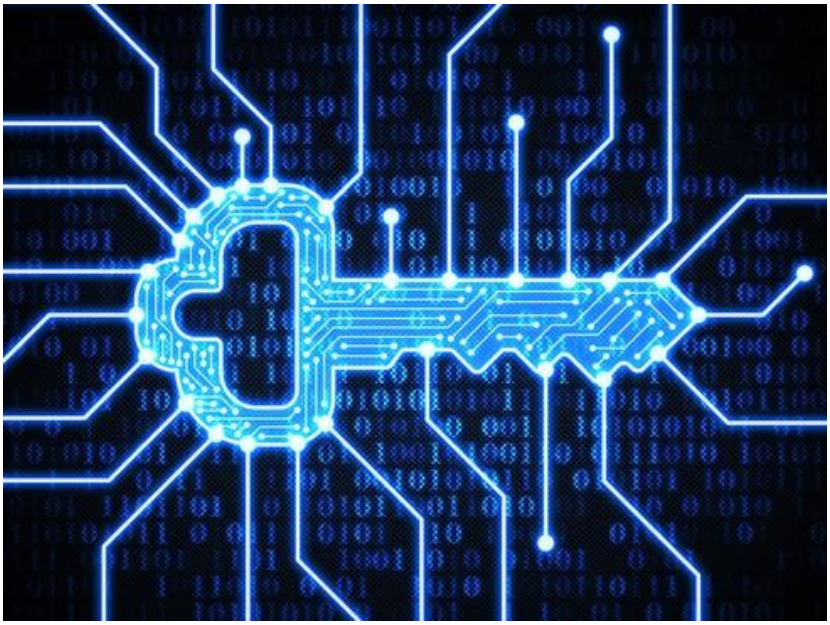
Práctica 1

CRIPTOSISTEMAS SIMÉTRICOS



Manuel Ruiz Maldonado

Índice

En la realización de esta práctica vamos a utilizar OpenSSL. La versión sobre la que están realizados estos ejercicios es la OpenSSL 1.0.1f 6 Jan 2014.

Ejercicio 1

Partiremos de un archivo binario de 1024 bits, todos ellos con valor 0. Para hacer referencia al mismo voy a suponer que se llama input.bin, pero podéis dar el nombre que os convenga.

Lo primero que hago es crear el archivo binario vacío con el comando:

```
$$ touch input.bin
```

Para editarlo y escribir los 1024 bits todos con valor 0 voy a utilizar un editor hexadecimal, ghex (ver Imagen 1). Abro el programa, abro el fichero input.bin y en la pestaña editar elijo "Modo insertar". A continuación escribo tantos ceros como sea necesario para llegar a 1024 bits (desplazamiento = 0x7F ya que corresponde al desplazamiento 128 que por 8 bits que tiene cada uno son los 1024 bits requeridos).

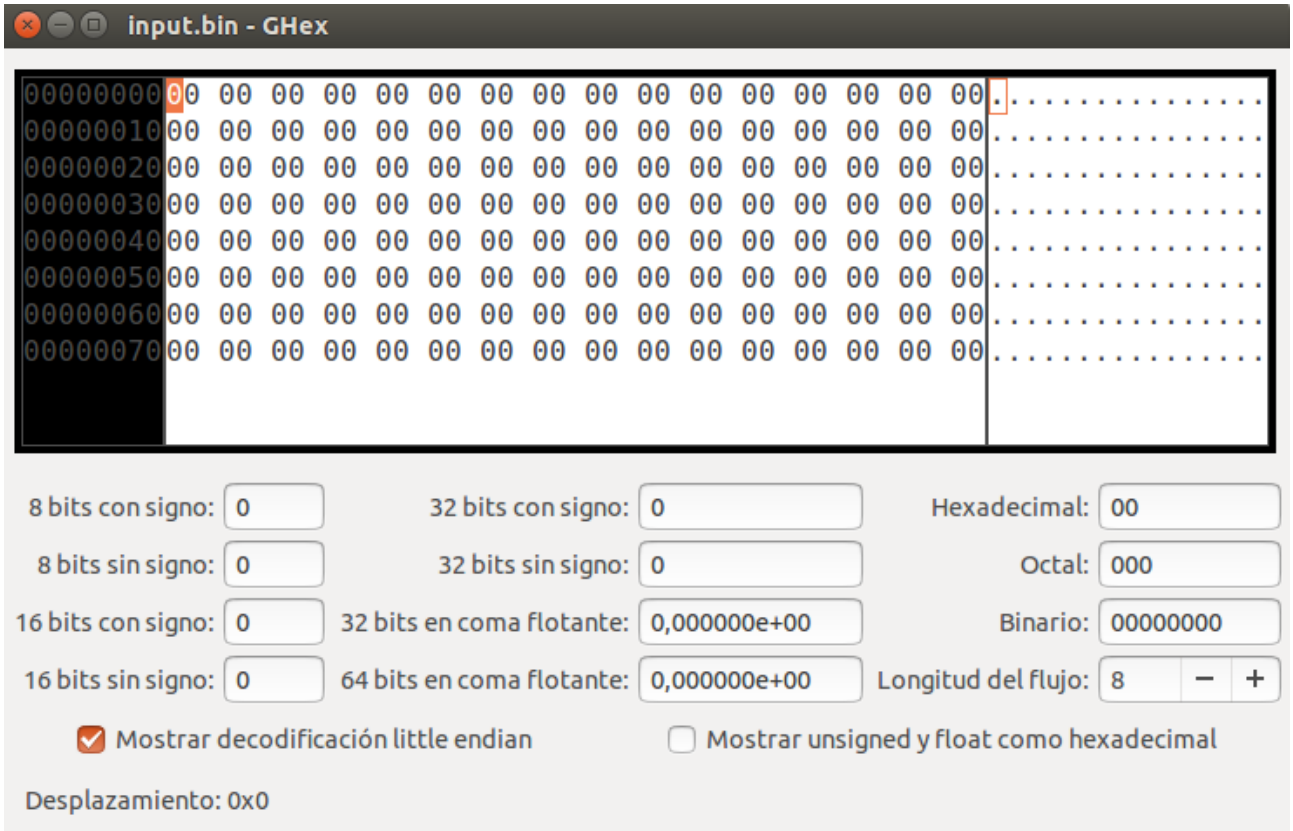


Imagen 1: Contenido del fichero input.bin.

