

Práctica 4. Uso de OpenSSL para Cifrado de Mensajes y Ficheros

Primera Parte

1. Usa la página de manual de openssl para obtener información sobre la forma de usar esta herramienta para cifrar y descifrar ficheros.

OpenSSL es un conjunto de herramientas de criptografía que implementa los protocolos de red Secure Sockets Layer (SSL v2/v3) y Transport Layer Security (TLS v1). Puede utilizarse para:

- » Creación y gestión de claves privadas, claves públicas y parámetros
- » Operaciones criptográficas de clave pública
- » Creación de certificados X.509, CSRs y CRLs
- » Cálculo de compendios de mensajes
- » Cifrado y descifrado con cifradores
- » Pruebas de cliente y servidor SSL/TLS
- » Manejo de correo firmado o cifrado S/MIME
- » Solicitudes, generación y verificación de sellos de tiempo

Para cifrar datos podemos valernos de algoritmos de clave simétrica (misma clave para cifrar y descifrar) o de clave asimétrica (diferentes claves para cifrar y descifrar). Podemos ver que algoritmos podemos utilizar con Openssl con el comando “openssl -help”. Como vemos openssl soporta varios algoritmos de clave simétrica como “DES”, “AES” o “CAMELLIA” entre otros.

```
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      aria-128-cbc      aria-128-cfb
aria-128-cfb1    aria-128-cfb8    aria-128-ctr      aria-128-ecb
aria-128-ofb     aria-192-cbc     aria-192-cfb      aria-192-cfb1
aria-192-cfb8    aria-192-ctr     aria-192-ecb      aria-192-ofb
aria-256-cbc     aria-256-cfb     aria-256-cfb1     aria-256-cfb8
aria-256-ctr     aria-256-ecb     aria-256-ofb      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       cast5-cfb        cast5-ecb
cast5-ofb      des            des-cbc          des-cfb
des-ecb         des-ede        des-ede-cbc      des-ede-cfb
des-ede-ofb     des-ede3       des-ede3-cbc     des-ede3-cfb
des-ede3-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       sm4-cbc          sm4-cfb
sm4-ctr        sm4-ecb        sm4-ofb
```

Para hacer uso de estos algoritmos nos valdremos del siguiente patrón para elaborar correctamente el comando.

openssl *comando* [*opciones del comando*] [*argumentos del comando*]

Para concretar más, en el siguiente ejercicio haremos una serie de ejemplos.

2. Experimenta con diversos algoritmos (AES, DES, CAMELLIA,) y modos de cifrado (ECB, CFB, CBC). Utiliza diferentes ficheros de entrada (texto y binarios) y con diferentes claves y vectores de inicialización.

Primero creo un fichero de texto con información en su interior y posteriormente lo cifro usando el algoritmo AES-256 en los tres modos de cifrados pedidos.

```

alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ ls
texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ openssl enc -aes-256-cbc -pas
s pass:1234 -p -in texto.txt -out cbccifrado.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=66B09459F6B899DA
key=F43A3C725C467E8CB9EED22EE5DFA68366244FE7C1DDF96FED566AAC7673EE85
iv =DE4661301710BD3312F1DA1491F4C81C
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ openssl enc -aes-256-cfb -pas
s pass:1234 -p -in texto.txt -out cfbcifrado.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=899AEEBBDE899015
key=99E62F5E87E93B9B86312DD276DE1769CACABDFE685FC049058308277EA5282C
iv =613337BAB095DF70C9FA8B016CED9FD1
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ openssl enc -aes-256-ecb -pas
s pass:1234 -p -in texto.txt -out ecbcifrado.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=D5251C035F344B5A
key=408BD7A1CEE1920F340F6674E64E4D24076D91F143566210099BFE56B4EB178E
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ ls
cbccifrado.enc cfbcifrado.enc ecbcifrado.enc texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$

```

Tras haber cifrado el archivo de texto, ahora crearemos un fichero binario y procederemos a cifrarlo con los mismos algoritmos que el primer fichero

```

alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ xxd -b texto.txt binario.bin
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ ls
binario.bin  cbccifrado.enc  cfbcifrado.enc  ecbcifrado.enc  texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ openssl enc -aes-256-cbc -pas
s pass:1234 -p -in binario.bin -out cbcbincifrado
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=7CE8CAEE7CBAE76E
key=DEB4464A5466F1C9F73EDF933C012EB02BD98CEB3B775F7B168782E9547A6DDB
iv =AA9BFA3E5D17D13E3614EA96828FD11C
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ openssl enc -aes-256-cfb -pas
s pass:1234 -p -in binario.bin -out cfbbincifrado
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=EF68CC3810E54319
key=E4A32C3F07A7C7A49E2C0DEEB52EBEF8CFAB81F7A7B5A6D8B506BB48C6127272
iv =6237B731DB3A7E0C26B479D6621FFB04
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ openssl enc -aes-256-ecb -pas
s pass:1234 -p -in binario.bin -out ecbbincifrado
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=8F7CAB7AEA0E2CB2
key=7EDBD1210A2B56D66BF2F224E78AA8A04021FA457F2B2ECC95973966A328A5F3
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$

```

Ahora en vez de AES utilizaremos DES, también en las tres variantes especificadas.

```

alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ ls
texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ openssl enc -des-cbc -pass
pass:1234 -p -in texto.txt -out cbccifrado.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=9ABCB31D9221DF58
key=345F5903A42B2E4C
iv =BB03458F814B431E
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ openssl enc -des-cfb -pass
pass:1234 -p -in texto.txt -out cfbifrado.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=334AB08B85EEE8F9
key=B990E71A2B096258
iv =07B2F76667EE3B52
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ openssl enc -des-ecb -pass
pass:1234 -p -in texto.txt -out ecbifrado.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=E1B8D77532FD2FD8
key=568456F922D0A38F
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ ls
cbccifrado.enc  cfbifrado.enc  ecbifrado.enc  texto.txt

