

Practica 1 Fundamentos de la Seguridad

Javier Díaz Domínguez

2019-2020

Cifrado simétrico de documentos.

En este apartado de la práctica se realizarán cifrados de documentos de texto empleando diferentes algoritmos de cifrado simétrico. Para ello se empleará la herramienta openssl, concretamente para este apartado el comando `enc` con sus diferentes variantes de modo de operación.

Comando `enc`

Este comando sirve para hacer cifrados simétricos. Para ello utiliza diferentes algoritmos de cifrado, tanto de bloque como de flujo, utilizando contraseñas proporcionadas por el usuario como claves.

Entre sus opciones más importantes podemos encontrar las opciones de gestión de contraseña (`-k`, `-kfile`, `-pass`), los de selección de modo de operación (`-e`, `-d`) para seleccionar entre cifrado y descifrado del archivo de entrada, y finalmente la opción de selección de modo de operación, con la que podremos seleccionar el modo de cifrado. De estos, los más comunes son:

- **ECB (Electronic CodeBook):** El más simple, simplemente se divide el mensaje en bloques y se cifran por separado.
- **CBC (Cipher Block Chaining):** También trocea el mensaje en bloques, pero además utiliza XOR para combinar el cifrado del bloque previo con el texto plano del bloque siguiente. Para cifrar el primer bloque, se emplea un vector de inicialización que cambia aleatoriamente cada vez que se inicia un cifrado.
- **CFB (CipherFeedBack):** Este modo de cifrado convierte un cifrado de bloque en un cifrado de flujo: para ello hace que el cifrado en bloque opere como una unidad de flujo de cifrado, generando bloques de flujo de claves que son operados con XOR y el texto plano para obtener el texto cifrado.
- **OFB (Output FeedBack):** Convierte un cifrado de bloque en **uncifrado de flujo síncrono**. Genera bloques pseudoaleatorios que son operados a través de XOR con el texto plano para generar el texto cifrado. Requiere vector de inicialización.
- **CTR (Counter):** Genera los bloques pseudoaleatorios utilizando un contador, que no es más que una función que produce una secuencia que se garantiza que no se repetirá en mucho tiempo.
- **DES:** Es el algoritmo prototipo del cifrado por bloques. Transforma un texto plano de una longitud determinada de bits en otro texto cifrado de la misma longitud. Utiliza cifrado en bloque, de 64 bits (56 efectivos, 8 últimos de paridad). La clave es de 64 bits. Este algoritmo está **roto**.
- **3DES (Triple DES):** utiliza una clave de 168 bits, aunque la eficiencia real es de 112 bits. Los bloques de cifrado siguen siendo de 64 bits. Este algoritmo no es tan vulnerable a los ataques por fuerza bruta como DES.
- **AES:** Está basado en una estructura de bloques, utilizando bloques de 128 bits y claves de 128, 192 y 256 bits. El cambio de un solo bit en la clave o en el bloque de texto a cifrar produce cifrados completamente distintos. Se caracteriza por su robustez y rapidez.

- **RC4**: Cifrado de flujo, hoy en día se considera débil. Utiliza dos algoritmos para el cifrado: KSA y PRGA.
- **Chacha20**: Sistema de cifrado de flujo que soporta claves de 128 y 256 bits. Está basado en una función pseudoaleatoria basada en operaciones add-rotate-xor.

Creación del fichero de texto.

Para la demostración práctica del funcionamiento de estos algoritmos, se crea un fichero de texto con el siguiente contenido

```
Hola, esto es un fichero de prueba para el cifrado mediante 5 algoritmos  
distintos. Utilizaremos los algoritmos AES y TDES obligatoriamente, y posteriormente un
```



Cifrado y descifrado

Se procede al cifrado utilizando los 5 algoritmos simétricos:

- AES 128

```
openssl enc -aes128 -e -in small-text.txt -out aes128-small-text.cif
```

- Triple DES

```
openssl enc -des3 -e -in small-text.txt -out des3-small-text.cif
```

- RC4

```
openssl enc -rc4 -e -in small-text.txt -out rc4-small-text.cif
```

- DES CBC

```
openssl enc -des-cbc -e -in small-text.txt -out des-cbc-small-text.cif
```