Para comprobar que el tamaño del fichero son 1024 bits utilizo el comando ls -l, el cual muestra el tamaño en bytes, 128, que son justamente 1024 bits (ver Imagen 3).

Ejercicio 2

Creamos otro archivo binario del mismo tamaño, que contenga un único bit con valor 1 dentro de los primeros 40 bits y todos los demás con valor 0. Me referiré a este archivo como input1.bin.

Del mismo modo creo el fichero nuevo. Lo edito con la diferencia de que en lugar de añadir todo ceros, en este caso añado un 1 dentro de los 40 primeros bits, o equivalentemente, un 1 dentro de los diez primeros caracteres hexadecimales (ver Imagen 2).

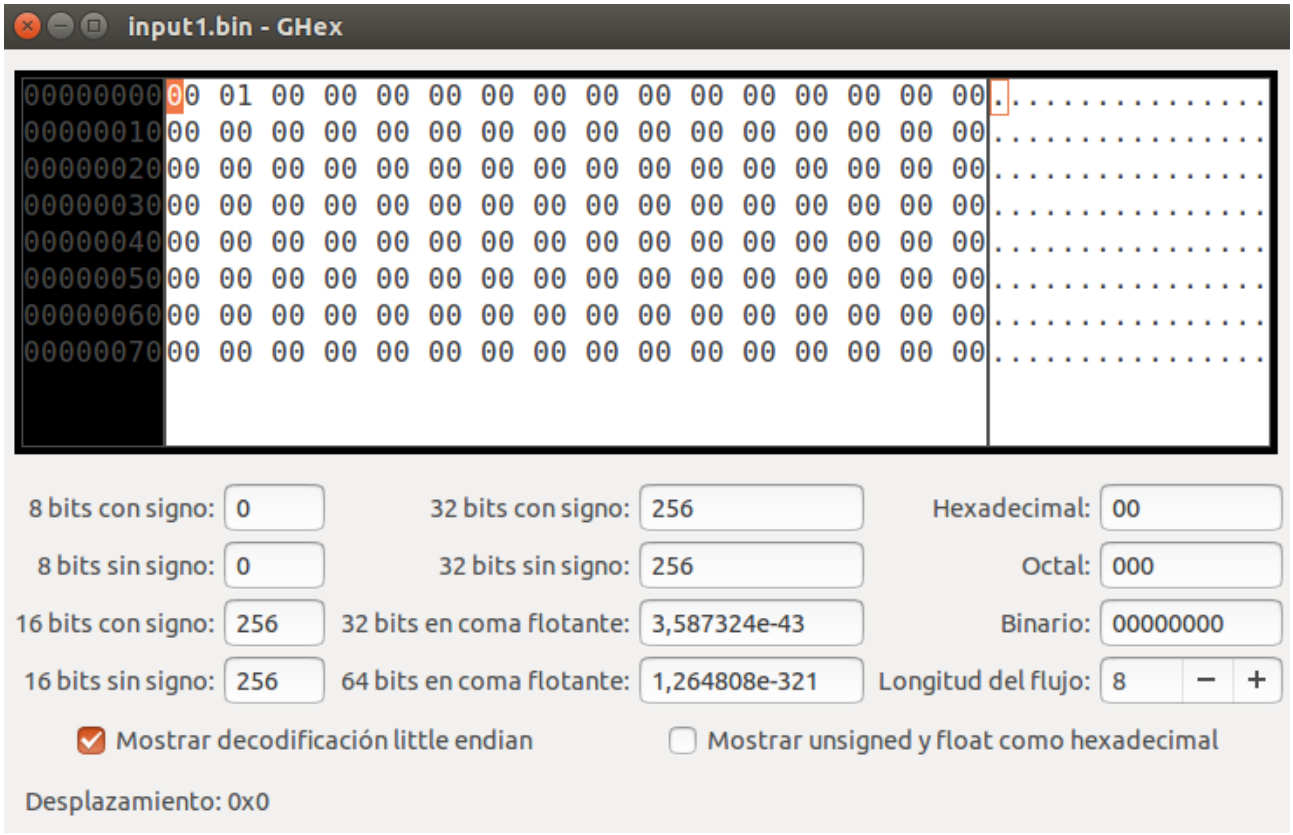


Imagen 2: Fichero input1.bin.

```
manolo@manolo-Parallels:~/Documentos/SPSI$ ls -l
total 12
-rw-rw-r-- 1 manolo manolo  85 sep 29 09:25 enlaces.txt
-rw-rw-r-- 1 manolo manolo 128 sep 29 09:22 input1.bin
-rw-rw-r-- 1 manolo manolo 128 sep 29 09:19 input.bin
-rw-rw-r-- 1 manolo manolo   0 oct 14 14:06 README.txt
manolo@manolo-Parallels:~/Documentos/SPSI$
```

Imagen 3: Tamaño de los ficheros creados.

Ejercicio 3

Cifrad input.bin con DES en modos ECB, CBC y OFB usando como claves una débil y otra semidébil, con vector de inicialización a vuestra elección, y explicad los diferentes resultados.

DES es un cifrado por bloques, cuya entrada es un bloque de 64 bits y la salida tiene la misma longitud. Esto quiere decir que para cifrar un fichero, en nuestro caso de 1024 bits, el fichero se dividirá en bloques de 64 bits que se cifrarán según el modo escogido en cada caso.

Para cifrar utilizando DES es necesario utilizar una clave cuya longitud sea la misma que el tamaño de los bloques. Además, para los modos de cifrado CBC y OFB también es necesario aportar un vector de inicialización, IV, también del mismo tamaño que los bloques.