```

Repiremos el proceso con un fichero binario

```

alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ xxd -b texto.txt binario.b
in
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ ls
binario.bin cbcifrado.enc cfbifrado.enc ecbifrado.enc texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ openssl enc -des-ecb -pass
pass:1234 -p -in texto.txt -out ecbbincifrado
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=77CD64DAB4764DF3
key=D73645D90127425B
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ openssl enc -des-cfb -pass
pass:1234 -p -in texto.txt -out cfbincifrado
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=DD7856BF69DC1E1D
key=781E428E015DDFF0
iv =44C14E2926011C8C
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ openssl enc -des-cbc -pass
pass:1234 -p -in texto.txt -out cbcincifrado
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=C1222278725E47FF
key=1E108AC4920E11CE
iv =C8F76FD4B617D0A8
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ ls
binario.bin cbcifrado.enc cfbifrado.enc ecbifrado.enc
cbcbincifrado cfbincifrado ecbbincifrado texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$

```

Al final de la imagen podemos observar todos los ficheros generados usando DES.

Por último, utilizaremos el algoritmo CAMELLIA-256 con los tres modos indicados

```

alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ ls
texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ openssl enc -camellia
-256-cbc -pass pass:1234 -p -in texto.txt -out cbcifrado.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=9B0A315E1034D772
key=1BAE5458099787BEF2FDB0DD5AA5C3B7D5E3484400CE94B23E43D208C04472E
iv =9C4EB96359351E4707E2A16BC9AC4110
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ openssl enc -camellia
-256-cfb -pass pass:1234 -p -in texto.txt -out cfbifrado.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=895221613D34EA24
key=E1C718CF107DB48B1C5C8B601559AA2C71CDFC5979D78C9182ABD62CE80D3D26
iv =8E50A27EAE9427D09E0C9A99FCF3C523
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ openssl enc -camellia
-256-ecb -pass pass:1234 -p -in texto.txt -out ecbifrado.enc
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=502FEE0F27A35A15
key=F023D34AC3C0E2CF19F478B990022CCBD94C72CBB4D3E81E63556E91B6B1D0EB
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ ls
cbcifrado.enc cfbifrado.enc ecbifrado.enc texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$

```

Ahora lo haremos con el fichero en binario.

```

alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ xxd -b texto.txt binario.bin
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ ls
binario.bin  cbcifrado.enc  cfbifrado.enc  ecbbifrado.enc  texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ openssl enc -des-cfb
-pass pass:1234 -p -in texto.txt -out cfbfbincifrado
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=38737E6F052EEF97
key=4CA9BB8D3147D32A
iv =6155CD96543DD0BF
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ openssl enc -des-cbc
-pass pass:1234 -p -in texto.txt -out cbcfbincifrado
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=551C78120387948B
key=D86CE29FE82C7D62
iv =145E413222D02C80
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ openssl enc -des-ecb
-pass pass:1234 -p -in texto.txt -out ecbbincifrado
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=C6CEF7F76436A4AD
key=28ADAED7D4890DAC
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ ls
binario.bin  cbcifrado.enc  cfbifrado.enc  ecbbifrado.enc
cbcfbincifrado  cfbfbincifrado  ecbbincifrado  texto.txt
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$

```

Podemos observar todos los ficheros generados usando CAMELLIA.

Nota: Como se puede apreciar para cada algoritmo hay una carpeta distinta en la que se almacenan los ficheros generados. El fichero de texto plano y el binario son el mismo para todas.

3. **Obtenga información sobre el concepto de entropía de Shannon y elabore un breve informe y discusión sobre el tema. Usando el programa Python (enlaces interesantes). obtenga la entropía de los ficheros usados en el apartado segundo (tanto cifrados como descifrados), así como la del fichero obtenido en el apartado cuarto y la de un fichero que contenga un único byte repetido un número de veces arbitrario.**

Cuando hablamos de entropía nos referimos a una medida de la incertidumbre de una fuente de información. También se puede definir como la media de la cantidad de información que contienen los símbolos usados. Es decir, los símbolos que menos aparezcan son los que más información aportan. En nuestro caso nos quedaremos con la segunda definición.

Para poder calcularla nos serviremos de la siguiente formula:

$$H = -p_1 \log_2(p_1) - p_2 \log_2(p_2) - \dots - p_k \log_2(p_k) = - \sum_{i=1}^k p_i \log_2(p_i)$$

Cada P_n hace referencia a la probabilidad de que ocurra un evento n en concreto. El valor más alto que puede llegar a obtener una entropía es 8.

Para obtener la entropía de cada fichero pedido hhe utilizado el código de Python que se encuentra en la siguiente página web: <https://code.activestate.com/recipes/577476-shannon-entropy-calculation/>

```

# Shannon Entropy of a file
# FB - 201012153
import sys
import math
if len(sys.argv) != 2:
    print 'Usage: file_entropy.py [path]filename'
    sys.exit()
f = open(sys.argv[1], 'rb')
byteArr = bytearray(f.read())
f.close()
fileSize = len(byteArr)
print 'File size in bytes:', fileSize
print

freqList = [0] * 256
for b in byteArr:
    freqList[b] += 1

ent = 0.0
for f in freqList:
    if f > 0:
        freq = float(f) / fileSize
        ent = ent + freq * math.log(freq, 2)
ent = -ent
print 'Shannon entropy (min bits per byte-character):', ent
print
print 'Min possible file size assuming max theoretical compression efficiency:'
print (ent * fileSize) / 8, 'bytes'

