Seguridad en Redes

Práctica 2.1. Criptografía de clave secreta y funciones resumen

A. OpenSSL

A.1. Introducción a OpenSSL

Para ver las principales opciones de OpenSSL usar:

```
$ openssl -help
```

Para ver las principales opciones de cifrado/descrifrado de OpenSSL y el listado de algoritmos de cifrado/descifrado disponibles usar:

```
$ openssl enc -help
```

Para ver las principales opciones de "messasge digest" de OpenSSL y el listado de funciones "hash" disponibles usar:

```
$ openssl dgst -help
```

También se pueden ver los comandos básicos de openssl, los principales comandos de cifrado y los principales comandos de "message digest" usando las siguientes órdenes:

```
$ openssl list-standard-commands
$ openssl list-cipher-commands
$ openssl list-message-digest-commands
```

A.2. Cifrado de bloque usando clave y vector de inicialización

Revisa la página de manual de `openssl` y la de los comandos que se irán presentando a continuación.

Para cifrar un archivo usando una clave y un vector de inicialización (IV), se usa el comando `enc` con la opción `-e` (usada por defecto), indicando un algoritmo de bloque (`aes-128-ecb`, `aes-128-cbc`...), la clave de cifrado y el IV:

```
$ openssl enc -e [-algoritmo] -iv <iv> -K <key> \
  -in <plaintextfile> -out <ciphertextfile>
```

Para descifrar, se usa el comando `enc` con la opción `-d`:

```
$ openssl enc -d [-algoritmo] -iv <iv> -K <key> \
  -in <ciphertextfile> -out <plaintextfile>
```

Por defecto, el archivo cifrado se genera en formato binario. En algunas ocasiones es conveniente generar un archivo cifrado en formato ASCII (por ejemplo, para enviarlo como

archivo adjunto a un e-mail). Para ello se debe añadir la opción `-a` o `-base64` (ambas opciones son equivalentes).

Para generar una clave aleatoria de 128 bits (16 bytes) en formato hexadecimal, se puede usar la siguiente orden (lo mismo para el vector de inicialización):

```
$ openssl rand 16 -hex
```

Ejercicio 1

- Crea un archivo de texto llamado `textoplano1.txt` (para ello se pueden usar varias ordenes: `cat`, `tee`, etc. o algún editor disponible: `vi`, `nano`, `gedit`, etc.)
- Genera una clave y un IV aleatorios, ambos de 128 bits y en formato hexadecimal
- Usa el algoritmo de cifrado `aes-128-cbc` para cifrar el archivo anterior (el archivo cifrado puede llamarse `texto-aes-128-cbc.bin`). Para ello, copia y pega la clave y el IV generado anteriormente
- Descifra el archivo cifrado usando el mismo algoritmo, clave e IV. Guarda el archivo descifrado con el nombre `textoplano2.txt`
- Comprueba que el `textoplano1.txt` y el `textplano2.txt` son iguales (puedes usar la orden `diff`)
- Intenta visualizar con el comando `cat` el archivo cifrado. Observa que está en formato binario

Ejercicio 2

- Repite el ejercicio anterior, pero en este caso generando un archivo cifrado en formato ASCII (opciones `-a` o `-base64`), que llamaremos `texto-aes-128-cbc.txt`
- Visualiza con el comando `cat` el archivo cifrado. Observa que está en formato ASCII.

Entrega #1: Copia y entrega la clave de cifrado y el IV generados, así como el archivo `texto-aes-128-cbc.txt`.

Ejercicio 3

- Repite los ejercicios anteriores usando otros algoritmos de cifrado distintos.

A.3. Cifrado de bloque usando contraseña (*password*)

El uso de una clave hexadecimal puede resultar poco práctico, ya que es difícil de recordar y, para evitar repetir la clave, debemos generar una nueva cada vez. Una opción alternativa es utilizar una contraseña (*password*) de cifrado en formato ASCII. Para ello, se usa la siguiente orden:

```
$ openssl enc -e [-algoritmo] -in <plaintextfile> \
  -out <ciphertextfile>
```

A continuación se solicitará al usuario que introduzca una contraseña por teclado (ésta se pedirá por duplicado, para confirmar la contraseña)

Para descifrar se usa la orden:

```
$ openssl enc –d [-algoritmo] -in <ciphertextfile> \
  -out <plaintextfile>
```

Alternativamente se puede introducir la contraseña como un argumento al ejecutar la orden de cifrado/descifrado:

```
$ openssl enc –e [-algoritmo] –pass pass:<password> \
  -in <plaintextfile> -out <ciphertextfile>

$ openssl enc –d [-algoritmo] –pass pass:<password> \
  -in <ciphertextfile>  -out <plaintextfile>
```