Hay claves débiles y semidébiles que son conocidas, y que para usarlas en este ejercicio las voy a tomar de [1] (ver Imagen 4 e Imagen 5).

01 01 01 01 01 01 01 01
1F 1F 1F 1F 0E 0E 0E 0E
E0 E0 E0 E0 F1 F1 F1 F1
FE FE FE FE FE FE FE FE

Imagen 4: Claves débiles de DES.

01FE01FE01FE01FE
FE01FE01FE01FE01
1FE01FE00EF10EF1
E01FE01FF10EF10E
01E001E001F101F1
E001E001F101F101
1FFE1FFE0EFE0EFE
FE1FFE1FFE0EFE0E
011F011F010E010E
1F011F010E010E01
E0FEE0FEF1FEF1FE
FEE0FEE0FEF1FEF1

Imagen 5: Claves semidébiles de DES.

Así, con todo lo explicado hasta ahora es el momento de utilizar los comandos que aporta OpenSSL para realizar el cifrado. Para ello voy a utilizar la sintaxis descrita en el manual de OpenSSL en [2].

El comando utilizado para cifrar con DES en modo ECB es:

```
$$ openssl enc -des-ecb -in input.bin -out des-ecb-debil-input.bin -K FEFEFEFEFEFEFEFE
```

El funcionamiento es el siguiente:
enc: te permite utilizar las rutinas de cifrado simétrico para cifrar y descifrar ficheros.
-des-ecb: es el método de cifrado escogido.
-in: indica el fichero que se desea cifrar.

-out: indica cual será el fichero de salida.
-K: la clave utilizada para el cifrado.

El resultado de este fichero ha resultado ser un fichero con 136 bytes (ver Imagen 6).

```
manolo@manolo-Parallels:~/Documentos/SPSI$ openssl enc -des-ecb -in input.bin -o
ut des-ecb-debil-input.bin -K FEFEFEFEFEFEFEFE
manolo@manolo-Parallels:~/Documentos/SPSI$ ls -l
total 484
-rw-rw-r-- 1 manolo manolo    136 oct 15 19:13 des-ecb-debil-input.bin
-rw-rw-r-- 1 manolo manolo     85 sep 29 09:25 enlaces.txt
-rw-rw-r-- 1 manolo manolo    128 sep 29 09:22 input1.bin
-rw-rw-r-- 1 manolo manolo    128 sep 29 09:19 input.bin
-rw-rw-r-- 1 manolo manolo 473956 oct 14 19:36 Practica1.odt
```

Imagen 6: Tamaño del fichero generado con clave débil cifrando con DES en modo ECB.

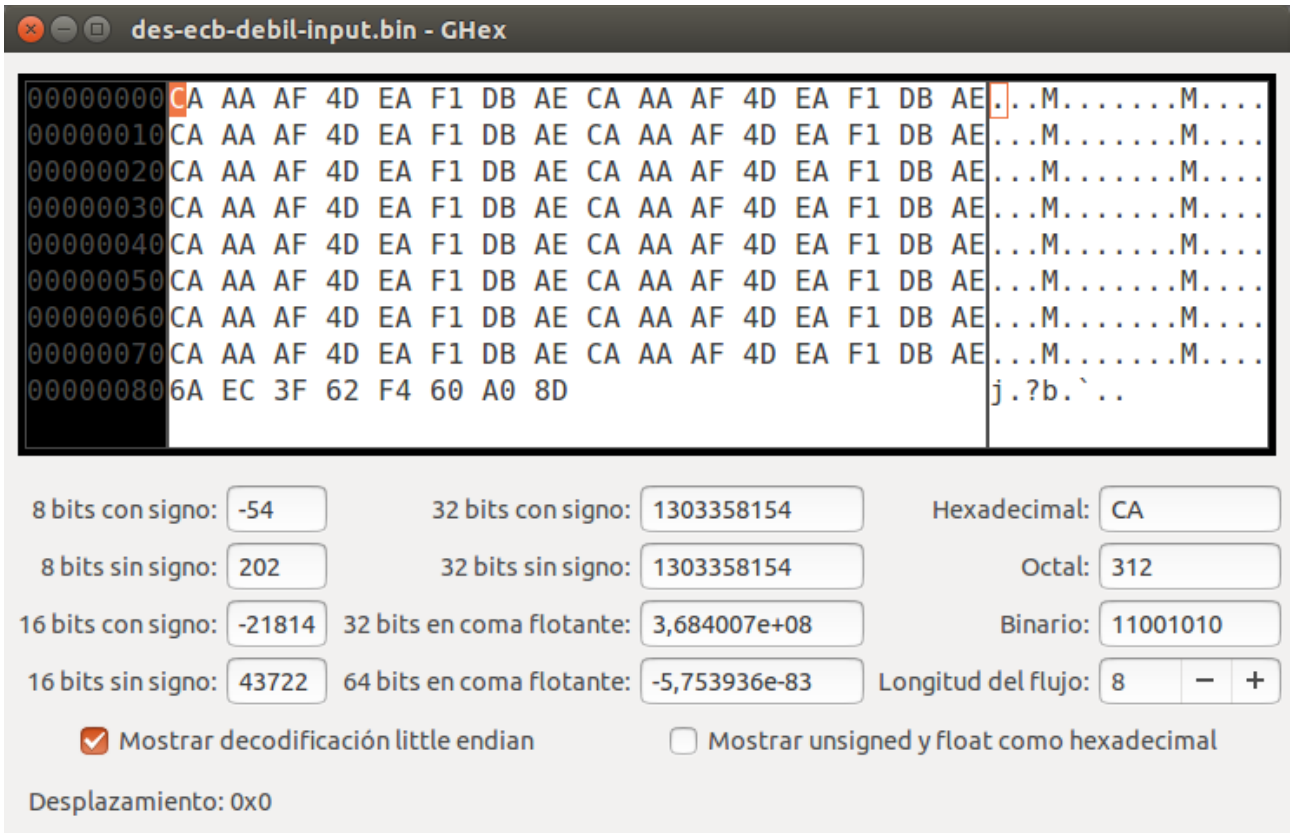


Imagen 7: Contenido del fichero generado con clave débil con DES en modo ECB.

