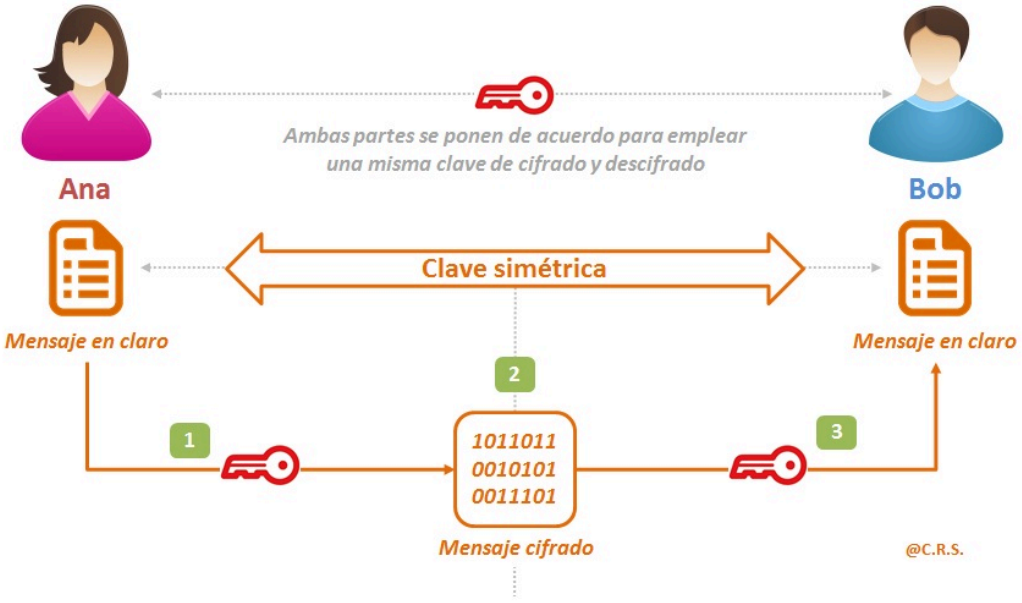
SAD - U2.2. Criptografía Simétrica

Descargar estos apuntes

Índice

1. Criptografía Simétrica



Los algoritmos de **cifrado simétrico, o clave privada**, utilizan la misma clave para encriptar y desencriptar datos. Los algoritmos implementan el conjunto de las operaciones matemáticas necesarias para encriptar un texto plano, o desencriptar un texto cifrado, utilizando una clave simétrica.

Si bien las implementaciones de software piden una contraseña de cifrado, no se utiliza dicha contraseña directamente. Generan una clave de cifrado de determinada cantidad de bits utilizando de base la contraseña provista, y una función de

derivación de clave.

Así, si `Ana` quiere encriptar un dato:

1. Introduce su contraseña.
2. Se crea una clave de cifrado derivada de dicha contraseña.
3. Se usa dicha clave para encriptar los datos.
4. `Ana` envía los datos cifrados a Bob.
5. `Bob` toma el dato cifrado, introduce la misma contraseña.
6. La función de derivación de clave genere la misma clave que `Ana`.
7. Se utiliza esta clave en el algoritmo para desencriptar el dato.

> **Warning**
>
> Ahora bien, las primitivas de encriptación y de desencriptación pueden o no ser las mismas.
>
> Esto significa que el algoritmo de encriptación puede ser diferente al de desencriptación, pero ambos deben ser capaces de trabajar con la misma clave.

Ventajas	Inconvenientes
Son rápidos y eficientes.	Exigen una clave diferente por cada pareja de interlocutores (el espacio de claves se incrementa enormemente conforme aumentan los interlocutores).
Resultan apropiados para el cifrado de grandes volúmenes de datos.	Requiere un control estricto sobre el intercambio seguro de la clave entre el emisor y el receptor.
	Son vulnerables a ataques por fuerza bruta, por lo que la fortaleza de la clave es fundamental.

1.1. Criptografía simétrica de flujo

Supongamos que queremos cifrar la palabra `balmis` utilizando la clave `0123`. Para ello necesitamos un algoritmo de cifrado. En este caso, supongamos que el algoritmo de cifrado, es decir, el conjunto de tareas que toma un texto plano de entrada, y una clave simétrica, y devuelve un texto cifrado, es la realización de la operación XOR entre los bits del texto plano y de la clave.

En nuestro caso, si traducimos los caracteres ASCII de la palabra `balmis` a su representación hexadecimal para facilitar la operación de bits, obtenemos lo siguiente:

```
ASCII:  b   a   l   m   i   s
Hexa:   62  61  6C  6D  69  73
```

Si hacemos lo mismo con la clave:

```
ASCII:  0   1   2   3
Hexa:   30  31  32  33
```

Ahora, procedemos a hacer el XOR entre cada valor hexadecimal.
Como la clave es más corta que la cadena, vamos a repetirla hasta completar el texto plano:

```
Texto plano:  balmis
Hexa balmis:        62  61  6C  6D  69  73
Hexa key "0123":    30  31  32  33  30  31
Resultado XOR:      52  50  5E  5E  59  42
ASCII cifrado:       R   P   ^   ^   Y   B
```

En este caso, la cadena cifrada que le envía `Ana` a `Bob` sería `RP^^YB`.

Ahora, veamos cómo descifrar el texto y obtener el mensaje original.