```

Ahora sacamos la entropía de todos los archivos pedidos. Primero empezamos con los archivos cifrados con AES.


```

alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ ls
binario.bin  cbccifrado.enc  cfbcifrado.enc  ecbcifrado.enc  texto.txt
cbcbincifrado  cfbbincifrado  ecbbincifrado  shannon.py
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ python3 shannon.py texto.txt
File size in bytes: 5
Shannon entropy (min bits per byte-character): 2.321928094887362

Min possible file size assuming max theoretical compression efficiency:
1.4512050593046013 bytes
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ python3 shannon.py cbccifrado
.enc
File size in bytes: 32
Shannon entropy (min bits per byte-character): 4.875

Min possible file size assuming max theoretical compression efficiency:
19.5 bytes
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ python3 shannon.py cfbcifrado
.enc
File size in bytes: 21
Shannon entropy (min bits per byte-character): 4.201841232302569

Min possible file size assuming max theoretical compression efficiency:
11.029833234794244 bytes
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ python3 shannon.py ecbcifrado
.enc
File size in bytes: 32
Shannon entropy (min bits per byte-character): 4.726409765557392

Min possible file size assuming max theoretical compression efficiency:
18.905639062229568 bytes
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ python3 shannon.py binario.bi
n
File size in bytes: 71
Shannon entropy (min bits per byte-character): 2.117840529445285

Min possible file size assuming max theoretical compression efficiency:
18.795834698826905 bytes
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ python3 shannon.py cbcbcincifr
ado
File size in bytes: 96
Shannon entropy (min bits per byte-character): 6.235902344426088

Min possible file size assuming max theoretical compression efficiency:
74.83082813311306 bytes
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ python3 shannon.py cfbbincifr
ado
File size in bytes: 87
Shannon entropy (min bits per byte-character): 6.112427547547996

Min possible file size assuming max theoretical compression efficiency:
66.47264957958446 bytes
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ python3 shannon.py ecbbincifr
ado
File size in bytes: 96
Shannon entropy (min bits per byte-character): 6.264599089240291

Min possible file size assuming max theoretical compression efficiency:
75.17518907088349 bytes

```

Ahora seguimos con los archivos cifrados con DES.

```

alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ ls
binario.bin  cbcifrado.enc  cfbifrado.enc  ecbifrado.enc  texto.txt
cbcbincifrado  cfbbincifrado  ecbbincifrado  shannon.py
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ python3 shannon.py cbcifrado.enc
File size in bytes: 24
Shannon entropy (min bits per byte-character): 4.501629167387823

Min possible file size assuming max theoretical compression efficiency:
13.50488750216347 bytes
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ python3 shannon.py cfbifrado.enc
File size in bytes: 21
Shannon entropy (min bits per byte-character): 4.297079327540665

Min possible file size assuming max theoretical compression efficiency:
11.279833234794246 bytes
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ python3 shannon.py ecbifrado.enc
File size in bytes: 24
Shannon entropy (min bits per byte-character): 4.501629167387823

Min possible file size assuming max theoretical compression efficiency:
13.50488750216347 bytes
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$

```

```

alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ python3 shannon.py ecbbincifrado
File size in bytes: 24
Shannon entropy (min bits per byte-character): 4.334962500721156

Min possible file size assuming max theoretical compression efficiency:
13.004887502163468 bytes
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ python3 shannon.py cfbbincifrado
File size in bytes: 21
Shannon entropy (min bits per byte-character): 4.297079327540665

Min possible file size assuming max theoretical compression efficiency:
11.279833234794246 bytes

```

```

alejandro@alejandro-VirtualBox:~/Desktop/descifrado$ python3 shannon.py cbcbincifrado
File size in bytes: 24
Shannon entropy (min bits per byte-character): 4.334962500721156

Min possible file size assuming max theoretical compression efficiency:
13.004887502163468 bytes
alejandro@alejandro-VirtualBox:~/Desktop/descifrado$

```

Acabamos con los cifrados con CAMELLIA.


```

alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ ls
binario.bin  cbcifrado.enc  cfbifrado.enc  ecbinifrado.enc  texto.txt
cbcbincifrado  cfbbincifrado  ecbbincifrado  shannon.py
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ python3 shannon.py ec
bifrado.enc
File size in bytes: 32
Shannon entropy (min bits per byte-character): 4.9375

Min possible file size assuming max theoretical compression efficiency:
19.75 bytes
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ python3 shannon.py cf
bifrado.enc
File size in bytes: 22
Shannon entropy (min bits per byte-character): 4.277613436819114

Min possible file size assuming max theoretical compression efficiency:
11.763436951252563 bytes
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ python3 shannon.py cb
cifrado.enc
File size in bytes: 32
Shannon entropy (min bits per byte-character): 4.875

Min possible file size assuming max theoretical compression efficiency:
19.5 bytes
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$

alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ python3 shannon.py ec
bbincifrado
File size in bytes: 24
Shannon entropy (min bits per byte-character): 4.334962500721156

Min possible file size assuming max theoretical compression efficiency:
13.004887502163468 bytes
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ python3 shannon.py cf
bbincifrado
File size in bytes: 22
Shannon entropy (min bits per byte-character): 4.277613436819114

Min possible file size assuming max theoretical compression efficiency:
11.763436951252563 bytes
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$ python3 shannon.py cb
cbincifrado
File size in bytes: 24
Shannon entropy (min bits per byte-character): 4.501629167387823

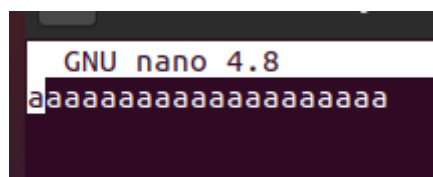
Min possible file size assuming max theoretical compression efficiency:
13.50488750216347 bytes
alejandro@alejandro-VirtualBox:~/Desktop/camelliacifrado$

```

Para terminar y poder comparar de una manera más rápida y eficaz pondremos en una tabla los datos obtenidos de todos los ficheros.