Esto es debido a que se ha añadido un bloque de padding al fichero (ver Imagen 7), por eso hay justo 64 bits más de lo que debiera. Si quiero que ese bloque no aparezca hay que añadir además la opción -nopad.

```
$$ openssl enc -des-ecb -in input.bin -out des-ecb-claveDebil.bin -K FEFEFEFEFEFEFEFE -nopad
```

Entonces el resultado es un fichero de 128 bytes (ver Imagen 8).

```
manolo@manolo-Parallels:~/Documentos/SPSI$ openssl enc -des-ecb -in input.bin -o
ut des-ecb-debil-input-nopad.bin -K FEFEFEFEFEFEFEFE -nopad
manolo@manolo-Parallels:~/Documentos/SPSI$ ls -l
total 488
-rw-rw-r-- 1 manolo manolo    136 oct 15 19:20 des-ecb-debil-input.bin
-rw-rw-r-- 1 manolo manolo    128 oct 15 19:20 des-ecb-debil-input-nopad.bin
-rw-rw-r-- 1 manolo manolo     85 sep 29 09:25 enlaces.txt
-rw-rw-r-- 1 manolo manolo    128 sep 29 09:22 input1.bin
-rw-rw-r-- 1 manolo manolo    128 sep 29 09:19 input.bin
-rw-rw-r-- 1 manolo manolo 473956 oct 14 19:36 Practica1.odt
```

Imagen 8: Tamaño del fichero generado con clave débil cifrando con DES en modo ECB sin padding.

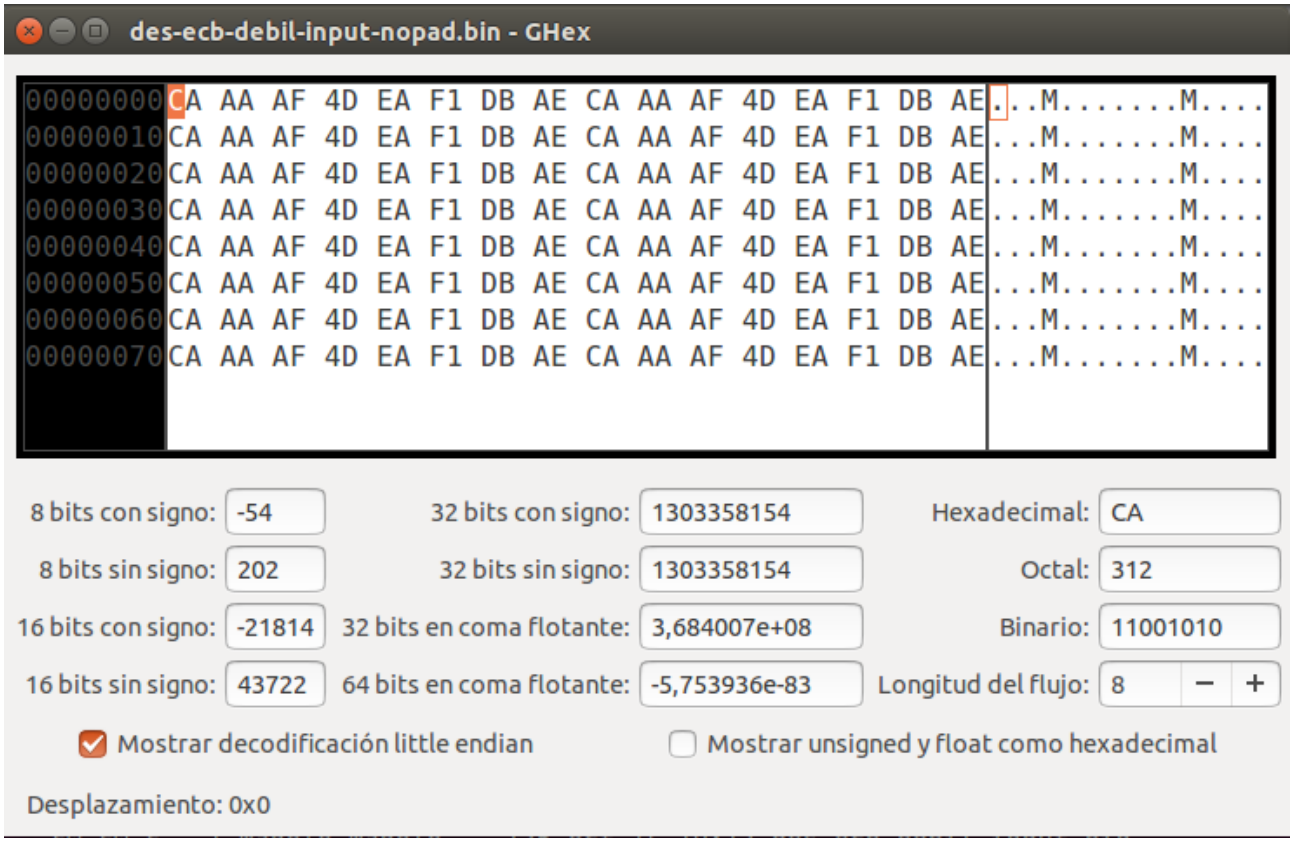


Imagen 9: Contenido del fichero generado con clave débil con DES en modo ECB sin padding.

Este fichero (ver Imagen 9) es el que corresponde justamente con el cifrado del fichero de entrada, por lo que para los siguientes métodos voy a utilizar siempre la opción -nopad para evitar posibles confusiones.

Ahora hago lo correspondiente con los otros métodos de cifrado en bloque, aunque manteniendo la misma clave para poder comparar luego los tres ficheros de salida.