En este caso particular, el algoritmo de cifrado, XOR, es simétrica (no confundir con el concepto de criptografía simétrica en general), por lo que utilizaremos el mismo algoritmo para descifrar el contenido. Hagamos la prueba:

```
Texto cifrado:      R   P   ^   ^   Y   B
Hexa cifrado:       52  50  5E  5E  59  42
Hexa key "0123":    30  31  32  33  30  31
Resultado XOR:      62  61  6C  6D  69  73
ASCII descifrado:   b   a   l   m   i   s
```

A este tipo de algoritmos, que cifran pequeñas porciones de dato, como ser bytes, o words, se los denomina **cifradores de flujo o de stream**.

Entre los `algoritmos de cifrado simétrico de stream` más conocidos se encuentran **RC4, y Salsa20 y Chacha**. En general se trata de algoritmos más eficientes que los de bloque. Igualmente, por su forma de trabajo, suelen ser más propensos a ataques, por ejemplo, RC4 hoy se considera obsoleto.

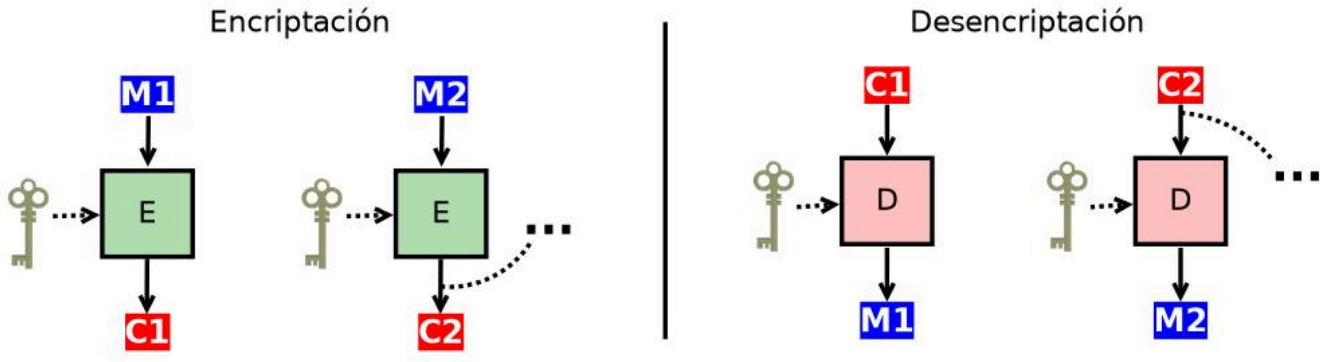
> **Función simétrica**
>
> Hemos utilizado la **misma clave para cifrar** el texto plano **y para descifrar** el texto cifrado.
>
> Además, las primitivas de encriptación y desencriptación de este algoritmo son la misma: la operación XOR.
> XOR es una función simétrica *(no confundir con el concepto de criptografía simétrica en general)*, por lo que permite utilizar el mismo algoritmo para las dos operaciones.

1.2. Criptografía simétrica de bloque

Existen otro tipo de cifradores simétricos que, en lugar de encriptar un flujo continuo de bits, dividen los datos en bloques de igual tamaño y los encriptan obteniendo bloques cifrados que luego se utilizarán para componer el texto cifrado.

En este caso el texto cifrado se genera encriptando bloques de tamaño fijo del texto plano. Los tamaños comunes de bloque son 128b, 256b, 384b, y 512b.

Estos cifradores generan, a partir de una contraseña y una función de derivación de clave, una clave del tamaño necesario para encriptar un bloque de datos. El tamaño de la clave, así, depende del tamaño del bloque seleccionado.



> Donde cada bloque de texto plano (M1, M2, …) genera texto cifrado (C1, C2, …) mediante el uso de un algoritmo de encriptación (E) y una clave simétrica. En la imagen de la derecha podemos ver el proceso con el algoritmo de desencriptación (D).

Los `algoritmos de cifrado simétrico de bloque` más conocidos son **AES, DES, 3DES, Twofish, y Blowfish**.

- AES (Advanced Encryption Standard) es un estándar de criptografía industrial. Ofrece tres tipos de longitudes de claves: 128, 192 y 256 bits.
- Camellia Desarrollado en Japón, crea claves de 128, 192 y 256 bits. Es utilizado por TLS.
- Triple DES Diseñado para agrandar el largo de la clave sin necesidad de cambiar el algoritmo Data Encryption Standard (DES), tarjetas de crédito y otros medios de pago electrónicos tienen como estándar el algoritmo Triple DES pero está desapareciendo lentamente, siendo reemplazado por AES que es hasta 6 veces más rápido.
- Twofish El tamaño de bloque en Twofish es de 128 bits y el tamaño de clave puede llegar hasta 256 bits. Es rápido, flexible y eficiente y es utilizado en las herramientas de gestión de contraseñas y sistemas de pago seguros.