- Chacha20

```
openssl enc -chacha20 -e -in small-text.txt -out chacha20-small-text.cif
```

Y posteriormente se procede al descifrado de los archivos cifrados

```
openssl enc -aes128 -d -in aes128-small-text.cif -out aes128-small-text.txt  
openssl enc -des3 -d -in des3-small-text.cif -out des3-small-text.txt  
openssl enc -rc4 -d -in rc4-small-text.cif -out rc4-small-text.txt  
openssl enc -des-cbc -d -in des-cbc-small-text.cif -out des-cbc-small-text.txt  
openssl enc -chacha20 -d -in chacha20-small-text.cif -out chacha20-small-text.txt
```

Tamaño de los archivos cifrados

A continuación se vuelcan por pantalla los diferentes tamaños de los archivos cifrados con los distintos algoritmos.

```
$ stat * -c "%N %s"
'aes128-small-text.cif' 352
'chacha20-small-text.cif' 345
'des3-small-text.cif' 352
'des-cbc-small-text.cif' 352
'rc4-small-text.cif' 345
'small-text.txt' 329
```

Como podemos observar, el tamaño original del archivo es de 329 bytes. Al utilizar los algoritmos de flujo `chacha20` y `rc4`, obtenemos un tamaño de fichero de 345 bytes. Esto se debe a la aparición de la sal al inicio del fichero, que es de 16 bytes. Los tamaños de los ficheros comprimidos con los cifradores de bloque también tienen su explicación: * Para `aes-128`, al fichero se le añaden 16 bytes de sal, $16 + 329 = 345$, y posteriormente se le añade el denominado *padding* para completar el tamaño del último bloque del cifrado hasta los 128 bits (16 bytes). Gracias al padding, el tamaño total del fichero es múltiplo de 16 bytes (128 bits). * Para `des3` ocurre lo mismo que con `aes-128`, y el padding coincide ya que el número más cercano de 345 múltiplo de 8 (que es el tamaño de bloque de `des3` en bytes) coincide con el de 16. * `des-cbc` comparte tamaño de bloque con `des3` por lo que la situación es la misma.

Contenido de los ficheros encriptados

- Fichero original

```
Hola, esto es un fichero de prueba para el cifrado mediante 5 algoritmos
distintos. Utilizaremos los algoritmos AES y TDES obligatoriamente, y posteriormente un
algoritmo de flujo. Los demás serán de libre elección. Para esta práctica concretamente
```

- `aes-128`

```
Salted__00000c0S00R00000I0
000Û/0'0-$ 0$00Y00100f0'0D00(0)00Z00 0Io0000gP00¿Q000#0j00000f050204x.00 c000
M00N00X 00A0[200,(0300000}0p0<0c:070%
```

- `des3`

```
600U0FD005-8>0500)wa00XZ0Ja00)0000oy000'u005f9i0000'00Z020gj0W0^K0000=° 0Avp00000 00):
\7i0%
```

- `des-cbc`

```
Salted__00f00?{[000oZ0]01odp00H<0-B007j00c*/0E000000Ye08V00]00k00000A}000Qk00_0j00&
900:90000n60*0000+0000JK00)00070000z/wF~q0zE&0
0z0 0I=K0hn00r0B00%0[U0` ]QX000p0 0c0 00c
00<0g000P0E0%V~g0j150000j[0V0000K05K 0`0q0{0$00Ix050v%
```

- chacha20

```
Salted__N0000G0F00d'000000n0)00]U00WP0]0500
JEx0=400z0zNAh0Qb00>z?K&0!}z10~8000B00kF@00
0]x'00XeUZ000
```

- rc4

```
Salted__qh00
ru00W?80404
9%0S0000000305:G00X0
0f00000q0>0j00W0s0@0#0000s000P!0
6:0U/p0Jt000q^00H0X;006007]0ÿy0i\e00{00.~000&0Z000W0t!00}0)00w}n000 '00o~10>0000ko@
60.f%&0{000000@k0000M$
```

Gestión de contraseñas

PKCS (Public Key Cryptography Standard) es un grupo de estándares de utilización de técnicas criptográficas en la gestión de claves públicas en algoritmos criptográficos. Concretamente openssl nos permite utilizar `pbkdf1` y `pbkdf2`, que son implementaciones de PKCS. La principal diferencia entre ambos es que `pbkdf1` no es capaz de generar claves de más de 160 bits, mientras que `pbkdf2` lo hace de 128, 256 y 512 bits.