O también podemos tener la contraseña de cifrado/descifrado almacenada en un archivo:

```
$ openssl enc –e [-algoritmo] –pass file:<password_file> \
  -in <plaintextfile> -out <ciphertextfile>

$ openssl enc –d [-algoritmo] –pass file:<password_file> \
  -in <ciphertextfile>  -out <plaintextfile>
```

Ejercicio 4

- Crea un archivo de texto llamado `textoplano1.txt`
- Usa el algoritmo de cifrado `aes-128-cbc` para cifrar el archivo anterior usando la contraseña "*seguridad*" y usa la opción `-a` o `-base64` para generar un archivo cifrado en formato ASCII, que llamaremos `texto-aes-128-cbc-1.txt`
- Vuelve a cifrar el mismo archivo con la misma contraseña, y guarda el nuevo archivo cifrado en formato ASCII con el nombre `texto-aes-128-cbc-2.txt`
- Compara los dos archivos cifrados `texto-aes-128-cbc-1.txt` y `texto-aes-128-cbc-2.txt`. Observarás que son distintos ¿sabrías decir por qué?

Entrega #2: Entrega el archivo `texto-aes-128-cbc-1.txt`

Ejercicio 5

- Repite el ejercicio 4 pasando la contraseña como una argumento a la orden de cifrado/descifrado (opción `-pass pass:<password>`)

Ejercicio 6

- Repite el ejercicio 4 pasando la contraseña dentro de un archivo (opción `-pass file:<password-file>`)

A.4. Clave, contraseña y *salt*

Cuando usamos una contraseña para cifrar, por defecto el sistema genera un valor aleatorio denominado *salt*, de manera que la clave y el IV se generan a partir la contraseña del usuario y el valor del *salt.* La *salt* se adjunta siempre al archivo cifrado. Esto permite usar múltiples veces la misma contraseña, pero dando lugar a claves de cifrado distintas, lo que hace el mecanismo de cifrado más seguro.

El algoritmo de generación de la clave a partir de la contraseña y la salt se llama PBKDF2 (https://www.openssl.org/docs/man1.1.0/man3/PKCS5_PBKDF2_HMAC_SHA1.html)

Para ver los valores de *salt*, la clave generada y el IV generado, se puede añadir la opción `-p` a la orden de cifrado. Si no deseamos usar *salt*, se puede usar la opción `-nosalt`

Ejercicio 7

- Crea un archivo de texto llamado `textoplano1.txt`
- Usa el algoritmo de cifrado `aes-128-cbc` para cifrar varias veces el archivo anterior en formato binario usando la contraseña "*seguridad*". Usa la opción `-p` para ver el valor de *salt*, clave y IV generados cada vez que se cifra.
- Visualiza con la orden cat los archivos binarios cifrados y comprueba que todos empiezan con la palabra "`Salted__`"

Ejercicio 8

- Repite el ejercicio anterior, pero guardando los archivos cifrados en formato ASCII (opción `-a` o `-base64`) con distintos nombres. Comprueba que los archivos cifrados son distintos.

Entrega #3: Copia y entrega los valores *salt*, *IV* y clave de cifrado de la salida de uno de los comandos anteriores. Entrega también el contenido de uno de los archivos cifrados.

Ejercicio 9

- Repite el ejercicio anterior, pero usando la opción `-nosalt` (incluye también la opción `-p` para ver la clave y el IV). Si ciframos varias veces, observa que al no usar *salt*, siempre se genera la misma clave y el mismo IV. Guarda los archivos cifrados en formato ASCII con distinto nombre y comprueba que, en este caso, son idénticos.

A.5. Cifrado de flujo

Para cifrar, se usa el comando `enc` con un algoritmo de flujo (por ejemplo, `rc4`).

Ejercicio 10

- Crea un archivo de texto llamado `textoplano1.txt`
- Usa el algoritmo de cifrado `rc4` para cifrar archivo anterior en formato binario usando la contraseña "*seguridad*". Guarda el archivo cifrado con el nombre `texto-rc4.bin`. Usa la opción `-p` para ver el valor de *salt* y clave (observa que en con este cifrado no se usa IV).