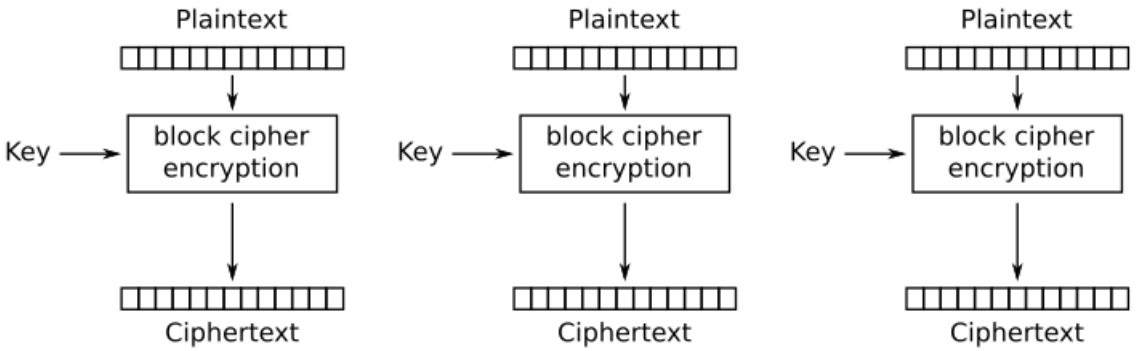
Estos cifradores suelen **encadenar** las operaciones de cifrado/descifrado de bloques de tal forma que se dificulte el trabajo a un atacante. Esta manera de encadenamiento se denomina **modo de operación** del algoritmo.

Cada uno de estos modos poseen sus ventajas y desventajas, algunos son paralelizables mientras que otros no, algunos carecen de difusión (propiedad criptográfica) y otros no, etc.

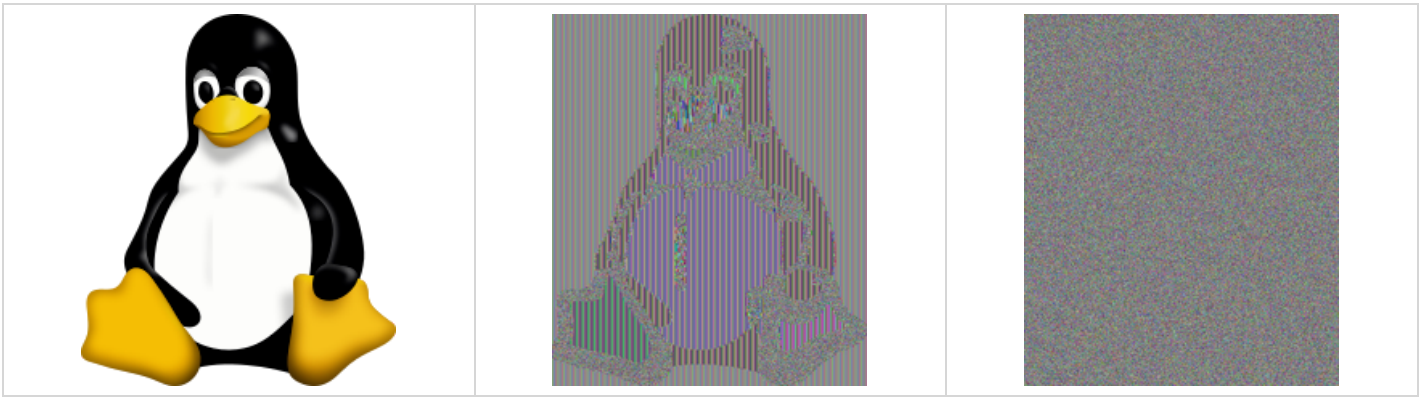
ECB (Electronic Code Book)

En este modo, cada bloque de texto plano se encripta de forma independiente. Es decir, el primer bloque de texto plano se encripta con la clave y se obtiene un bloque de texto cifrado, el segundo bloque de texto plano se encripta con la misma clave y se obtiene un segundo bloque de texto cifrado, y así sucesivamente.

Este modo es muy sencillo de implementar, pero presenta un problema: si dos bloques de texto plano son iguales, los bloques de texto cifrado resultantes también lo serán. Esto puede ser aprovechado por un atacante para obtener información sobre el texto plano.



Electronic Codebook (ECB) mode encryption



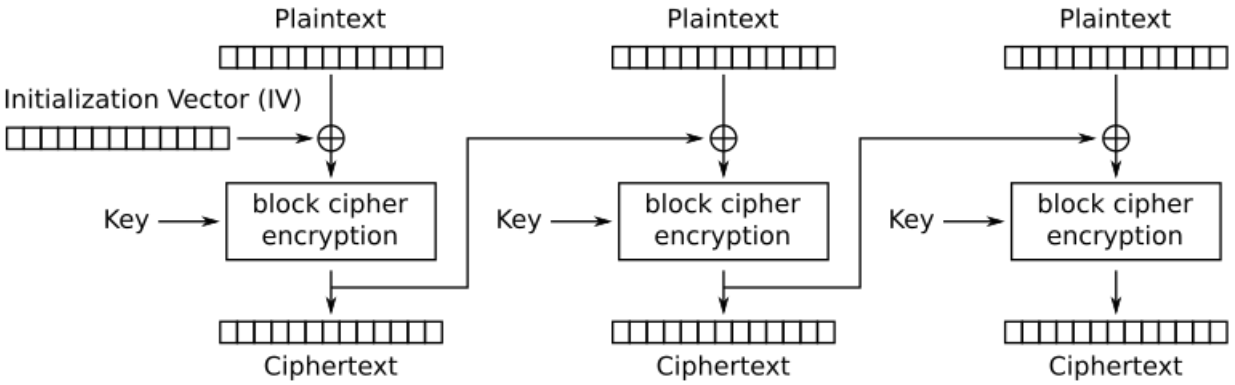
Difusión

Se dice que este método carece de **difusión**. La difusión es una propiedad de los algoritmos criptográficos que implica modificar la mayor cantidad posible de bits del texto cifrado al cambiar un bit del texto plano.

