121256241
16195201202006719669880263997596558741115016516193402029796444375610668047156486062368146491296062572942703075642509739780313772183021466354042578693586058081126958
029515715848461244023995578120168855552094738962282230074588116480698505109691512952589758905978135219899981222638838091580601339841842516134184164331686134792968367
84148184111344061514115701942977322510974984463210683954487875916521058147001330181682526253498132862416112659489897843726110109184422149894200580774024194093079065
680523894768888054749529375298245356515077389708846213548338078688371199157409464955148082462457593497202255512550936627503761859352530425142993118422609562618962460044
246741287094406812352069944679463181325162766956892710367030721770661514228656571422482425366917867763116091619778065875946316059501890681589502444180816430735316341591524
968859661632789445958993424979618043583866628270126669377379445350267614182159133746379763944959497717011623946263570091327692379149407283309759265554151573983494493721
5435333605192688365750028515984512692290462365864581733328919254755165962573053405189545348483144367996443003741762240175081880320875451629017027862646790746695955217
0498942370767618875927113628611826967787867001189045110069202672843618493234249527166078977112313554263321061791285688133378942735336667212447003495871398279448
306709515365727463745930332936849140430502797395835316158644352678732725364957126954667668915474745821572853211071103052406208321979689655129774543126841075667889433
12820461518854992384329563990247162118176140882329876351761762185400223043964348318932787374530641995458610233611928955847941216023193882611739549477543301987455555
1811939270785887805088887619495306273187642209662303987829965733443688488712318651034713411864226622961344514180192414622732310351985712543221192729581562504344
6320371734323890676675738837087542024880759056753154121921529021488939829567472910398161984412000985287113
## Nuestro resultado: Es probablemente primo
## Resultado GMP: Es probablemente primo
## Probabilidad equivocacion: 0.000000
## Pasa el test: 10 veces de 10 tests realizados
T Ejecucion: 10.565367 min
Se ha liberado correctamente la memoria...

```

```

### Tamanio bits Indicado: 8096
### Probabilidad error indicada: 0.200000
### Numero generador: 1376717698612581440624495980610010113808692599965678397412565325688829894627178286458219285962255205525346294378365633356515947845938793552916495
44846447438224841037852302176151862016660947477236556319262790758333674889401176087632086770974491173166912352897143349354221887940629076238955031656195593024558174
4442186693487720642862296251804696204462280800318876568860768304106997951166357057703528095799909123797868187410242530608987536089579749545952100135926537488004288308376
000372218801168672227655687612741032065721954942603050667390000985488910675825873989324829446631441700530116012308959154452746724126284414349491208074294674544710084
884344767167170115930168204534365498026107857475396129660120421766479568044619908542654924448832810700919063990443984280042162759148707763913625010192700137249212014
355158616999318996018183774116616254012091552656823845454604899610108550056436902740502455327145192293101370172715802337631713805491623989221764842271690503890107309308
7042786863020679997196318786959275073592367330863471097424049014822262392217729203526725532849072628653398106726459266843598292
482902888185946897556583190856889944821362615282179043618146750956404436884581796465657422341773079608128972590252193689530964691483832579248866470788234282385057375
9589895953113106598981441802086611388565406683875132496979429942953626117755007176496981182078357691446033488989969697058751182068939114468546787519225513703618459825635
696711886679891216931641808388283848326134180783610686818814827839075573388565842195991434862818365615994190048098683794969809776411502123500763150141099593692462562
78737839095095397320523680731627504039855323407834760011276502431452980120594983016412688691409079096343302327692233890133360514658685163952705467999141374968
621815375858986495600604088128309567361673569574309034747039611383330383333784492015915156569824313170322734932280192211388662687336524715718539379639599241830501436048
70789963300180219690324379680262799623291073488321529214772076761418487565962364021235243136107350063289770938440709400064216475174650934502262369829377680548085
17812548400935642704596378587909288217812579494821617670274323901987513519119722802197864191568276735167919096373356658735316797615588981457787180729126131253
45206767031933594571615407345338834217225286191585642865907499170756669683967067786995363696592647901676711
### Nuestro resultado: Es probablemente primo
### Resultado GMP: Es probablemente primo
### Probabilidad equivocacion: 0.00000
### Pasa el test: 10 veces de 10 tests realizados
T Ejecucion: 4.928133 min