El algoritmo funciona aplicando una función pseudoaleatoria (HMAC con una función hash aprobada) a la contraseña de entrada junto con un valor de sal, repitiendo este proceso muchas veces (mínimo 1000 iteraciones) produciendo una clave derivada. Ésta se puede utilizar como una clave criptográfica en operaciones posteriores. También se puede utilizar la salida de este algoritmo para rellenar el vector de inicialización. El uso de la sal en la gestión de la contraseña permite que para una misma contraseña se generen diferentes claves criptográficas.

Descifrado de un fichero con clave, vector y sal.

Se propone la demostración de que un fichero puede ser descifrado con la clave criptográfica, el vector de inicialización y la sal, sin el conocimiento de la contraseña.

Para proceder con la demostración en primer lugar se cifra un fichero. Se utiliza como contraseña prueba, y la opción `-p` para mostrar el vector de inicialización, la sal y la clave.

```
$ openssl enc -des-cfb -in small-text.txt -out cfb-small-text.cif -k prueba -p > cvs.txt
```

comprobamos el contenido del fichero generado

```
$ cat cvs.txt
salt=99C6D2581D8BF4E0
key=952CEA08115F4846
iv =FA67BC49486D61CB
```

preparamos el fichero eliminando los primeros 16 bytes de la sal del inicio del fichero

```
$ cat cfb-small-text-1.cif | dd ibs=16 obs=16 skip=1 > cfb-small-text1.cif
20+1 registros leídos
20+1 registros escritos
329 bytes copied, 0,000149952 s, 2,2 MB/s
```

y procedemos al descifrado utilizando estos datos

```
$ openssl enc -des-cfb -d -in cfb-small-text1.cif -K 952cea08115f4846 -iv fa67bc49486d61cb
Hola, esto es un fichero de prueba para el cifrado mediante 5 algoritmos
distintos. Utilizaremos los algoritmos AES y TDES obligatoriamente, y posteriormente un
algoritmo de flujo. Los demás serán de libre elección. Para esta práctica concretamente
```

Demostración de la peligrosidad del modo de operación ecb

Para ello utilizaremos la siguiente imagen

ulpgc.png

Se procede eliminando las cabeceras para la encriptación

```
$ head -n 3 ulpgc.pgm > header.txt
```

se extrae el cuerpo de la imagen

```
$ tail -n +4 ulpgc.pgm > body.bin
```

se cifra el cuerpo de la imagen

```
$ openssl enc -aes-128-ecb -nosalt -k prueba -in body.bin -out body.cif
```

por último, se juntan las cabeceras y el cuerpo cifrado

```
$ cat header.txt body.cif > ulpgc-ecb.pgm
```

y se obtiene la siguiente imagen

ulpgc-ecb.png

que efectivamente prueba la peligrosidad del modo de operación ecb, ya que deja ver patrones similares a los de la foto original. En contrapunto, si utilizáramos otro modo de encriptación como cbc, el resultado sería el siguiente

ulpgc-cbc.png

Cifrado y comprobación de resúmenes: Generación de claves asimétricas (pública-privada) y firmado de resúmenes

Openssl ofrece la posibilidad de generar resúmenes de archivos con diferentes algoritmos de resumen mediante la opción `dgst`. Haciendo uso de esta herramienta, se generan varios resúmenes utilizando los siguientes algoritmos

1. SHA-1

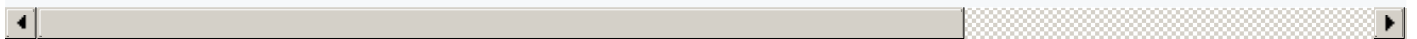
```
$ openssl dgst -sha1 small-text.txt
SHA1(small-text.txt)= 0df857cb098430eeb259bf9e0c830889d3174f47
```