CBC (Cipher Block Chaining)

En este modo, cada bloque de texto plano se encripta con el bloque de texto cifrado anterior. Es decir, el primer bloque de texto plano se encripta con la clave y se obtiene un bloque de texto cifrado, el segundo bloque de texto plano se encripta con la clave y se obtiene un segundo bloque de texto cifrado, pero antes de encriptar este segundo bloque, se realiza una operación XOR entre el segundo bloque de texto plano y el primer bloque de texto cifrado.

Este modo es más seguro que el modo ECB, ya que no se pueden obtener patrones en el texto cifrado. Además, es más difícil de paralelizar, lo que dificulta el trabajo de un atacante.



Cipher Block Chaining (CBC) mode encryption

Difusión vs Paralelización

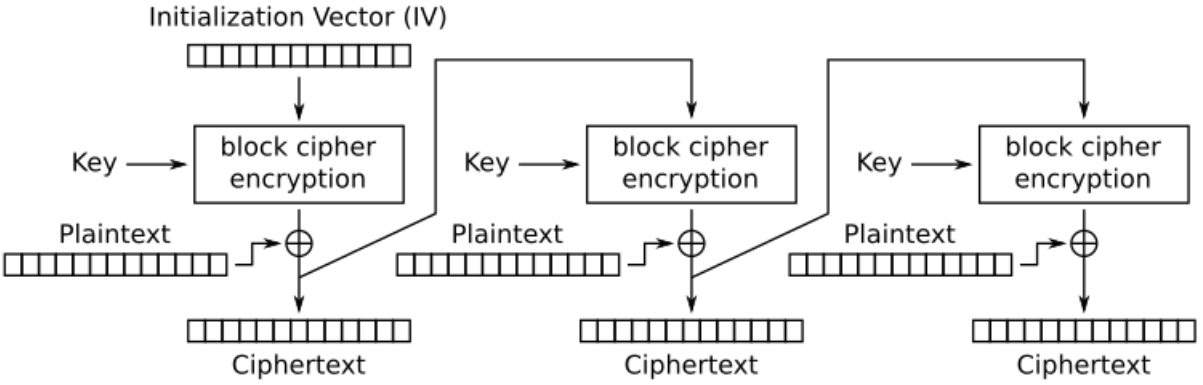
La ventaja de este modo (y de todos los demás) es que genera diferentes bloques cifrados para dos bloques de texto plano iguales, lo que incrementa la difusión del algoritmo.

Como desventaja, la encriptación en este modo no es paralelizable, por lo que el procesamiento es secuencial.

CFB (Cipher Feedback)

En este modo, el cifrador se utiliza como un generador de secuencia de bits. El cifrador se inicializa con la clave y se encripta un bloque de texto plano. El resultado se encripta de nuevo con la clave y se obtiene un bloque de texto cifrado. Este bloque de texto cifrado se utiliza para encriptar el siguiente bloque de texto plano.

Este modo es muy similar al modo CBC, pero en lugar de encriptar el bloque de texto plano, se encripta el bloque de texto cifrado anterior.



Cipher Feedback (CFB) mode encryption

Otros modos

Existen otros modos de operación, como **OFB (Output Feedback)**, **CTR (Counter)**, **GCM (Galois/Counter Mode)**, **XTS (XEX-based tweaked-codebook mode with ciphertext stealing)**, entre otros.

> Cada uno de estos modos posee sus ventajas y desventajas, algunos son paralelizables mientras que otros no, algunos carecen de difusión (propiedad criptográfica) y otros no, etc.

Podemos ver un extracto de los modos de operación de los cifradores de openssl en la siguiente imagen:

```
C:\Users\Vicente>openssl enc -ciphers
Supported ciphers:
-aes-128-cbc               -aes-128-cfb               -aes-128-cfb1
-aes-128-cfb8              -aes-128-ctr               -aes-128-ecb
-aes-128-ofb               -aes-192-cbc               -aes-192-cfb
-aes-192-cfb1              -aes-192-cfb8              -aes-192-ctr
-aes-192-ecb               -aes-192-ofb               -aes-256-cbc
-aes-256-cfb               -aes-256-cfb1              -aes-256-cfb8
-aes-256-ctr               -aes-256-ecb               -aes-256-ofb
-aes128                    -aes128-wrap               -aes192
-aes192-wrap               -aes256                    -aes256-wrap
-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
-aria128                   -aria192                   -aria256
-bf                        -bf-cbc                    -bf-cfb
-bf-ecb                    -bf-ofb                    -blowfish
```

Cifradores simétricos

Actividad U2.1: Calcula cuantas claves necesitan intercambiar un grupo de cuatro personas que se quieran enviar correos electrónicos cifrados entre ellas, tanto para el caso de utilizar criptografía simétrica como asimétrica. ¿Y si el grupo lo conforman seis personas? ¿Y si son mil?

Actividad U2.2: Investiga sobre los cifradores simétricos más utilizados en la actualidad. ¿Qué algoritmos son los más seguros? ¿Qué algoritmos son los más rápidos? ¿Qué algoritmos son los más eficientes? ¿Qué tamaño de clave permite cada uno de ellos? ¿Cuál es su principal uso?