El comando en este caso es:

```
$$ openssl enc -des-cbc -in input.bin -out des-cbc-debil-input.bin -K FEFEFEFEFEFEFEFE -nopad
-iv 1234567890ABCDEF
```

La opción -iv indica cual será el vector de inicialización usado en los métodos CBC y OFB.

El resultado obtenido para el modo CBC es el de la Imagen 10. En la Imagen 11 se puede ver el resultado obtenido para el modo OFB obtenido con la siguiente orden:

```
$$ openssl enc -des-ofb -in input.bin -out des-ofb-debil-input.bin -K FEFEFEFEFEFEFEFE -nopad -iv 1234567890ABCDEF
```

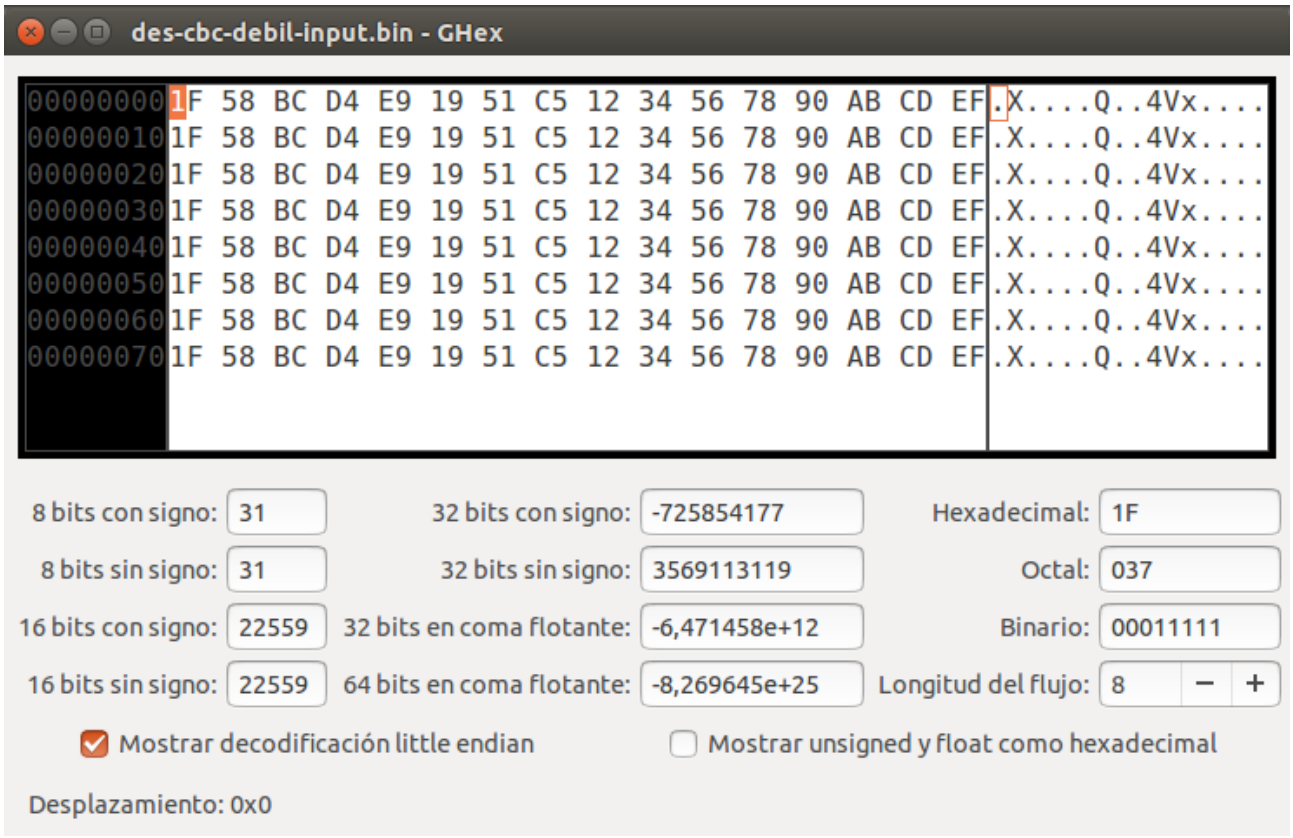


Imagen 10: Cifrado DES en modo CBC con clave débil sobre el fichero input.bin.