Archivo	File size (bytes)	Shannon entropy	Min posible file size
Texto.txt	5	2.32	1.45
Binario.bin	71	2.118	18.79
AES-CBC	32	4.875	19.5
AES-CFB	21	4.2	11.02
AES-ECB	32	4.72	18.9
AES-CBC (BINARIO)	96	6.23	74.83
AES-CFB (BINARIO)	87	6.11	66.47
AES-ECB (BINARIO)	96	6.26	75.17
DES-CBC	24	4.5	13.5
DES-CFB	21	4.29	11.27
DES-ECB	24	4.5	13.5
DES-CBC (BINARIO)	24	4.33	13
DES-CFB (BINARIO)	21	4.29	11.27
DES-ECB (BINARIO)	24	4.33	13
CAM-ECB	32	4.9375	19.75
CAM-CFB	22	4.277	11.76
CAM-CBC	32	4.875	19.5
CAM-ECB (BINARIO)	24	4.33	13
CAM-CFB (BINARIO)	22	4.27	11.76
CAM-CBC (BINARIO)	24	4.5	13.5

Ahora vamos a ver que pasa con la entropía de un fichero de texto en el que solo hay un tipo de carácter repetido varias veces.



```

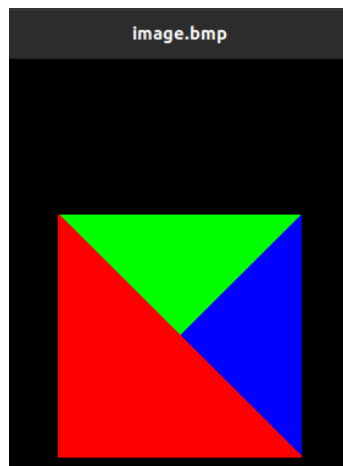
alejandro@alejandro-VirtualBox:~/Desktop/cifrado$ python3 shannon.py rep.txt
File size in bytes: 21
Shannon entropy (min bits per byte-character): 0.2761954276479391
Min possible file size assuming max theoretical compression efficiency:
0.7250129975758401 bytes

```

Como podemos observar la entropía se acerca a 0.

- Obtenga de la red un fichero que contenga una imagen en formato BMP (libre de derechos, a poder ser) y cifrelo usando AES con modos ECB y CBC. Salve copias de la imagen cifrada, sustituya por la cabecera (54 bytes) del fichero original las cabeceras de los ficheros obtenidos y cárguelos en un programa de visualización de imágenes. Comente el resultado. Nota: Para trasladar la cabecera de un fichero in.bmp a otro out.bmp dejando el resto inalterado, se puede usar: 'dd if=in.bmp of=out. Bmp bs=54 count=1 conv=notrunc'.

Obtenemos una imagen en formato ".bmp" libre de derechos.



Ciframos la imagen usando AES-256 con las versiones ECB y CBC

```

alejandro@alejandro-VirtualBox:~/Desktop/image$ openssl enc -aes-256-ecb -pass
pass:1234 -p -in image.bmp -out ecbimage.bmp
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=459030A30E16E54C
key=1859D1EDD5B878B1C022B92A8CDD721AD86BDA4F8F7B50FBE79A687AB9C758E0
alejandro@alejandro-VirtualBox:~/Desktop/image$ openssl enc -aes-256-cbc -pass
pass:1234 -p -in image.bmp -out cbcimage.bmp
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=40F1E0E9F678410F
key=2FBB611B9C0BD02D3E9C23108E93B2EB67C42ACC65ED480393AA1A5E8CD70782
iv =3611EDE6CFBC78EC14588ED167DC26F8
alejandro@alejandro-VirtualBox:~/Desktop/image$ ls
cbcimage.bmp  copias  ecbimage.bmp  image.bmp

```

Después haremos una copia de cada una de las imágenes ya cifradas.

```

alejandro@alejandro-VirtualBox:~/Desktop/image$ cp ecbimage copias/ecbimage
alejandro@alejandro-VirtualBox:~/Desktop/image$ cp cbcimage copias/cbcimage
alejandro@alejandro-VirtualBox:~/Desktop/image$ cd copias/
alejandro@alejandro-VirtualBox:~/Desktop/image/copias$ ls
cbcimage  ecbimage
alejandro@alejandro-VirtualBox:~/Desktop/image/copias$

```

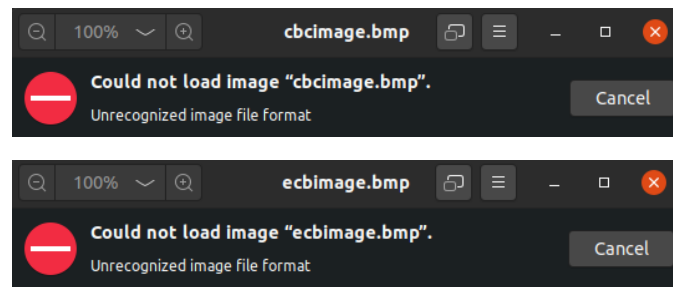
Procedemos a cambiar la cabecera de las copias de los archivos cifrados con las instrucciones dadas del enunciado.

```

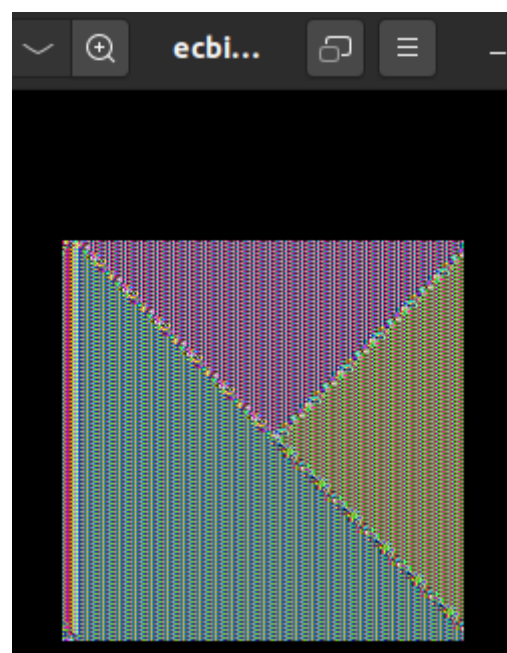
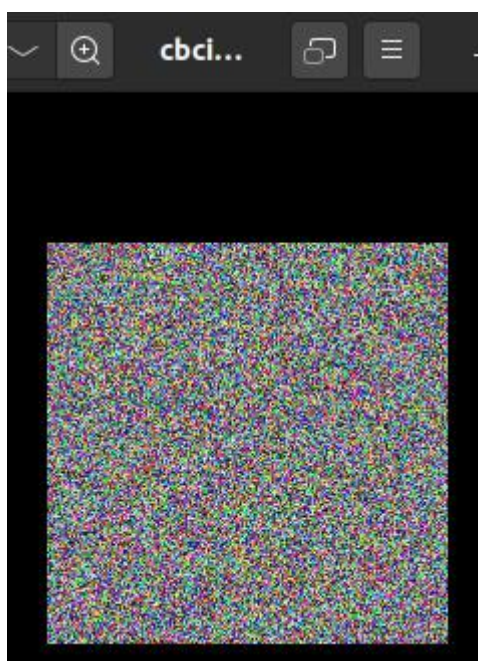
alejandro@alejandro-VirtualBox:~/Desktop/image$ cp cbcimage.bmp copias/cbcimage
.bmp
alejandro@alejandro-VirtualBox:~/Desktop/image$ cp ecbimage.bmp copias/ecbimage
.bmp
alejandro@alejandro-VirtualBox:~/Desktop/image$ cd copias/
alejandro@alejandro-VirtualBox:~/Desktop/image/copias$ ls
cbcimage.bmp  ecbimage.bmp  image.bmp
alejandro@alejandro-VirtualBox:~/Desktop/image/copias$ dd if=image.bmp of=cbcim
age.bmp bs=54 count=1 conv=notrunc
1+0 records in
1+0 records out
54 bytes copied, 0,00283301 s, 19,1 kB/s
alejandro@alejandro-VirtualBox:~/Desktop/image/copias$ dd if=image.bmp of=ecbim
age.bmp bs=54 count=1 conv=notrunc
1+0 records in
1+0 records out
54 bytes copied, 0,00812971 s, 6,6 kB/s