3. SHA224

```
$ openssl dgst -sha224 small-text.txt
SHA224(small-text.txt)= 64010395cc20c147c7a7566a54a6ac5be73cb04a30e1389548c2183a
```

3. SHA384

```
$ openssl dgst -sha384 small-text.txt
SHA384(small-text.txt)= 54ffce1aa48d5e4ac279b5841759c5580d6e70f09f558cc3a3f6cfbde21760e4
```



Se realiza un pequeño cambio en el contenido del fichero: se sustituyen dos caracteres, y se comprueba que, en efecto, los resúmenes del archivo cambian considerablemente

1. SHA-1

```
$ openssl dgst -sha1 small-text.txt
SHA1(small-text.txt)= 1cae6758d51562e16a6c99f9fa55fb6e852527c8
```

3. SHA224

```
$ openssl dgst -sha224 small-text.txt
SHA224(small-text.txt)= d6da3cb542cee30107019fb4b3a616c4dbcebdb92d308b02b8acc912
```

3. SHA384

```
$ openssl dgst -sha384 small-text.txt
SHA384(small-text.txt)= a7e82cb8640ccd70e455d5285fa89ea67be185f5a7f1111f59d1be99d8179275
```



Generación de un par de claves asimétricas RSA de 2048 bits.

Se genera la clave privada

```
$ openssl genpkey -algorithm RSA -aes256 -out privateRSA.pem -pkeyopt rsa_keygen_bits:2048
```



y se introduce como contraseña `prueba` . A continuación se muestra el contenido del fichero que contiene la clave privada

```
$ cat privateRSA.pem
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFLTBxBgkqhkiG9w0BBQ0wSjApBgkqhkiG9w0BBQwwHAQI4V+LRBBOFzoCaggA
MAwGCCqGSIb3DQIJBQAwHQYJYIZIAWUDBAEqBBCtVxlBvJVXvGRlpEilcQQUBIIE
0JOTlbyl+NBrBwu01bbZvIOiDesAiXGKBnn36plukfFkEkf/cMNR7vyDqtqe0/Sz
H5gDLWKnS8BM8gnnywMaeG+jJ9ILvKIEOw5zzHT2Ji3hHze4CeEHBwRSMQR3B01Y
Kdy62I61HoBZomkR+BLN6uWVakL6kom6jVIyUIMZTm00sCSf+5gTYdTMxWNYV/s0
RkUihENPpI/dvwlTaWjjeD25oEtXPLhat6B0+9RpywcXn/PDAXdqpqYdXZ/sBdMa
SaMfb5FbhNcGtV96iw3lzs7Tzn17AFv1nKHmlzl+cbNBcUSxsMFNBoQ8yrDq3ze
X2BYJcEk7mzt/jB+SaT9GkDTR9QHy0tv1qv/Pv/vutF+B+IwkDymC0vBiCpjrnwx
H1CizDr4nVMjnye+UISLuJqULazblNuykgIezm28VY23SwDpmglynjIvCO9ueaLF
uB4kaVC7FGBK2XVSLMtX1JMsVsMbdL+nNNvdnhVwkB0sBwplXicmlwwkfRAwgCmT
f6AKUou+/1IMogXgo14C3HpL3u0bKM7dPrDxKnpSMGaMGLi/tp4SL8e3TGL/2Gs8
e0Unxo8CJTBAxhL4NMdUvw0CCqXA7i97WwxvX9COQaDAaVTSxQsxHvFBnNJMGGS
2xvX05nR6iWeofVpme5xdHQPNIe/DnJS3E8MdPwGp3lX+7ADCv2YAnYFEw5SFXSE
LktoyAfg70JPBfx0vee7AlMzHKMkokjNQ2E8QftlOuwg+xGex4BwDrTuM+wpjpZe
e3Uj0r/Z/6zhKUBnHln9ixb61h0Gt3e1RMUyVhGpicWgAOeq6PXv4qgn0S4SzaGj
jPtZokGG5WDLw4TiZBQbuXjMGjP0gd73TYiEzspNUWea3TwDUDDcb4wMdB0Jcy0Z
w6NxX56j7HjnEfmuLjAXdiasubQ4f3wL3T98+hu2EGoiKCOMx1Sa91JQ4JIg5UOk
H3kuoLrunvaSvBQGDHiq3vmKWxdvBVrONcm2gevFrAcPqukvCRF6q1nLTqjVAhxJ
vYgXw+KsHAY/fD50m2r501ToLG00RSRs7DPYCXLH9jbaLDnhY2JXn9C3LgRwyHf
i+B0pNexmXPLG40uNESVb5VWUQrhTPG1w12E3sCe/1WlVPGa+ujErQtReJTrZrgi
vW6lvxHr0iUVnKK2VAZ63K01Bbu40BfGNEDDAPKhU2phM4/souKxs/jRK1QXPck9
msqWWLC6ZTaog2z605e81Lhpngwh/wNkhHsNBRedP3RRhpEdbWa0rfNE0vbL2skW
6NeLdqIiVMiJf3DlD9hmU3bPnnLcDgdip7fesVt70WUoKCuPNIJ6ea9e3UjRiWTl
fWfULmWlj3uAM61FJEO9fNb8I55y3qmcZCnxTobQaPQa4fKqvVkspcilvWUXL6QFr
o6acJ9YUcHWFT9+ODjYd8JdYQ0XPMGaUzLtYl20DBf0mPzBFZ0A1853QabFVier
l1o10YJZSiJjo2GnIgl9ot+0ZmOkAMTn2eXZmL7xu2/ZVZeh9lw4tWo2xDduImlj
/9rNHEAND/nZFhjTAonURE1VsB3WqdAufsUbMadvrgIfaMhYLnHeIFN0BaPPb1Do
kh7IfaETgDge1SLSZy5k1VW/K1qVfxwRV+nYxGekFZCz
-----END ENCRYPTED PRIVATE KEY-----
```