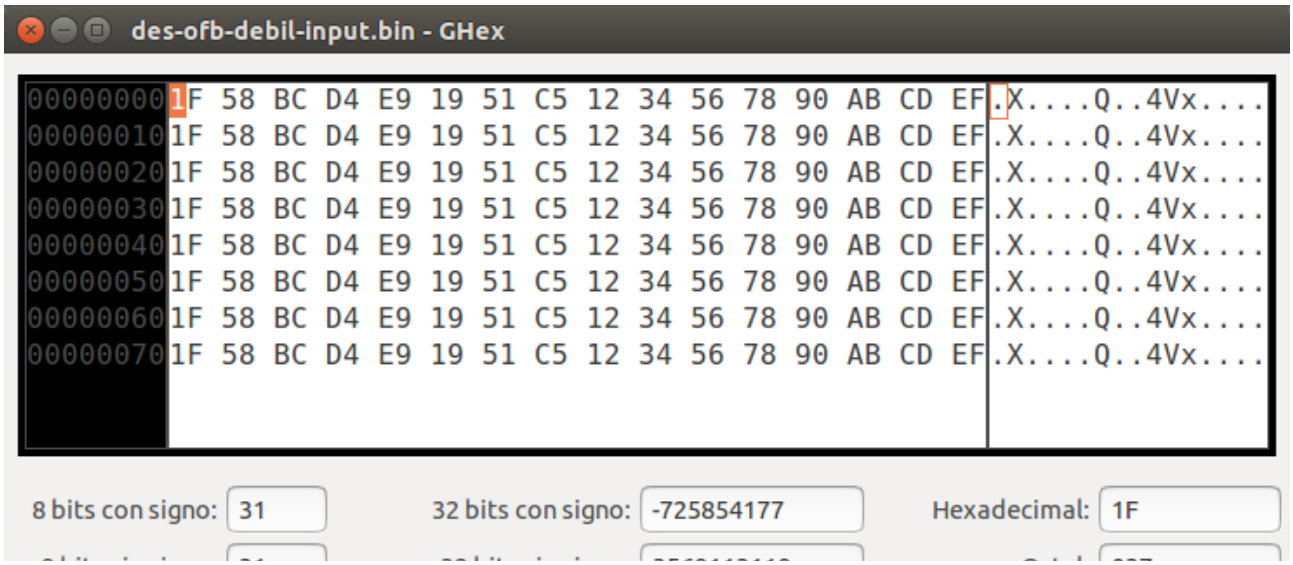


Imagen 11: Cifrado DES en modo OFB con clave débil sobre el fichero input.bin.

Como primer resumen de estos 3 modos de cifrado en bloque para el criptosistema DES utilizando una clave débil se pueden extraer varias conclusiones.

La primera es que si mi fichero está formado por 1024 bits todos a cero, se subdivide el fichero en bloques de 64 bits todos a cero también y cada uno de esos bloques es la entrada para el cifrado ECB, entonces cada uno de los bloques cifrados será exactamente idéntico a los demás. Esto es debido al funcionamiento del cifrado ECB, el cual coge un bloque del mensaje, lo cifra usando el algoritmo especificado y así genera el mensaje cifrado. Como en este caso, el algoritmo usado es siempre DES y siempre con la misma clave, dado que los bloques de entrada son idénticos se tiene en consecuencia que los bloques de salida sean idénticos (CA AA AF 4D EA F1 DB AE).

La segunda está relacionada con la salida del cifrado CBC. Como se puede observar, los bloques impares son todos idénticos entre sí y además los bloques pares son todos idénticos al vector de inicialización (IV). Esto se debe de nuevo al funcionamiento del CBC. Voy a explicar en un par de pasos el funcionamiento y luego simplemente habrá que generalizar e iterar para comprender la salida. Sabiendo que cada mensaje (m_i) es un bloque de 64 bits todos cero y que cada mensaje cifrado se obtiene como $c_i = e(m_i + c_{i-1})$, donde e es el algoritmo de cifrado, en este caso DES con clave débil, c_0 = IV y el símbolo + representa la suma bit a bit. Entonces, como primer bloque cifrado obtengo $c_1 = e(m_1 + c_0) = e(c_0) = e(IV)$, es decir, el vector de inicialización cifrado (1F 58 BC D4 E9 19 51 C5). La suma (que es un XOR) del primer bloque (todo de ceros) con el vector de inicialización da como resultado el propio vector de inicialización. Ahora intento obtener el segundo bloque cifrado. De nuevo, aplicando la fórmula anterior obtengo que $c_2 = e(m_2 + c_1) = e(c_1) = e(e(IV)) = IV$. Este bloque es el resultado de cifrar la suma de un bloque entero de ceros con un bloque que representa el cifrado del vector de inicialización. Obviamente la suma da como resultado el bloque cifrado anterior. Si ahora se intenta cifrar el bloque cifrado, debido a que la clave utilizada por DES es débil y por tanto sus subclaves de ronda todas iguales, las operaciones de cifrado y descifrado son iguales ya que da igual aplicarlas en orden ascendente (1 a 16) o descendente (16 a 1). Por tanto, e(e(m)) = m, por lo que se produce justamente el descifrado del mensaje, que en este caso es el vector de inicialización (12 34 56 78 90 AB CD EF). Por lo tanto, para el siguiente bloque estaría sumando de nuevo un bloque entero de ceros con el vector de inicialización y por eso se repiten los bloques.

Por último, el cifrado OFB. Lo que ocurre en este caso es muy similar al caso anterior. La fórmula en este caso es $c_i = m_i + e(s_i)$, donde e es el algoritmo de cifrado, $s_i = e(s_{i-1})$ y s_0 = IV. Así, s_0 = IV, $s_1 = e(IV)$ y por tanto $c_1 = m_1 + e(IV) = e(IV)$. Como se puede observar en la Imagen 11, el primer bloque es igual al primer bloque de CBC en la Imagen 10 ya que e(IV) en ambos casos corresponde a un cifrado con DES con la misma clave y el mismo vector de inicialización. El segundo paso se obtiene igual, $s_2 = e(s_1) = e(e(IV)) = IV$, $c_2 = m_2 + IV = IV$. Por eso, igual que en el caso anterior los bloques pares coinciden todos con el IV. Otra vez hay que indicar que esto ocurre debido a que las subclaves de ronda son todas iguales ya que la clave es débil.

Entendiendo el funcionamiento de estos tres modos de cifrado en bloques será muy sencillo comprender sus resultados en los ejemplos futuros.

Ahora voy a repetir lo realizado anteriormente pero cambiando la clave débil por una semidébil y sin modificar el IV. Las Imágenes 12, 13 y 14 muestran el contenido de los ficheros tras aplicar DES en modos ECB, CBC y OFB respectivamente con la misma clave semidébil.

```
$$ openssl enc -des-ecb -in input.bin -out des-ecb-semidebil-input.bin -K FEE0FEE0FEF1FEF1
```