```

Ahora intentamos abrir las imágenes cifradas originales y, como vemos no podemos visualizar nada.



Ahora lo intentamos con las copias modificadas. En ambas nos permite ver algo, en la versión CBC (izq) vemos una imagen borrosa que nada tiene que ver con la inicial. Pero, para sorpresa en la versión ECB (dcha) si que podemos apreciar un parecido con la versión original de la foto. Aunque para imágenes más detalladas que la de ejemplo sería difícil hacerse una idea de la imagen original.



Segunda Parte

1. Selecciona un servidor web público (puedes utilizar el de la UVA) e investiga cuáles son los puertos a nivel de transporte que utiliza para brindar el servicio web. ¿Qué sentido tiene?

De manera genérica se utiliza el puerto 80 tcp, donde se ubica el servicio http. Este servicio establece una conexión entre cliente y servidor no segura. Es decir, este servicio no protege la información que viaja por la red. Por ello hay otro puerto, el 443 tcp, en el que se ubica el servicio https. La diferencia entre http y https es que este último protege la información con cifrado asimétrico (RSA) donde se proporciona un certificado digital. Esto hace que sepamos en todo momento que estamos comunicándonos con el servidor original.

Vamos a comprobar si la web de la Uva tiene estos dos puertos abiertos.

```

alejandro@alejandro-VirtualBox:~$ nmap www.uva.es -p80
Starting Nmap 7.80 ( https://nmap.org ) at 2022-11-10 12:15 CET
Nmap scan report for www.uva.es (157.88.25.8)
Host is up (0.093s latency).

PORT      STATE SERVICE
80/tcp    open  http









Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
alejandro@alejandro-VirtualBox:~$ nmap www.uva.es -p443
Starting Nmap 7.80 ( https://nmap.org ) at 2022-11-10 12:16 CET
Nmap scan report for www.uva.es (157.88.25.8)
Host is up (0.35s latency).
rDNS record for 157.88.25.8: sostenibilidad.uva.es

PORT      STATE SERVICE
443/tcp    open  https

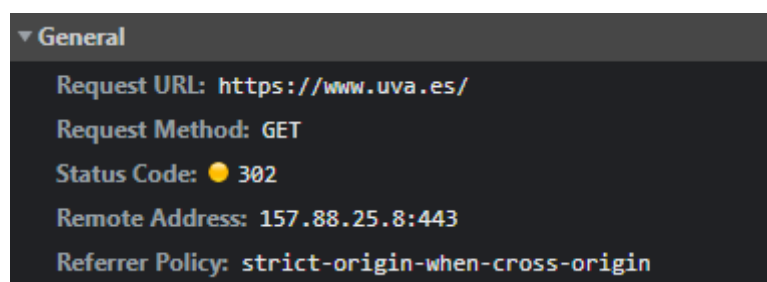
Nmap done: 1 IP address (1 host up) scanned in 4.87 seconds

```

Vemos que ambos puertos están abiertos, ahora vamos a ver qué pasa si usamos el servicio http buscando `www.uva.es`.

Name	Status	Type	Initiator	Size	Time	Waterfall
 www.uva.es	307	docume...	Other	0 B	3 ms	
 www.uva.es	302	docume...	www.uva.es/	1.8 kB	218 ms	
 uva/	302	docume...	www.uva.es/	0 B	2 ms	
 uva/	302	docume...	/export/sites/uv... (disk ca...	3 ms		

Vemos que hay un redireccionamiento de puerto (status 302) donde nos envía al puerto 443 para que la conexión sea segura.



Esta redirección se hace para que los usuarios puedan comunicarse de manera más segura con el host.

2. Analiza el certificado digital presente en el servidor web. Puedes hacer uso de las herramientas **sslsn** y **ssltest** del apartado anterior e indica:

Primero accedemos a los certificados de la pagina web de la Uva. Para ello le daremos al candado ubicado a la izquierda de la barra de búsquedas, a la opción de “La conexión es segura” y por último a la opción “El certificado es válido”.