- Descifra el archivo `texto-rc4.bin` usando el mismo algoritmo y contraseña y comprueba que el archivo descifrado es igual al original

Ejercicio 11

- Repite el ejercicio 10 pero guardando el archivo cifrado en formato ASCII (opción `-a` o `-base64`). Realiza el cifrado dos veces usando nombres distintos para los archivos cifrados (`texto-rc4-1.txt` y `texto-rc4-2.txt`). Comprueba que los archivos cifrados son distintos al usar distinto valor de *salt*.

Entrega #4: Copia y entrega los valores *salt*, *IV* y clave de cifrado de la salida de uno de los comandos anteriores. Entrega también el contenido de uno de los archivos cifrados.

Ejercicio 12

- Repite el ejercicio 11 usando la opción `-nosalt`. Comprueba que tanto la clave usada como los archivos cifrados son iguales.

A.6. Funciones *hash* y *HMAC*

Para obtener un resumen (*digest*) de un archivo, se usa el comando `dgst` con un algoritmo (`md5`, `sha1`, `sha256`…):

```
$ openssl dgst [-algoritmo] archivo
```

También se puede usar:

```
$ openssl [algoritmo] archivo
```

Ejercicio 13

- Copia los archivos `/etc/services` y `/etc/hosts` al directorio local. Observa el contenido de ambos y comprueba que el archivo `services` es mucho más largo que el archivo `hosts`
- Calcula dos veces el resumen (digest) del archivo `hosts` usando la función *hash* `sha1` y comprueba que ambos resúmenes, para un mismo archivo, son idénticos.
- Calcula el resumen (digest) del archivo `services` usando la misma función *hash* (`sha1`). Comprueba que, aunque el archivo `hosts` y el archivo `services` son de distinto tamaño, la longitud del resumen (*digest*) para ambos archivos es similar

Entrega #5: Copia y entrega los códigos hash generados en el ejercicio anterior

Ejercicio 14

- Repite el ejercicio 12 con otras funciones hash (ej. `sha256`, `sha512`, etc.) y comprueba las distintas longitudes de los resúmenes calculados.

La opción `-hmac` permite obtener un HMAC (*Keyed-Hash Message Authentication Code*) tal y como como se usa en RIP o en IPsec:

```
$ openssl dgst [-algoritmo] -hmac <key> archivo
```

Ejercicio 15

- Calcula códigos HMAC del archivo `/etc/services` con distintos algoritmos

usando la clave “seguridad”.

Entrega #6: Copia y entrega al menos dos códigos HMAC generados en el ejercicio anterior

B. GnuPG

B.1. Cifrado y descifrado

Revisa la página de manual de `gpg2`. Para ver las principales opciones y algoritmos de cifrados soportados, se puede usar la opción `--help`:

```
$ gpg2 --help
```

También se pueden ver los algoritmos de cifrado soportados usando la opción `--version`:

```
$ gpg2 --version
```

Para cifrar con clave simétrica, se usa la opción `--symmetric`:

```
$ gpg2 --symmetric [--cipher-algo <algoritmo>] [-a] file.txt
```

Por defecto, el archivo cifrado se llamará igual que el archivo plano, pero con extensión `.gpg` (es decir, `file.gpg`). Si queremos usar otro debemos usar la opción `-o <output_filename>` como primera opción de la orden.

Para guardar el archivo cifrado en formato ASCII en lugar de formato binario, se puede usar la opción `-a` (o `--armor`). En este caso, el archivo cifrado se llamará, por defecto, igual que el archivo original con textensión `.asc` (es decir, `file.asc`)

Para descifrar, se usa simplemente:

```
$ gpg2 file.gpg
```

O, si se usó la opción `-a`:

```
$ gpg2 file.asc
```

Ejercicio 16

- Crea un archivo de texto y utiliza GnuPG para cifrar/descifrar el archivo con distintos algoritmos de cifrado distintos.

Entrega #6: Entrega el resultado de cifrar el archivo `/etc/services` (copiándolo primero al directorio `$HOME`) con la contraseña “seguridad” y la opción `-a`.

B.2. Funciones *hash*

Para calcular el resumen (*digest*) de un archivo con GnuPG, se usa la siguiente orden:

```
$ gpg2 --print-md <función_hash> plain.txt
```

Las funciones *hash* soportadas se pueden ver usando las órdenes:

```
$ gpg2 --help
$ gpg2 --version
```

Ejercicio 17

- Calcula el resumen (*digest*) de varios archivos con GnuPG usando distintas funciones *hash*. (ej. `sha1`, `sha256`, `sha512`, etc.). Comprueba que dichos resúmenes son similares a los obtenidos con OpenSSL.

Entrega #7: Copia y entrega al menos dos códigos hash del archivo `/etc/services`.