-nopad

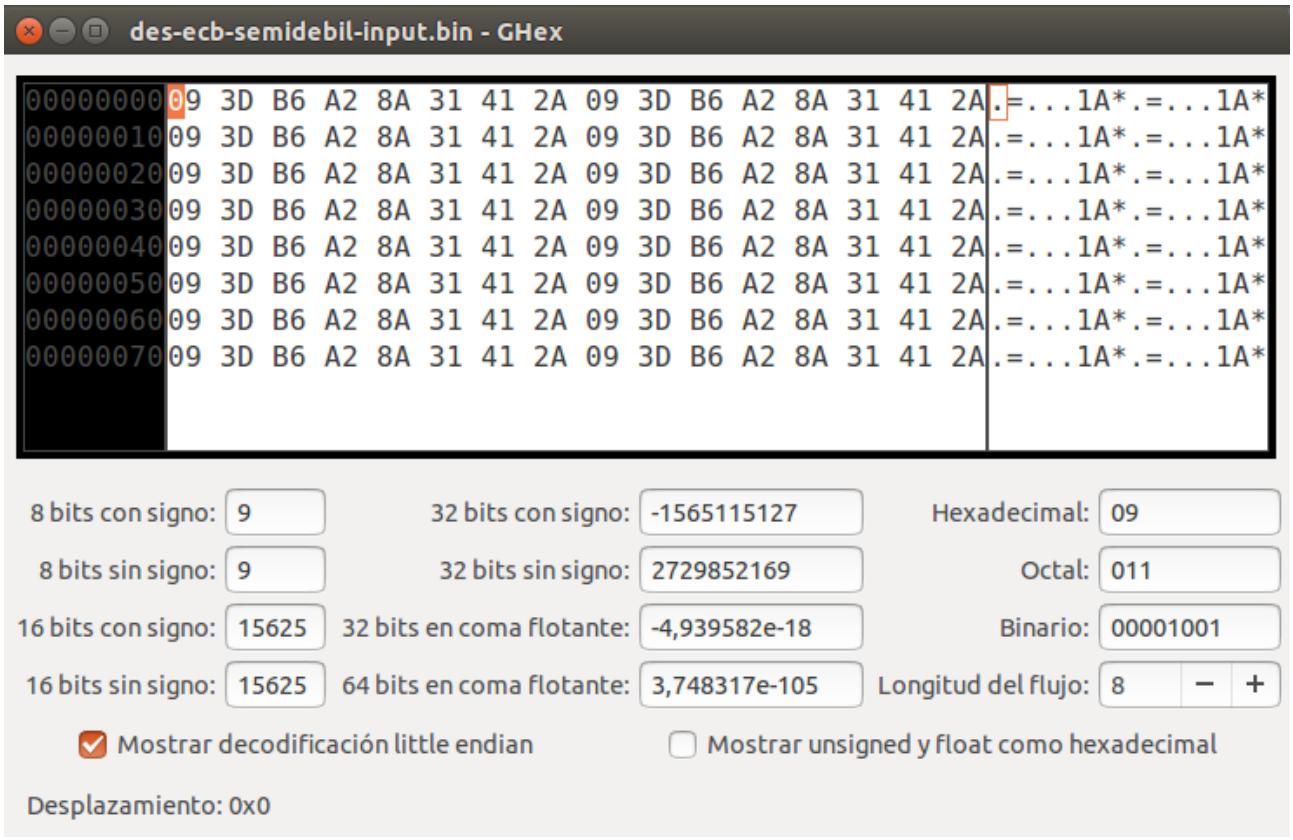


Imagen 12: Cifrado DES en modo ECB con clave semidébil sobre el fichero input.bin.

$$ openssl enc -des-cbc -in input.bin -out des-cbc-semidebil-input.bin -K FEE0FEE0FEF1FEF1 -nopad -iv 1234567890ABCDEF

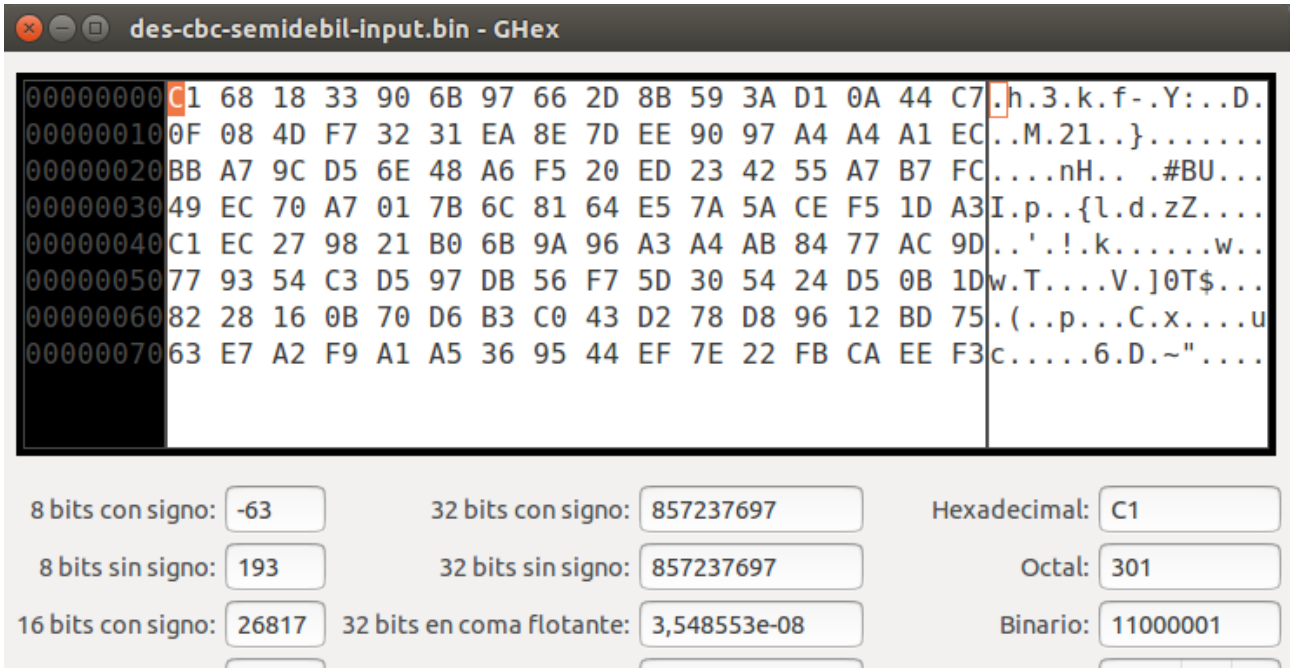


Imagen 13: Cifrado DES en modo CBC con clave semidébil sobre el fichero input.bin.

```
$$ openssl enc -des-ofb -in input.bin -out des-ofb-semidebil-input.bin -K FEE0FEE0FEF1FEF1
-nopad -iv 1234567890ABCDEF
```