Para extraer la clave pública del fichero utilizamos la siguiente orden

```
openssl pkey -in privateRSA.pem -pubout -out publicRSA.pem
```

y obtenemos la siguiente clave pública


```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEARw4swEraW1U47cSipXfG
0F906mGj/Q8wVdn4fW1cTJR5DZJdygh3IrGRlQETTRbzoqN8UiuOQTEcgLTef+vY
x7Meiuui2oSwhLlPMFEaZ5RAYXSB4V8zVtTY4QHNmEUMd5hY2PLtmDmzh/NAbGgL
6pZJRcRiaSuuK8vQCL/xHf04fWPHs4dtlvbUpllcuFV+N0pU5FrDzPKo8DEx2Mq
xigg+3copXL1tp6FsX4vazFbGmiy9wIDoWXXUSbEuPuw50UbyqOC3FU/V6fh7BiW
F1ucrIKs8czJ8jdmFEdyf36AVwp6PgQ+ZvYS202w1T5T2Rd1XbhGDphE1YJm9G1a
9wIDAQAB
-----END PUBLIC KEY-----
```

Exportación del par de claves en formatos DER Y PEM

Para seleccionar el formato de exportación de las claves se utiliza la opción `-outform DER | PEM` del comando `openssl genpkey`. Los comandos de generación de claves quedarían así para el formato DER

```
$ openssl genpkey -algorithm RSA -aes256 -out privateRSA.der -outform DER -pkeyopt rsa_l
```

y para el formato PEM

```
$ openssl genpkey -algorithm RSA -aes256 -out privateRSA.pem -outform PEM -pkeyopt rsa_l
```

Para la conversión de las claves entre ambos formatos podemos utilizar el comando `pkey`, de la siguiente manera

```
$ openssl pkey -in privateRSA.pem -inform PEM -outform DER -out privateRSA.der
```

y viceversa

```
$ openssl pkey -in privateRSA.der -inform DER -outform PEM -out privateRSA.pem
```

Lo mismo se aplica para la conversión de la clave pública.

Firma y comprobación de la firma

En primer lugar se crea el fichero `mensaje.txt` que contiene `Esto es un mensaje`, y se crea el resumen firmado con la clave privada que contiene el fichero `privateRSA_1.pem` (que es el fichero creado anteriormente).

```
$ openssl dgst -sha256 -sign privateRSA_1.pem -out resumen_firmado.sha mensaje.txt
```

y se verifica que el resumen es correcto con la siguiente instrucción

```
$ openssl dgst -sha256 -verify publicRSA_1.pem -signature resumen_firmado.sha mensaje.t
```

Creación dos claves DH y demostración de la equivalencia entre combinaciones.

