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`pr

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 un **cifrado 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 utilzando 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 eficiciencia 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 algoritmo de flujo. Los demás serán de libre elección.
Para esta práctica concretamente se seleccionan los siguientes algoritmos:
aes128, des3, rc4, cbc, chacha20.
```

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 se seleccionan los siguientes algoritmos: aes128, des3, rc4,
cbc, chacha20.
```

- aes-128

```
Salted__������c�S��R��O��I�
���Ū/�'�-$ ⍰$���Ɏ��l��f‘�D��(⍰)��Z��
⍰I⍰o����gP��ϡQ���#⍰j⍰⍰0⍰⍰f⍰5⍰2⍰4x.⍰⍰
C⍰O⍰mz`⍰⍰⍰(⍰⍰u⍰⍰ɬ⍰~⍰⍰⍰⍰Kɸ⍰x⍰⍰⍰R⍰4"⍰⍰⍰⍰⍰⍰$xS⍰⍰R⍰⍰x⍰v⍰3\
⍰H⍰⍰⍰L⍰a⍰l⍰bHl⍰Z#⍰t⍰ o⍰⍰⍰U⍰j1{⍰⍰
⍰:]⍰⍰⍰⍰⍰⍰⍰z3⍰w⍰J*3⍰⍰⍰f⍰V⍰⍰⍰˜⍰9⍰l
M⍰⍰N⍰⍰X⍰⍰A⍰[2⍰⍰,(⍰3⍰⍰⍰O⍰}⍰p⍰<⍰c:⍰7⍰%
```

- des3

```
6O⍰U⍰FD⍰⍰5-
8>⍰5⍰⍰)wa⍰⍰XZ⍰Ja⍰⍰)⍰⍰⍰⍰ϑ⍰⍰⍰'u⍰⍰5f9i⍰⍰⍰⍰'⍰⍰Z⍰2⍰gj⍰W⍰^K⍰⍰
⍰O=˚⍰ÂVp⍰⍰⍰⍰⍰⍰⍰)1ȳ7X⍰⍰⍰q|
⍰⍰⍰%m⍰(⍰⍰⍰f9V⍰w*3⍰⍰Od@~Ġ)5⍰⍰_⍰@⍰⍰⍰0⍰⍰;⍰q⍰⍰⍰8⍰⍰⍰⍰⍰5⍰⍰⍰⍰.⍰
⍰r⍰Cl⍰⍰⍰⍰(M⍰4⍰RA⍰cB#R⍰⍰*⍰4ف(⍰⍰⍰⍰⍰⍰⍰⍰⍰
\7i⍰%
```

- des-cbc

```
Salted__⍰⍰f⍰⍰?{[⍰⍰⍰oZ⍰ǰ⍰1odp⍰⍰H<⍰-B⍰⍰7j⍰⍰c*/
⍰E⍰⍰⍰⍰⍰⍰Ye⍰8V⍰⍰]⍰⍰ƙ⍰⍰⍰⍰⍰A}⍰⍰⍰Qk⍰⍰_⍰j⍰⍰&;N⍰)⍰⍰⍰⍰;
9⍰⍰:9⍰⍰⍰⍰n6⍰*⍰⍰⍰⍰+⍰⍰⍰⍰JK⍰⍰)⍰0⍰7⍰⍰⍰⍰z/wF~q⍰zE&⍰
                                        ⍰z⍰
⍰I=K⍰hn⍰⍰r⍰B⍰⍰%⍰[U⍰`]QX⍰⍰⍰p⍰⍰c⍰⍰⍰6⍰⍰

⍰⍰<⍰g⍰⍰⍰P⍰E⍰%V~ǵ⍰j15⍰0⍰⍰j[⍰V⍰⍰⍰⍰K⍰5Ƙ ⍰`⍰q⍰{⍰$⍰⍰Ix⍰5⍰v%
```

- chacha20

```
Salted__N⍰⍰⍰⍰G⍰F⍰⍰d'⍰⍰⍰⍰⍰⍰n⍰)⍰⍰]U⍰⍰WP⍰]⍰5⍰⍰
                                        JEx⍰=4⍰⍰z⍰zNAh⍰Qb⍰⍰>z?
K&⍰!}zl⍰~8⍰⍰⍰B⍰⍰kF@⍰⍰⍰⍰a⍰⍰⍰\⍰⍰~⍰Æb⍰⍰⍰⍰⍰/~ţ⍰⍰ĭ⍰⍰j⍰⍰~mH⍰@
⍰⍰⍰⍰za:J<⍰0J⍰e⍰V⍰⍰⍰⍰K⍰A⍰⍰$kp|O<H⍰⍰ɱ⍰⍰⍰V⍰⍰⍰⍰⍰;⍰;⍰

⍰]x'⍰⍰XeUZ⍰⍰⍰I/⍰u⍰⍰⍰1RX⍰k⍰nQ}OTȘw⍰E?⍰WsR⍰⍰L:⍰⍰e⍰;⍰⍰vF!
⍰cĎ@3⍰⍰⍰⍰%⍰v%
```

- rc4

```
Salted__qh⍰⍰ʿ
           ru⍰⍰W?8⍰4⍰4
9%⍰S⍰0⍰⍰⍰⍰⍰3⍰5:G⍰⍰XO
```

```
f[illegible]q>j[illegible]W s@@#[illegible]s[illegible]P!
6:U/p Jt[illegible]q^[illegible]H X;[illegible]7]Эy i\e[illegible]{[illegible].~0&Z O W t!
[illegible])[illegible]w}n[illegible] '[illegible]o~l>[illegible]ko@@[illegible]
[illegible]n pr AT[illegible]c[illegible]d[illegible]& ķ[illegible]Ч}
[illegible]*FC[illegible]e[illegible]#[illegible]#r)[illegible]&K.,[illegible]~Sq
6.f%@&{[illegible]@k[illegible]MŞ
```

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 se seleccionan los siguientes algoritmos: aes128, des3, rc4,
cbc, chacha20.
```

Demostración de la peligrosidad del modo de operación ecb

Para ello utilizaremos la siguiente imagen

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
```

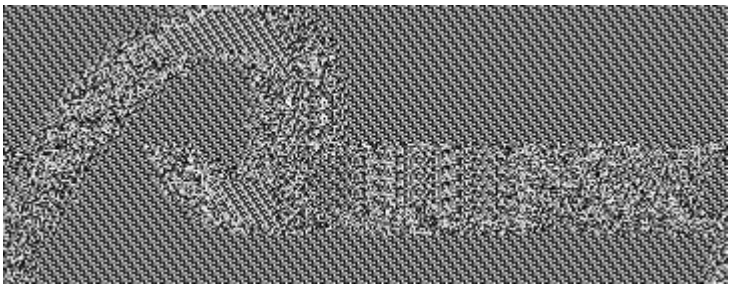
se cifra el cuerpo de la imagen

```
$ 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



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 utilizaramos otro modo de encriptación como cbc, el resultado sería el siguiente