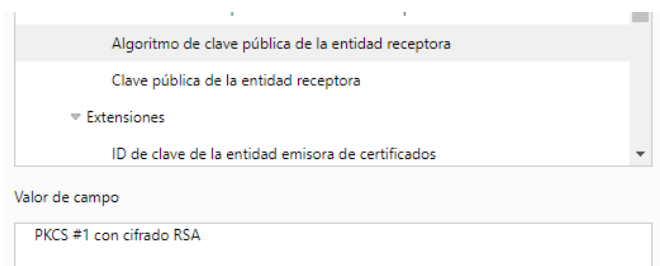


a. Protocolo/s criptográfico y versión utilizado a nivel de transporte.

Analizamos la página de la Uva con **sslsn** y obtenemos que usa el protocolo TLS en sus versiones 1.0, 1.1 y 1.2.

b. Algoritmo de criptografía asimétrica (clave pública) utilizado y longitud de la clave pública.

En el visor del certificado podemos ver ambas cuestiones. El algoritmo utilizado es RSA y la longitud de la clave es de 2048 bits.



Clave pública de la entidad receptora

▼ Extensiones

ID de clave de la entidad emisora de certificados ▼

Valor de campo

Módulo (2048 bits):

```
E4 DC 6A 0E F7 CE 8D 30 55 B0 70 77 79 92 0E 95
47 B8 CC 27 F0 08 E0 BF 47 A3 48 36 16 D1 82 17
BC 0C C8 2B C2 B8 0B 45 54 61 23 FE 0A F3 B9 9F
D7 B2 9B D0 D5 ED 9A 05 2C B4 FB 14 B7 47 AB B5
```

c. Indica la clave pública presente en el certificado digital. ¿Por qué se utiliza esa y no otra?

Se utiliza 65537 como clave pública. Esto se debe a que este número es el número 4 de Fermat y este número sólo tiene dos bits con valor 1, 0x10001 en hexadecimal. Al aplicar el algoritmo de exponenciación rápida en las operaciones de cifrado, donde e actúa de exponente, $C = M^e \bmod n$, debido a que está compuesto de muchos ceros, permite agilizar las operaciones de cálculo [0]. Por tanto, las operaciones de cifrado con la clave pública e en RSA son mucho más eficientes y rápidas que las operaciones de descifrado con la clave privada d .

d. Algoritmo de criptografía simétrica (clave privada) utilizado y longitud de la clave privada.

Se utiliza AES_256 como se puede ver en el certificado mostrado por la web. Al usar AES_256 la longitud es de 256 bits.

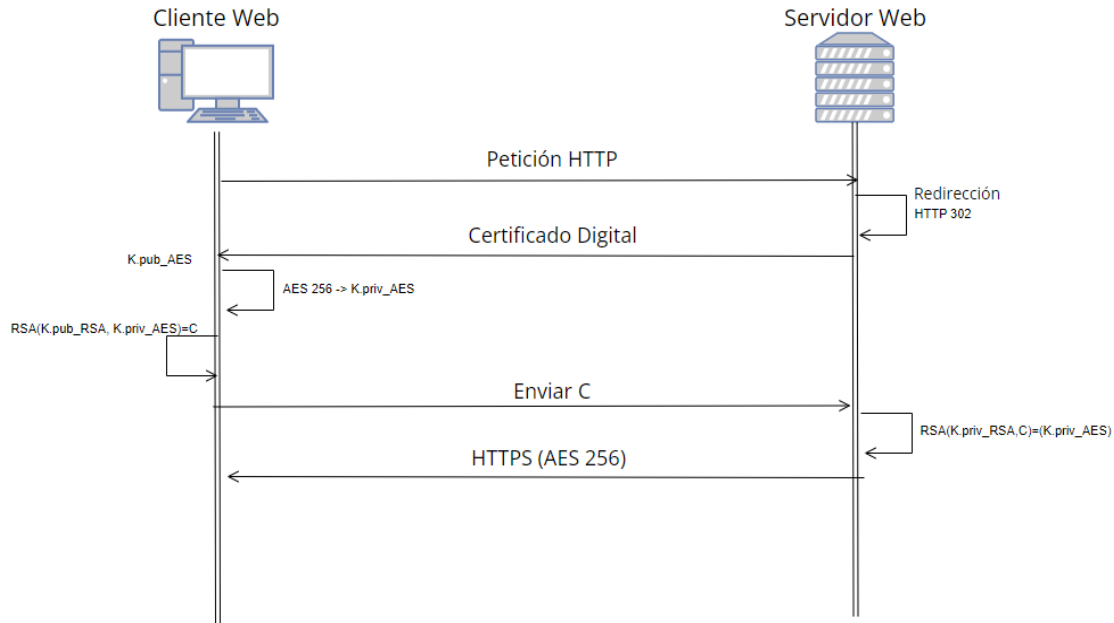
e. Algoritmo de firma digital utilizado en el certificado digital.

Algoritmo de firma de certificado ▼

Valor de campo

PKCS #1 SHA-384 con cifrado RSA

3. Explica con un diagrama de secuencia por qué en un certificado digital se usa criptografía de clave pública y privada.



En el diagrama quedan plasmados el conjunto de sucesos que ocurren cuando se intenta acceder a una página a través del puerto 80 TCP pudiéndolo hacer por el 443 TCP. Al contactar con el servidor, este hará un redireccionamiento de puertos y devolverá la solicitud de certificado digital con la clave pública establecida. El cliente, tras codificar su clave privada con AES, usa el algoritmo rsa junto con las claves públicas y privadas para crear el mensaje protegido. Tras enviar el mensaje, el servidor puede descifrar la clave privada del cliente a través de la suya propia, pudiendo acceder así a la información del mensaje. Tras estos pasos queda establecida la conexión https con el servidor. Es decir, en un certificado digital se usa criptografía de clave pública y privada para evitar que otros dispositivos no autorizados se conecten a la red.