Se generan los pares de claves

```
$ openssl genpkey -algorithm X25519 -out privateDH_1.pem
$ openssl pkey -in privateDH_1.pem -pubout -out publicDH_1.pem
$ openssl genpkey -algorithm X25519 -out privateDH_2.pem
$ openssl pkey -in privateDH_2.pem -pubout -out publicDH_2.pem
```

y se deriva un valor utilizando la combinación `privateDH_1.pem`, `publicDH_2.pem`.

```
$ openssl pkeyutl -derive -inkey privateDH_1.pem -peerkey publicDH_2.pem -out secret1
```

y la combinación `privateDH_2.pem`, `publicDH_1.pem`.

```
$ openssl pkeyutl -derive -inkey privateDH_2.pem -peerkey publicDH_1.pem -out secret2
```

Se comprueba que, efectivamente, los valores generados son equivalentes

```
$ cmp secret1 secret2
```

Cifrado asimétrico de documentos

En esta parte de la práctica se pide cifrar un documento de texto y enviarlo a un compañero junto con la clave cifrada con su clave pública RSA. Para cumplir la especificación, se crea el fichero de texto `mensaje.txt` y se cifra utilizando `aes-256`.

```
$ openssl enc -aes-256-cfb -in mensaje.txt -out mensaje.aes-256-cfb
```

El siguiente paso es cifrar la contraseña con la clave pública de mi compañero (Néstor Pérez Haro)

```
$ openssl pkeyutl -pubin -encrypt -in contraseña -out contraseña.cif -inkey clavepublica
```

y por último firmar el resumen del archivo de texto con mi clave privada

```
$ openssl dgst -sha256 -sign myprivatekey.pem -out resumen-mensaje-firmado.sha256 mensa
```

Con estos tres archivos se puede enviar un mensaje cifrado que cumple con los principios de la seguridad en la comunicación: confidencialidad, integridad, autenticación y no repudio.

Así mismo, los archivos que he obtenido del intercambio de mensajes son los siguientes:

- `archivoaes-nes`, cifrado mediante `aes-256-cbc`
- `contra-nes.cif`, que es la contraseña del cifrado simétrico del archivo anterior
- `firmanes.rsa`, que es el resumen firmado del documento cifrado.

Con estos datos es posible lo siguiente

- Descifrar la contraseña simétrica contenida en `contra-nes.cif` utilizando mi clave privada

```
$ openssl pkeyutl -decrypt -in contra-nes.cif -out contra-nes.txt -inkey myprivate
```

- Luego de esto, se utiliza la contraseña obtenida para descifrar el archivo `archivoaes`
`bash $ openssl enc -aes-256-cbc -d -in archivoaes-nes -out mensaje-nestor.txt -kfile contra-nes.txt`
- Por último debemos verificar la autenticidad del emisor con su firma
`$ openssl dgst -md5 -verify clavepublicanes.pem -signature firmanes.rsa mensaje.txt`

Envío, recepción y decodificación manual de mensajes S/MIME firmados y cifrados, empleando certificados creados por el estudiante y firmados con el certificado raíz de prácticas de la asignatura.

Para empezar se crea la clave privada personal. Esta clave se utilizará para crear el certificado personal posteriormente. Como clave utilizamos `unacontraseña`.

```
$ openssl genpkey -algorithm RSA -aes256 -out certificadoPersonal.key -pkeyopt rsa_keyg
```

A continuación se genera el susodicho certificado introduciendo la información que se solicita

```
$ openssl req -new -key certificadoPersonal.key -out certificadoPersonal.csr
Enter pass phrase for certificadoPersonal.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]:ES
State or Province Name (full name) [Some-State]:Las-Palmas
Locality Name (eg, city) []:Arucas
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Ulpgc
Organizational Unit Name (eg, section) []:Estudiante
Common Name (e.g. server FQDN or YOUR name) []:www.javidiazdom.dev
Email Address []:javier.diaz119@alu.ulpgc.es
```