Se ha liberado correctamente la memoria...

```

Se puede apreciar que el programa tarda más tiempo en encontrar un primo válido que satisfaga el porcentaje de error (indicado por argumento) cuanto menor sea este, aunque han habido ocasiones en las que ha tardado poco en encontrar un número probable primo grande, ya que el programa genera un número aleatorio al inicio distinto en cada ejecución, dándose diversas situaciones donde el programa crea un número más o menos lejano al primo que buscamos. Sin embargo, si esta probabilidad es muy grande, el programa va a encontrar rápidamente un primo que

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

consiga pasar un suficiente número pequeño de veces el test de Miller-Rabin, como podemos ver en el primer caso con probabilidad 90% que ha devuelto un primo que ha fallado 6 de los 10 tests ejecutados, y puede llegar a considerarse un probable primo débil. Pero en los demás, cuanto menor es este porcentaje, más tarda, pero devuelve un primo que ha pasado los tests muchas más veces, incluso pasando los 10 de 10 tests tratados. Por lo tanto, una elección correcta de la probabilidad de equivocación implica una mayor seguridad en la elección de un número grande como supuesto primo, aunque suponga de manera general un mayor tiempo de ejecución.

### C. Factorización del módulo del RSA mediante el conocimiento de d (A. las Vegas)

En este apartado nos han pedido desarrollar un programa, que conociendo el exponente de descifrado ( $d$ ) y obviamente la clave pública ( $e, n$ ), sea capaz de calcular los dos números primos que componen el módulo del RSA  $n = p * q$ .

Para ello hemos desarrollado el programa vegas, con los ficheros **vegas.c** y su librería **vegas.h**. Los argumentos que recibe vegas por terminal son los siguientes:

**./vegas -b numbits**

-b: indica el número de bits que tienen los dos primos  $p$  y  $q$ .

Inicialmente, como en todo los programas que hemos realizado, comprobamos que los argumentos introducidos por terminal sean correctos.

Para calcular  $n$ , primero necesitamos generar los dos números primos de **numbits** que van a componer el módulo del RSA. Hemos creado una función **genera\_primo\_n\_bits()** la cual genera de manera aleatoria un número de numbits y después comprueba que ese número sea primo haciendo uso del algoritmo Miller-Rabin explicado en el anterior apartado. Con esa función obtenemos los valores de  $p$  y  $q$ , y por lo tanto el producto de estos dos valores es  $n$ .

Una vez obtenido el valor de  $n$ , para completar la clave pública necesitamos el valor de  $e$ . Este valor cumple  $1 < e < \varphi(n)$  y debe ser coprimo con  $\varphi(n)$ , es decir,  $\text{mcd}(e, \varphi(n)) = 1$ . Vamos probando valores de  $e$  en ese intervalo y después utilizando el algoritmo de Euclides Extendido comprobamos que sea coprimo con  $\varphi(n)$  y en caso de serlo devuelve el inverso multiplicativo de  $e$  en  $\varphi(n)$  que se trata del valor de  $d$ , el exponente de descifrado.

**Laura Sánchez Herrera**

**Daniel Mateo Moreno**

**Pareja: 5**

Por lo tanto, ya tenemos todos los valores necesarios para hallar los dos números primos que componen el módulo del RSA. Para ello, utilizamos el **Algoritmo de las Vegas**.

El Algoritmo de las Vegas tiene dos respuestas posibles, no responde, en caso de no encontrar ninguno de esos dos números. Y si responde, es el caso de encontrar uno de los dos que puede ser p o q. En el caso de responder, p o q es el valor que devuelve y el otro valor de p o q es n entre el valor que devuelve ( $n = p * q \rightarrow p = n / q$  y  $q = n / p$ ).

Este algoritmo lo hemos implementado en una función aparte, llamada **algoritmo\_vegas()**. Lo primero que hacemos es calcular **k** y **m** que cumplen que  $e^{d-1} = 2^k * m$ . Generamos un número aleatorio **a** que cumple  $1 < a < n - 1$ . Tras esto, si  $\text{mcd}(a, n) > 1$  entonces ha encontrado el resultado de p o q que es el  $\text{mcd}(a, n)$ . En caso contrario, calculamos  $x = a^m \bmod n$  y si  $x = 1$  o  $x = n-1$  entonces el algoritmo no responde. Si es distinto a esos valores, el algoritmo continúa.

Llegado a este punto, comenzamos el bucle que se ejecuta de 1 a k-1. En cada iteración se calcula  $x = x^2 \bmod n$  y si  $x = 1$ , devuelve como valor de p o q  $\text{mcd}(y+1, n)$  (y contiene el valor de x de la anterior iteración) y si es  $x = n-1$  ( $-1 \bmod n$ ) devuelve no responde.

Si tras pasar todas las iteraciones x no es ni 1 ni n-1, devuelve como valor de p o q  $\text{mcd}(x+1, n)$ , x es el valor de x al salir del bucle.

Para calcular los mcd hemos utilizado el algoritmo de Euclides, tanto este algoritmo como el algoritmo de Euclides Extendido están implementados con gmp en el fichero **euclides\_ext.c** y su librería **euclides\_ext.h**.

Ejemplo de ejecución con 5000 bits:

**Laura Sánchez Herrera  
Daniel Mateo Moreno  
Pareja: 5**

Ejemplo de ejecución con 32 bits para ver más fácilmente para la vista que funciona.

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P3$ ./vegas -b 50
### p generado: 26479
### q generado: 24841
### n: 657764839
### e generado: 325427017
### d generado: 378995353
> p o q hallados!
### p o q: 24841
### p o q: 26479

Se ha liberado correctamente la memoria...
laura@Universidad:~/Escritorio/CUARTO/cripto/P3$
```

Como vemos los resultados de  $\rho$  y  $q$  hallados son correctos en ambos ejemplos.