4. Utiliza la herramienta OpenSSL para la generación de un certificado digital que tenga una longitud de clave de 1024 bits y analízalo con ssllscan y ssltest.

Primero creamos una carpeta llamada “rsa” en el directorio opt. En este nuevo directorio crearemos la clave privada de Alice con la instrucción “openssl genrsa -out alice.key 1024”.

El 1024 indica el tamaño en bytes de la clave generada. Normalmente las claves tienen un mínimo de 2048 o 4096 bytes, pero como es un ejemplo solo utilizaremos 1024.

```

alejandro@alejandro-VirtualBox:~$ sudo mkdir /opt/rsa
[sudo] password for alejandro:
alejandro@alejandro-VirtualBox:~$ cd /opt/rsa/
alejandro@alejandro-VirtualBox:/opt/rsa$ sudo openssl genrsa -out alice.key 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
  
```

Una vez creada la clave podemos ver que si intentemos leerla con el comando “cat” no obtendremos nada de información ya que la clave está cifrada.

```

alejandro@alejandro-VirtualBox:/opt/rsa$ sudo cat alice.key
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQD0yYWZ1faSVQkgfECU3W6Uc+G0gd8dehBs/HDaCpt4NDy0qEwZ
k3T7snF3I4d059ZC4ITbnFhqqK0H0QGE2T4ne3/mjI6Zo8bEl5wsQTzg5wFRnUvt
XA3yNaLJqRLj9jkNWE5ArOCeDkKsoijYrRutv3d0tCiWjs09CcJ5nXFUMQIDAQAB
AoGAJxJeww1xlH6qxc+UaExam4Q1xzZ00N4CgoVEUX6Vl6XPHpd6IY8CGJjD5a35
4AfgLSy9Db3B2wXLK8Uyq0Zc/sPPIQ74PnPYpblghb9FWYiTTeq6mym0czrdilG1
eWko1SIB2fTVy6zDoA80jP08XTRH+b7Xn7j90kc6ZBCWTECQQDuXPrEl34yZoLP
24aE237sCEfU8Vo1r6wTgR9wh3YPIQBjSdxx0JjpJlxvu0mQmsEXG/miYHMPGuFY
lxXDVArNAKEA3hzv2otjtzA4egklcftZ2vMrvGxTX2Q9rQIYTFHHJ9u//8pAgZ
maQUbvKG6Is189pJqVTkeZ739hdp071I/QJBAMYApq63yTTEwtU3Xj7utjgLiN/y
vSTrkn2C3UL0w7U9jLztloMqqJrEKvUFds6QPuWbb3ra886oGW5p0sI3BeECQBF0
lAAKWqEIUjgNRF36w8cQKvtJlUak0iFNLnSj+dYdjZJcMWA3HsP7qHT75myanHf5
HTcI7Bg/DWAZQ0cahoUCQGiqBkyhM6o/E57dqLrAY/qT+fLAYvkaJaVhk33cFACL
I8WJoZ0NyRI3w+iSNIhid7PS7sSkPh5T/59ufbP4QkI=
-----END RSA PRIVATE KEY-----

```

Para obtener información hay que hacer uso de openssl, en concreto utilizaremos el comando “sudo openssl rsa -text -in clave”

```

alejandro@alejandro-VirtualBox:/opt/rsa$ sudo openssl rsa -text -in alice.key
RSA Private-Key: (1024 bit, 2 primes)
modulus:
 00:ce:c9:85:99:d5:f6:92:55:09:20:7c:40:94:dd:
 6e:94:73:e1:8e:81:df:1d:7a:10:6c:fc:70:da:0a:
 9b:78:34:3c:b4:a8:4c:19:93:74:fb:b2:71:77:23:
 87:4e:e7:d6:42:e0:84:db:9c:58:6a:80:ad:07:39:
 01:84:d9:3e:27:7b:7f:e6:8c:8e:99:a3:c6:c4:97:
 9c:2c:41:3c:e0:e7:01:51:9d:4b:ed:5c:0d:f2:35:
 a2:c9:a9:12:e3:f6:39:0d:58:4e:40:ac:e0:9e:0e:
 42:ac:a2:28:d8:ad:1b:ad:bf:77:4e:b4:28:96:8e:
 cd:3d:09:c2:79:9d:71:54:99
publicExponent: 65537 (0x10001)
privateExponent:
 27:12:5e:c3:0d:71:94:7e:aa:c5:cf:94:68:4c:5a:
 9b:84:35:c7:36:74:d0:de:02:82:85:44:51:7e:95:
 97:a5:cf:1e:97:7a:21:8f:02:18:98:c3:e5:ad:f9:
 e0:07:e0:2d:2c:bd:0d:bd:c1:db:05:cb:2b:c5:32:
 ab:46:5c:fe:c3:cf:89:0e:f8:3e:73:d8:a5:b9:60:
 85:bf:45:59:88:93:b4:4a:ba:9b:29:b4:73:3a:dd:
 8a:51:b5:79:69:28:d5:22:01:d9:f4:d5:cb:ac:c3:
 00:0f:34:8c:fd:3c:5d:34:47:f9:be:d7:9f:b8:fd:
 3a:64:10:96:4d:31
prime1:
 00:ee:5c:fa:c4:97:7e:32:66:82:cf:db:86:84:db:
 7e:ec:08:47:d4:f1:5a:35:af:ac:13:81:1f:70:87:
 76:0f:21:00:63:49:dc:f1:38:98:e9:26:5c:6f:bb:
 49:90:9a:c1:17:1b:f9:a2:60:73:29:19:47:d8:97:
 15:c3:54:0a:cd