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

La firma del certificado generado se completa ejecutando el siguiente comando

```
$ openssl x509 -req -days 365 -in certificadoPersonal.csr -CA certificadoRaiz.crt -CAkey
```



Para visualizar los detalles del certificado se utiliza

```
$ openssl x509 -text -noout -in certificadoPersonal.crt
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            73:b0:89:63:0d:56:25:20:24:14:2c:85:09:78:15:67:b0:a3:12:f7
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = ES, ST = Las Palmas, L = Las Palmas de G.C., O = AA - Autoridad de p
        Validity
            Not Before: Mar 31 15:50:15 2020 GMT
            Not After : Mar 31 15:50:15 2021 GMT
        Subject: C = ES, ST = Las-Palmas, L = Arucas, O = Ulpgc, OU = Estudiante, CN = w
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:a6:70:07:c8:1a:b2:e6:bb:fc:20:13:a7:9a:bc:
                    5d:9c:84:16:7e:98:fc:50:7e:59:4f:1f:9c:4a:df:
                    11:66:eb:01:ea:b5:7a:37:63:ac:c7:89:90:7d:b5:
```

```
47:ec:9e:c9:80:84:69:8c:db:7b:60:cc:b0:77:ac:
c2:99:65:b4:0b:a5:76:b5:3f:2b:9d:4a:60:77:79:
9f:6c:f6:b5:41:75:44:11:b1:5a:a0:3e:3a:aa:8b:
b2:dc:12:6e:a1:84:94:27:27:b3:14:ef:7a:f9:b2:
d0:16:bb:6c:c1:64:bb:36:aa:4c:c0:a1:42:ba:2b:
ac:8b:f9:24:27:4d:45:36:11:9b:e8:67:35:dd:8b:
61:30:c9:9c:32:9f:8d:67:18:f2:40:f9:d0:48:82:
ed:4c:f4:c6:c6:76:03:3d:2c:dd:64:7b:4d:20:bc:
f7:57:03:7a:62:f7:08:6c:0c:7e:ce:fd:52:57:60:
9f:8f:c5:8b:cb:92:eb:e6:51:4b:a9:06:d7:fb:96:
fd:15:e2:28:7a:15:0e:02:0d:79:3a:59:56:44:ac:
6e:72:1b:69:f8:70:d5:a2:37:7f:b9:9c:64:c9:c0:
41:dd:3c:30:f5:bc:a8:f5:c6:8a:b0:8b:9a:68:da:
8d:7e:b9:4b:c5:89:3a:ec:14:49:c4:b8:64:09:d5:
e3:63
```

Exponent: 65537 (0x10001)

Signature Algorithm: sha256WithRSAEncryption

```
be:39:22:cc:0b:38:e4:34:2b:de:20:09:65:61:0e:bc:6a:6e:
42:a7:83:e4:93:36:74:d1:4c:f7:51:0b:c7:a6:59:e6:b6:e4:
93:8b:d4:72:f6:19:c5:15:5b:c1:f5:0a:22:9b:27:d6:89:2a:
43:5a:08:bf:da:cf:16:15:27:4e:b0:64:4f:2b:a2:11:a4:6b:
cf:93:e2:e0:00:d8:26:d5:1c:64:f1:82:1b:a0:6a:ad:91:f8:
78:e6:08:f1:a7:dd:c5:14:3f:1f:32:ea:89:df:dd:e2:96:87:
4a:35:de:d2:eb:fb:23:c4:91:ce:81:96:27:0c:39:11:ca:89:
12:48:8c:2d:4b:cf:ec:bf:b1:49:0f:ea:b9:e0:eb:78:ad:56:
32:be:d0:00:3e:b8:1c:a2:03:98:28:4b:2a:a7:2b:b5:23:c3:
23:e0:57:af:f6:fc:d1:72:76:7d:fa:9a:54:ac:ae:af:11:83:
ad:57:15:6a:f3:14:18:0b:0a:76:66:21:ec:1c:fc:88:fa:1e:
fd:e3:8d:4a:8f:c8:06:98:18:a9:61:19:51:dc:cf:a6:a1:fb:
be:16:e9:42:62:bb:7c:07:b9:0f:30:ca:33:36:10:13:75:0f:
b1:68:0e:21:8b:72:23:17:f2:c1:bb:0f:5f:47:bb:43:59:fb:
a6:ce:5b:b8
```