Crea una tabla comparativa con los resultados obtenidos.

2. Criptografía simétrica con OpenSSL

OpenSSL es una biblioteca de software que implementa protocolos y algoritmos de cifrado seguros. Es ampliamente utilizada en aplicaciones de software para garantizar la seguridad de la información.

2.1. Instalación de OpenSSL

2.1.1. Instalación en Windows

WINDOWS
Para la instalación de OpenSSL en Windows, se debe de realizar correctamente los pasos que se describen a continuación. Además, al instalar OpenSSL puede que el sistema requiera usar Microsoft Visual C++ [Descarga].

1. Descargar la última versión de OpenSSL desde la página de slproweb.

2. Ejecutar el archivo .exe descargado y mantener la configuración que aparece por defecto excepto cuando pida copiar las DLL de OpenSSL que se seleccionará la opción: `The OpenSSL binaries (/bin) directory`.
3. Ejecutar la consola `sysdm.cpl` y en la penstaña de *Opciones Avanzadas*
   i. Añadir `C:\Program Files\OpenSSL-Win64` y `C:\Program Files\OpenSSL-Win64\bin` al PATH del sistema.
   ii. Crear una nueva variable de usuario llamada `OPENSSL_CONF` con el valor `C:\Program Files\OpenSSL-Win64\bin\cnf`.
   iii. Abrir una nueva consola y ejecutar `openssl version` para comprobar que la instalación ha sido correcta.

2.1.2. Instalación en GNU/Linux

En la mayoría de distribuciones GNU/Linux, las librerías y utilidades de OpenSSL ya vienen preinstaladas.

En cualquier caso, para instalar OpenSSL en sistemas basados en Debian, como Ubuntu, realizamos las siguientes acciones:

1. Averiguar cuál es y descargar la última versión de OpenSSL:
   Para averiguar cuál es la última versión de OpenSSL hay que acceder a https://www.openssl.org/source y apuntar el nombre del fichero en la primera línea de la tabla de descargas
   A fecha de la última actualización de este documento la versión disponible es la **openssl-3.4.0.tar.gz** que es la que usaremos en este ejemplo.

> ⚠ **Entorno y librerías**
>
> Para compilar OpenSSL necesitamos tener instalado el paquete `build-essential` y las librerías de desarrollo de `zlib`.
> Si no están instaladas (falla el comando config del paso 3), podemos hacerlo con el siguiente comando:
>
> ```
> sudo apt-get update && sudo apt-get upgrade
> sudo apt install build-essential checkinstall zlib1g-dev -y
> ```

2. Descargar, descomprimir e instalar la última versión de OpenSSL:

```
cd /tmp
wget https://www.openssl.org/source/openssl-3.4.0.tar.gz
tar xvf openssl-3.4.0.tar.gz
```

3. Compilar, probar e instalar OpenSSL:

```
cd openssl-3.4.0
sudo ./config --prefix=/usr/local/ssl --openssldir=/usr/local/ssl shared zlib
sudo make
sudo make test
sudo make install
```

4. Configurar las librerías de OpenSSL:

```
sudo sh -c "echo /usr/local/ssl/lib >> /etc/ld.so.conf.d/openssl-3.4.0.conf"
sudo ldconfig -v
```

5. Configurar el PATH añadiendo `/usr/local/ssl/bin` al principio de la variable de entorno PATH, esto hará que el comando openssl se busque antes en el directorio de nuestra instalación que en el ya existente (típicamente `/usr/bin):

```
sudo sh -c "echo \"PATH=/usr/local/ssl:$PATH\" >> /etc/profile"
```

6. Salir de la sesión y volver a entrar para que los cambios en el PATH surtan efecto. Una vez hecho esto, comprobamos que la instalación ha sido correcta:

```
openssl version
```

2.2. Uso de OpenSSL para cifrado simétrico

Para proceder a cifrar o descifrar mensajes se hará uso del comando `enc`. Se utiliza tanto para cifrados de bloque como de flujo, haciendo uso de **claves basadas en contraseñas o claves proporcionadas explícitamente** (clave y vector de inicialización, en su caso).