```



```

prime2:
  00:de:16:6f:da:8b:63:b7:30:38:7a:09:25:95:c7:
  ed:67:6b:cc:ae:f8:06:c5:35:f6:43:da:d0:21:84:
  c5:1c:72:7d:bb:ff:fc:a4:08:19:99:a4:14:6e:f2:
  86:e8:8b:35:f3:da:49:a9:54:e4:79:9e:f7:f6:17:
  69:d3:bd:48:fd
exponent1:
  00:c6:00:a6:ae:b7:c9:34:c4:c2:d5:37:5e:3e:ee:
  b6:38:0b:88:df:f2:bd:24:eb:90:dd:82:dd:42:f4:
  c3:b5:3d:8c:bc:ed:96:83:2a:a8:9a:c4:2a:f5:05:
  76:ce:90:3e:e5:9b:6f:7a:da:f3:ce:a8:19:6e:69:
  d2:c2:37:05:e1
exponent2:
  11:74:94:00:0a:5a:a1:08:52:38:0d:45:fd:fa:c3:
  c7:10:2a:fb:49:95:46:a4:d2:21:4d:2e:74:a3:f9:
  d6:1d:8d:92:5c:31:60:37:1e:c3:fb:a8:7b:7b:e6:
  6c:9a:9c:77:f9:1d:37:08:ec:18:3f:0d:60:33:40:
  e7:1a:86:85
coefficient:
  68:aa:06:4c:a1:33:aa:3f:13:9e:dd:a8:ba:c0:63:
  fa:93:f9:f9:40:62:f9:1a:25:a5:61:93:7d:dc:14:
  00:8b:23:c5:89:a1:9d:0d:c9:12:37:c3:e8:92:34:
  88:62:77:b3:d2:ee:c4:a4:3e:1e:53:ff:9f:6e:7d:
  b3:f8:42:42
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDOyYWZ1faSVQkgfECU3W6Uc+G0gd8dehBs/HDaCpt4NDy0qEwZ

```

Con esta información podemos ver que el número del exponente (“e”) es 65537 y que la “n” está formada por el número que contiene “modulus”. También podemos ver los números primos usados para formar la clave. Tras obtener la clave podemos empezar a crear el certificado digital. Para ello nos valdremos del siguiente comando.

```

alejandro@alejandro-VirtualBox:/opt/rsa$ sudo openssl req -new -key alice.key -
out alice.csr

```

Nos pedirá que rellenemos cierta información de manera no obligatoria.

```

Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Valladolid
Locality Name (eg, city) []:Valladolid
Organization Name (eg, company) [Internet Widgits Pty Ltd]:4ck.com
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:www.4ck.com
Email Address []:admin@4ck.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

```

Este comando sirve para obtener la solicitud del certificado digital. Ya tenemos la clave privada y la solicitud del certificado.

```

alejandro@alejandro-VirtualBox:/opt/rsa$ ls
alice.csr  alice.key

```

Mediante el comando “`sudo openssl req -noout -text -in solicitud`” “enviamos” la solicitud del certificado digital.

```

alejandro@alejandro-VirtualBox:/opt/rsa$ sudo openssl req -noout -text -in alic
e.csr
Certificate Request:
  Data:
    Version: 1 (0x0)
    Subject: C = ES, ST = Valladolid, L = Valladolid, O = 4ck.com, CN = www
.4ck.com, emailAddress = admin@4ck.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public-Key: (1024 bit)
      Modulus:
        00:ce:c9:85:99:d5:f6:92:55:09:20:7c:40:94:dd:
        6e:94:73:e1:8e:81:df:1d:7a:10:6c:fc:70:da:0a:
        9b:78:34:3c:b4:a8:4c:19:93:74:fb:b2:71:77:23:
        87:4e:e7:d6:42:e0:84:db:9c:58:6a:80:ad:07:39:
        01:84:d9:3e:27:7b:7f:e6:8c:8e:99:a3:c6:c4:97:
        9c:2c:41:3c:e0:e7:01:51:9d:4b:ed:5c:0d:f2:35:
        a2:c9:a9:12:e3:f6:39:0d:58:4e:40:ac:e0:9e:0e:
        42:ac:a2:28:d8:ad:1b:ad:bf:77:4e:b4:28:96:8e:
        cd:3d:09:c2:79:9d:71:54:99
      Exponent: 65537 (0x10001)
    Attributes:
      a0:00
    Signature Algorithm: sha256WithRSAEncryption
      2a:52:38:c0:0a:b7:f8:78:99:ed:7b:82:8f:7e:1a:47:c5:7a:
Show Applications 5:21:37:ec:b2:42:30:99:fb:da:92:c3:b7:71:43:26:
7d:c5:11:bc:2b:dd:f9:9d:4c:98:4f:b9:53:c5:76:69:f2:f1:
34:50:bb:3a:33:50:ef:53:53:41:d6:d0:1a:3a:30:5a:5b:b4:

```

Después de ejecutar el comando podemos ver la información que este ha enviado en la solicitud. Se ve que envía datos públicos de la clave privada de Alice.

Para obtener definitivamente el certificado que siga el estándar x509 y que tenga 1 año de validez nos valdremos del comando que se ve en la captura.

```

alejandro@alejandro-VirtualBox:/opt/rsa$ sudo openssl x509 -req -days 365 -in a
lice.csr -signkey alice.key -out alice.crt
Signature ok
subject=C = ES, ST = Valladolid, L = Valladolid, O = 4ck.com, CN = www.4ck.com,
emailAddress = admin@4ck.com
Getting Private key
alejandro@alejandro-VirtualBox:/opt/rsa$

```

Si vemos los permisos de los 3 archivos creados (clave, solicitud y certificado) nos podemos fijar en que para el certificado solo tiene acceso el super-usuario. Sin embargo, para la solicitud y la clave privada tiene acceso el usuario sin permisos.

```

alejandro@alejandro-VirtualBox:/opt/rsa$ sudo ls -l /etc/ssl/
total 32
drwxr-xr-x 2 root root    16384 nov 10 13:22 certs
-rw-r--r-- 1 root root    10909 mar  6 2020 openssl.cnf
drwx--x--- 2 root ssl-cert 4096 nov 10 13:23 private
alejandro@alejandro-VirtualBox:/opt/rsa$

```