Lo siguiente es obtener el fichero PKCS#12 con la parte pública y privada del certificado personal generado anteriormente.

```
$ openssl pkcs12 -export -in certificadoPersonal.crt -inkey certificadoPersonal.key -ou
```

Instalación de los certificados en el cliente de correo e intercambio de certificados y correo firmado y cifrado con otro compañero

Con el certificado generado en el apartado anterior, concretamente con su exportación en formato PKCS12, se podrá configurar el cliente de correo electrónico *Mozilla thunderbird*. Después de configurar el cliente con el correo personal (javidiazdom@gmail.com) e importar tanto el certificado raíz como el certificado personal, se configura la cuenta de correo para que utilice el certificado personal.

Configuración de un servidor web seguro utilizando un certificado autofirmado, con privilegios de asministrador en un servidor web Apache.

Configurar apache para mostrar una página web a través del protocolo https con un certificado autofirmado es bastante simple. Partiendo de una máquina ubuntu 19.10 con apache previamente instalado, los pasos a seguir son los siguientes:

1. Crear tanto el certificado autofirmado como la clave privada. Para generar la clave

```
$ openssl genpkey -algorithm RSA -aes256 -out certificadoServidor.key -pkeyopt rsa_keyg
```

y para el certificado

```
$ openssl req -new -key certificadoServidor.key -out certificadoServidor.csr  
$ openssl x509 -req -days 365 -in certificadoServidor.csr -signkey certificadoServidor.k
```

2. Definir el host `www.ejemplo.com` en el archivo `/etc/hosts`.

```
127.0.0.1    localhost  
::1         localhost  
127.0.1.1    pop-os.localdomain  pop-os  
127.0.1.1    www.ejemplo.com
```

3. Configuración de apache:

Cargar el módulo ssl

```
$ a2enmod ssl
```

Crear el archivo que define el host virtual que escucha en el puerto https (443)

```
$ touch /etc/apache2/sites-available/fs-ssl.conf
```

En este fichero se debe introducir lo siguiente

```
<VirtualHost 127.0.1.1:443>  
    DocumentRoot /var/www/fs  
    ServerName www.ejemplo.com  
    SSLEngine on  
    SSLCertificateFile "/home/javidiazdom/Documentos/Practica FS/apache/certificadoServi  
    SSLCertificateKeyFile "/home/javidiazdom/Documentos/Practica FS/apache/certificadoSe  
</VirtualHost>
```

Crear el contenido de la página y el directorio donde se aloja

```
$ mkdir /var/www/fs
$ touch /var/www/fs/index.html
```

En `index.html` se podrá introducir cualquier estructura html. En mi caso he introducido

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <h1> EJEMPLO </h1>
  </body>
</html>
```

Habilitar la página y recargar apache para aplicar los cambios

```
$ a2ensite fs-ssl
$ systemctl reload apache2
```

Con esto ya está completada la configuración, apache mostrará el contenido de `index.html` cuando se acceda a `www.ejemplo.com` a través del protocolo https.

Esto se comprueba accediendo a `https://www.ejemplo.com` a través del navegador. □

En efecto, muestra el contenido de `/var/www/fs/index.html`. Además, si se accede a la información de seguridad, se aprecia el protocolo utilizado, TLS 1.3, además del cifrado obtenido AES_128_GCM. □ También se detalla información acerca del certificado utilizado, y se puede ampliar esa información para ver todos los detalles del certificado. □

Bibliografía

Gran parte de la información para el desarrollo de la práctica ha sido obtenida de la página proporcionada en el campus de la asignatura [How to openssl](#). Además de esta web, se han utilizado otras webs para puntualizaciones y consultas específicas, así como para la última parte de la práctica.

- [Openssl manual oficial](#)
- [Wikipedia](#)
- [Stack overflow](#)
- [Digicert](#)
- [Liquid Web](#)