El subcomando `enc -list` lista los algoritmos simétricos disponibles en la instalación de OpenSSL.

```
$ openssl enc -list
aes-128-cbc               -aes-128-cfb              -aes-128-cfb1
-aes-128-cfb8              -aes-128-ctr              -aes-128-ecb
-aes-128-ofb               -aes-192-cbc              -aes-192-cfb
-aes-192-cfb1              -aes-192-cfb8             -aes-192-ctr
-aes-192-ecb               -aes-192-ofb              -aes-256-cbc
-aes-256-cfb               -aes-256-cfb1             -aes-256-cfb8
-aes-256-ctr               -aes-256-ecb              -aes-256-ofb
-aes128                    -aes128-wrap              -aes128-wrap-pad
-aes192                    -aes192-wrap              -aes192-wrap-pad
-aes256                    -aes256-wrap              -aes256-wrap-pad
-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
-aria128                   -aria192                  -aria256
-bf                        -bf-cbc                   -bf-cfb
-bf-ecb                    -bf-ofb                   -blowfish
-camellia-128-cbc          -camellia-128-cfb         -camellia-128-cfb1
-camellia-128-cfb8         -camellia-128-ctr         -camellia-128-ecb
-camellia-128-ofb          -camellia-192-cbc         -camellia-192-cfb
-camellia-192-cfb1         -camellia-192-cfb8        -camellia-192-ctr
-camellia-192-ecb          -camellia-192-ofb         -camellia-256-cbc
-camellia-256-cfb          -camellia-256-cfb1        -camellia-256-cfb8
-camellia-256-ctr          -camellia-256-ecb         -camellia-256-ofb
-camellia128               -camellia192              -camellia256
-cast                      -cast-cbc                 -cast5-cbc
-cast5-cfb                 -cast5-ecb                -cast5-ofb
-chacha20                  -des                      -des-cbc
-des-cfb                   -des-cfb1                 -des-cfb8
-des-ecb                   -des-ede                  -des-ede-cbc
-des-ede-cfb               -des-ede-ecb              -des-ede-ofb
-des-ede3                  -des-ede3-cbc             -des-ede3-cfb
-des-ede3-cfb1             -des-ede3-cfb8            -des-ede3-ecb
-des-ede3-ofb              -des-ofb                  -des3
-des3-wrap                 -desx                     -desx-cbc
-id-aes128-wrap            -id-aes128-wrap-pad       -id-aes192-wrap
-id-aes192-wrap-pad        -id-aes256-wrap           -id-aes256-wrap-pad
-id-smime-alg-CMS3DESwrap  -idea                     -idea-cbc
-idea-cfb                  -idea-ecb                 -idea-ofb
-rc2                       -rc2-128                  -rc2-40
-rc2-40-cbc                -rc2-64                   -rc2-64-cbc
-rc2-cbc                   -rc2-cfb                  -rc2-ecb
-rc2-ofb                   -rc4                      -rc4-40
-seed                      -seed-cbc                 -seed-cfb
-seed-ecb                  -seed-ofb                 -sm4
-sm4-cbc                   -sm4-cfb                  -sm4-ctr
-sm4-ecb                   -sm4-ofb
```

En esta lista pueden verse algunos datos importantes: **algoritmos** (AES, CAST, DES, BF, etc.), muchos de ellos con sus **tamaños de clave** asociados (128, 192, 256, etc.) y, si se trata de cifradores simétricos de **bloque**, el **modo de operación**

de dichos algoritmos (ECB, CBC, CTR, OFB, CFB, etc.).

> **Si no se especifica un modo de operación, se utilizará el modo ECB por defecto**.

Estos algoritmos son los que debemos elegir más adelante para encriptar y, posteriormente, desencriptar un archivo.

> **Salt & Pepper**
>
> Cuando vamos a encriptar o desencriptar archivos usando herramientas como OpenSSL, introducimos una contraseña, o una passphrase, que solo nosotros y el destinatario del mensaje conocemos.
>
> Ahora bien, los algoritmos y sus claves son de un tamaño predeterminado: 128 bits, 256 bits, etc.
> Si la clave que vamos a usar es de 128 bits, **¿es requisito que nuestra contraseña de cifrado/descifrado tenga ese tamaño?**
>
> La realidad es que no, nosotros ponemos contraseñas complejas o simples, y la implementación criptográfica genera, mediante una **función de derivación de clave**, una clave de determinado tamaño para usar en el cifrado. Por supuesto, si a la función de derivación de clave le introducimos la misma contraseña, generará la misma clave.
>
> Esta función de derivación de clave, o KDF (Key Derivation Function) depende de la implementación de la aplicación criptográfica. Existen varias implementaciones, pero una de las más utilizadas es PBKDF2.
>
> key = pbkdf2(PRF, Pass, Salt, Iter, Size)
>
> > PRF: Pseudo-Random Function / Pass: Contraseña / Salt: Sal criptográfica / Iter: Iteraciones (10000) / Size: Tamaño de la clave
>
> Cuando se cifra con contraseña, OpenSSL pasa esta contraseña, junto con 64 bits aleatorios de `sal criptográfica`, por un algoritmo de derivación de claves como el PBKDF1 (obsoleto) o el PBKDF2 (explicado en las "opciones de enc").
>
> Estos algoritmos utilizan funciones de resumen de forma recursiva para obtener suficientes bits para la clave de cifrado y para el vector de inicialización si es necesario. Esto permite que para una misma contraseña, se generen diferentes claves, al variar en cada caso la sal obtenida aleatoriamente.
>
> Salts and hashes: How apps and websites protect your passwords
> Linux-style shadow file

2.2.1. Cifrado simétrico de bloque

Para cifrar un archivo de texto plano, utilizamos el siguiente comando:

```
$ echo "Hola, mundo" > mensaje.txt
$ openssl enc -e -aes-256-cbc -salt -in mensaje.txt -out mensaje.enc
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:

*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

En este caso, el comando `enc` cifra **(-e)** el archivo `mensaje.txt` **(-in)** utilizando el algoritmo AES con una clave de 256 bits en modo CBC **(-aes-256-cbc)**. El archivo cifrado se guarda en `mensaje.enc` **(-out)**.

Podemos observar el mensaje de que se está utilizando una derivación de clave obsoleta. Para evitarlo, podemos utilizar la opción `-pbkdf2` para usar la función de derivación de clave PBKDF2.

Para descifrar el archivo cifrado, utilizamos el siguiente comando:

```
$ openssl enc -d -p -aes-256-cbc -in mensaje.enc -out mensaje.dec
enter aes-256-cbc decryption password:
salt=86239987878B82C6
key=37081952A242A1720D09A0AB511DD7AE34F90AAF1EFAB9A90B76FA0C6044C95C
iv =629F89ACCDE7F852B54AD9A0AA4F2B0E
```

En este caso, el comando `enc` descifra **(-d)** el archivo `mensaje.enc` **(-in)** utilizando el algoritmo AES con una clave de 256 bits en modo CBC **(-aes-256-cbc)**. El archivo descifrado se guarda en `mensaje.dec` **(-out)**. Además, hemos usado el modificador `-p` para que se muestre información adicional (Salt, Key, IV) sobre el proceso de cifrado/descifrado.

El archivo `mensaje.enc` es un archivo binario que contiene el texto cifrado. Si intentamos abrirlo con un editor de texto, o con el comando `type` (windows) o `cat` (Linux) veremos caracteres extraños.

Si queremos generar un archivo cifrado, pero que solo contenga caracteres ASCII, podemos utilizar la opción `-a` para que el archivo cifrado sea codificado en base64.

```
$ openssl enc -e -aes-256-cbc -pbkdf2 -a -in mensaje.txt
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:

U2FsdGVkX1/bxhJnZxZH39gxnhOd7/QyB2yDcexnLE2nnlJ1+EZH3ka3z2oHOvMR
TQk6dQ7CwLJ7tLIFeUId1jbvYVxgMTGE99kJHikaD4x8Kf8Jw3/LsQmRPLRHnUiZ
```

En el ejemplo anterior ya se ha usado la opción `-kbkdf2` para evitar el mensaje de advertencia y suar un algoritmo de derivación de clave más seguro. También se ha usado la opción `-a` para que el archivo cifrado sea codificado en base64 y se ha omitido la opción `-out` para que el resultado se muestre por pantalla.

TRIPLE DES

DES es el algoritmo prototipo del cifrado por bloques. Su clave es de 56 bits, lo que lo hace vulnerable a ataques de fuerza bruta. Para aumentar la seguridad, se creó **3DES**, que aplica DES tres veces a cada bloque de datos. Aunque es más seguro, es más lento que DES.

La mayoría de las tarjetas de crédito y otros medios de pago electrónicos tienen como estándar el algoritmo **Triple DES** (anteriormente usaban el DES). Por su diseño, el DES y por lo tanto el 3DES son algoritmos lentos. AES puede llegar a ser hasta 6 veces más rápido y a la fecha no se ha encontrado ninguna vulnerabilidad

Existen diferentes tipos de TDES, aunque el más utilizado es el EDE3 (Encrypt-Decrypt-Encrypt) con dos claves DES diferentes. En el siguiente ejemplo se cifra un archivo con 3DES:

```
openssl enc -des-ede3-cbc -p -in mensaje.txt -out des_ede3_cbc.enc

salt=AB5835474350C692
key=D9A1F0F149CB70901A2411C5607BEA89C9DFF9786B018DF7B2158967D42B0D4D
iv =4AE3E7B85BE8AE5C752187CFC0CC9672
```

Y a continuación se descifra:

```
openssl enc -d -des-ede3-cbc -p -in des_ede3_cbc.enc

salt=AB5835474350C692
key=D9A1F0F149CB70901A2411C5607BEA89C9DFF9786B018DF7B2158967D42B0D4D
iv =4AE3E7B85BE8AE5C752187CFC0CC9672
"Hola, mundo"
```

Note

También podemos descifrar el mensaje sin conocer la contraseña, si conocemos el resto de los parámetros (salt, key, iv). Esto es útil para recuperar mensajes cifrados en caso de pérdida de la contraseña.

```
openssl enc -d -des-ede3-cbc -in des_ede3_cbc.enc -K D9A1F0F149CB70901A2411C5607BEA89C9DFF9786B018DF7B2158967D4
ÃhÁs{N╣µ·¦mtæ┬óh=åÅundo"
```

Como vemos, el SALT que se ha añadido al mensaje cifrado, ha modificado el mensaje original. Assí que, para recuperar el mensaje original, necesitamos eliminar el SALT del mensaje cifrado.

- Editamos el archivo cifrado (es binario) en modo hexadecimal y eliminamos de la cabecera los primeros 8 bytes ("Salted__") y los siguientes 8 bytes (el valor del SALT).
- En linux lo podemos hacer con `dd`:

```
cat des_ede3_cbc.cif | dd ibs=16 obs=16 skip=1 > des_ede3_cbc_nosalt.enc
```

```
Address  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 53 61 6c 74 65 64 5f 5f dd 63 00 18 39 34 91 fa Salted__Ýc..94'ú
00000010 4b 96 81 9c c3 b7 c7 78 1e 08 0a 55 8c cb 2e 05 K-.œÃ·Çx...UŒË..
00000020 53 c1 69 de d2 1a b9 ab d8 e1 21 b9 2a e1 66 9e SÁiÞÒ.¹«Øá!¹*áfž
00000030 69 d4 e5 37 23 6d 9f 76 4f dd 13 38 dd e2 e3 a3 iÔå7#mŸvOÝ.8Ýâã£
00000040 9b f3 f3 fd e0 8a 3d af 63 6e cc e8 9a 8f ac a7 ›óóýàŠ=¯cnÌèš.¬§
```

Una vez eliminado el SALT, procedemos a descifrar el archivo usando la clave y el vector de inicialización.

```
openssl enc -d -des-ede3-cbc -in des_ede3_cbc_nosalt.enc -K D9A1F0F149CB70901A2411C5607BEA89C9DFF9786B018DF7B21

"Hola, mundo"
```

AES (Advanced Encryption Standard)

AES, también conocido como Rijndael (pronunciado "Rain Doll" en inglés), es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos. Desde 2006, el AES es uno de los algoritmos más populares usados en criptografía simétrica.

Daemen & Rijmen

El cifrado fue desarrollado por dos criptólogos belgas, Joan Daemen y Vincent Rijmen, ambos estudiantes de la Katholieke Universiteit Leuven, y fue enviado al proceso de selección AES bajo el nombre "Rijndael".

Fueron los ganares del concurso AES, y el algoritmo fue seleccionado como el estándar de cifrado simétrico por el Instituto Nacional de Estándares y Tecnología (NIST) en 2001.

Se trata de un algoritmo del que **todavía no se ha registrado ningún ataque factible hacia él**, convirtiéndose en un estándar de cifrado para las principales organizaciones como bancos, gobiernos y sistemas de alta seguridad en todo el mundo. Además, es un estándar mucho más rápido que los vistos anteriormente.

Basado en la estructura de bloques de AES, el cambio de un solo bit, ya sea en la clave, o en el bloque de texto sin cifrado, da como resultado un bloque de texto cifrado completamente diferente. Este algoritmo tiene una **longitud de bloque de 128 bits y longitudes de clave de 128, 192 y 256**.

En el primer ejemplo del apartado anterior, hemos cifrado un archivo de texto plano con AES y una clave de 256 bits en modo CBC.

2.2.2. Cifrado simétrico de flujo

Para algunas aplicaciones, tales como el cifrado de conversaciones telefónicas, el cifrado en bloques es inapropiada porque los flujos de datos se producen en tiempo real en pequeños fragmentos. Las muestras de datos pueden ser tan pequeñas como 8 bits o incluso de 1 bit, y sería un desperdicio rellenar el resto de los 64 bits antes de cifrar y transmitirlos.

Los cifradores de flujo son algoritmos de cifrado que pueden realizar el cifrado incrementalmente, convirtiendo el texto en claro, en texto cifrado bit a bit, es decir, cada dígito de texto sin formato se cifra uno a la vez con el dígito correspondiente de la secuencia pseudoaleatoria, para dar un dígito del flujo de texto cifrado. Una secuencia pseudoaleatoria es una secuencia de bits de tamaño arbitrario que puede emplearse para oscurecer los contenidos de un flujo de datos combinando esta secuencia con el flujo de datos mediante la función XOR. Si la secuencia pseudoaleatoria es segura, el flujo de datos cifrados también lo será. Los bloques se cifran empleando una clave compartida por el emisor y el receptor.

Uno de los algoritmos de flujo más utilizados es el RC4 (ya obsoleto), reemplazado en la actualidad por algoritmos más eficientes como el **Chacha20** y los modos de operación en flujo (OFB o CFB) de los cifradores de bloque como **TDES**, **AES**, etc.

RC4

Dentro de la criptografía RC4 o ARC4 es el sistema de cifrado de flujo Stream cipher más utilizado y se utilizó en algunos de los protocolos más populares como **Secure Sockets Layer (SSL)** para proteger el tráfico de Internet y **Wired Equivalent Privacy (WEP)** para añadir seguridad en las redes inalámbricas.

En los últimos años, RC4 se ha excluido de los estándares de alta seguridad por sus debilidades. No está recomendado su uso en los nuevos sistemas y el estándar **TLS de cifrado web** y los protocolos **WPA de cifrado en Wifi** lo excluyen.

Chacha20

Es un sistema de cifrado en flujo, que soporta `claves de 128 y 256 bits` y de alta velocidad creado por Bernstein en 2008. Salsa20 es el cifrado original creado también por Bernstein en 2007, el cuál mantiene una estrecha relación con su sucesor, ya que ambos cifrados se basan en una función pseudoaleatoria basada en operaciones add-rotate-xor (ARX). Salsa20 y ChaCha poseen la inusual ventaja de que el usuario puede buscar de manera eficiente cualquier posición en el flujo de claves en tiempo constante.

La principal diferencia entre ellos, es que Chacha20 ofrece un aumento de la difusión por ronda y logra ligeramente un mejor rendimiento. Además, se considera que en implantaciones software es más eficiente y rápido que AES.

Se cifra el texto.txt con Chacha20 usando esta vez la función -pass pass:password **(desaconsejada)**.

```
openssl enc -chacha20 -a -pbkdf2 -pass pass:balmis -in texto.txt -out chacha20.cif

U2FsdGVkX19A+Yp7zsXDMD/VDIsgYsdM5nxu037Ku0a8zlyCocLhq9G6t+UPSDug
N98wSwM0RPfBOUVyEKMk99hWwNFWKESgg0WCFLXi
```

Para descifrar ejecutamos el siguiente comando incluyendo la contraseña como se hizo para cifrar:

```
openssl enc -chacha20 -d -a -pbkdf2 -pass pass:balmis -in chacha20.cif

"Hola, mundo"
```