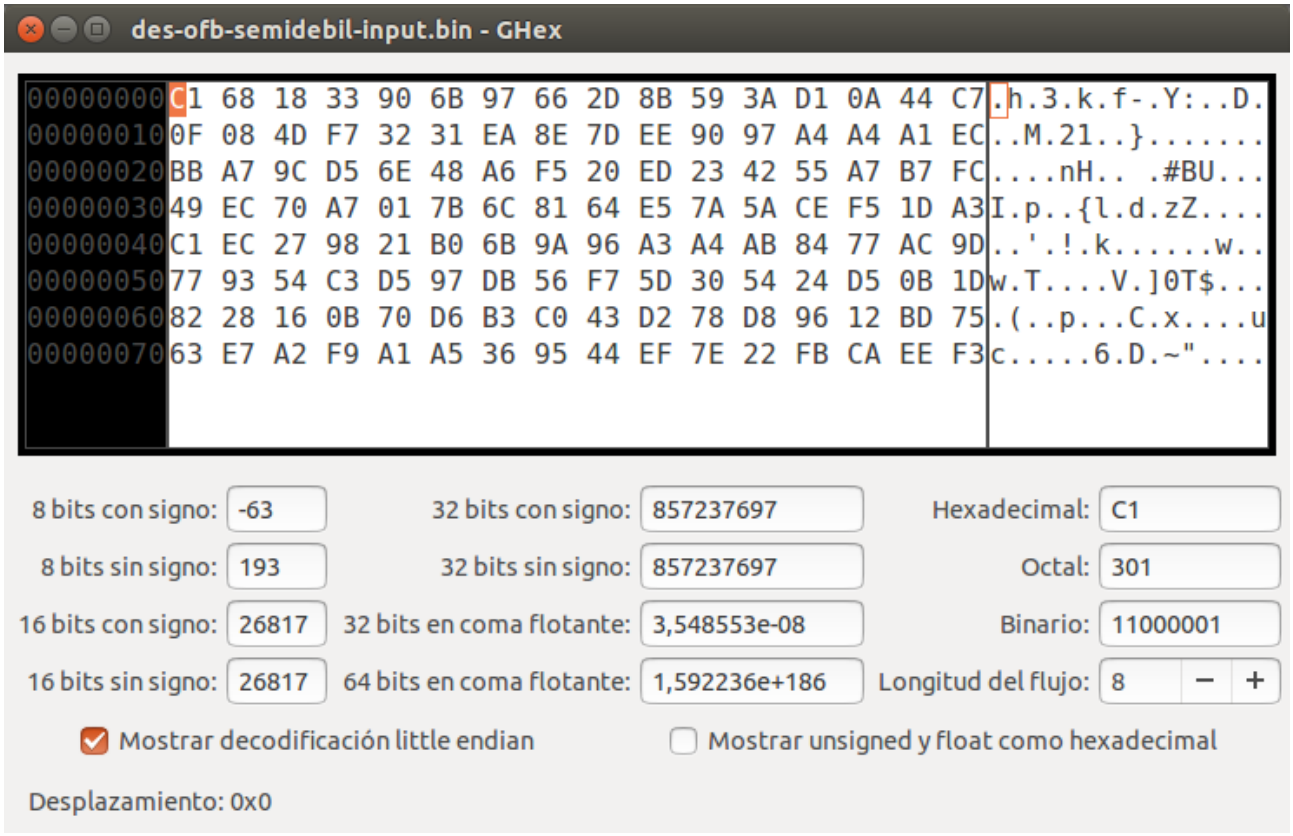


Imagen 14: Cifrado DES en modo OFB con clave semidébil sobre el fichero input.bin.

Como ya es conocido el funcionamiento de los 3 métodos de cifrado la explicación va a ser más rápida. En el caso del ECB se observa que de nuevo se repiten los bloques cada 64 bits. Esto es por su funcionamiento ya que cifra independientemente cada bloque y, por tanto, ante bloques iguales salidas iguales. Con respecto al uso de clave semidébil frente a la débil, dado que la clave ha cambiado, el bloque generado ha cambiado.

Para el caso del CBC hay que fijarse que ahora aparentemente no hay ningún bloque repetido. Esto es debido al uso de la clave semidébil. Ahora esta clave genera dos (o cuatro) subclaves de ronda distintas que se repiten alternándose en las distintas rondas de DES. Por tanto, en la fórmula que se veía en la explicación anterior, cuando se daba la iteración de cifrado sobre cifrado, e(e(IV)), en este caso no se descifra ya que para descifrar ahora sí es importante el orden en el que se han usado las claves de ronda y habría que usarlo justamente en el orden inverso. Ahora la aplicación del algoritmo en cada paso hace que se cifre un poco más el resultado anterior y por eso cada bloque es distinto.

En el caso del OFB se obtiene lo mismo que en el CBC ya que el primer bloque que se generaba era el mismo que para ese, e(IV), y como las subclaves de ronda se mantienen siempre en el mismo orden el algoritmo da como resultado el mismo bloque cifrado.

Ejercicio 4

Cifrad input.bin e input1.bin con DES en modo ECB y clave a elegir, pero no débil ni semidébil. Explicad la forma de los resultados obtenidos.

Para realizar este ejercicio se van a utilizar prácticamente las mismas órdenes usadas en el ejercicio anterior, solo que esta vez cambiando la clave por una que no sea débil ni semidébil (voy a llamarla fuerte).

```
$$ openssl enc -des-ecb -in input.bin -out des-ecb-fuerte-input.bin -K 1A2B3C4D5E6F7890 -nopad
```

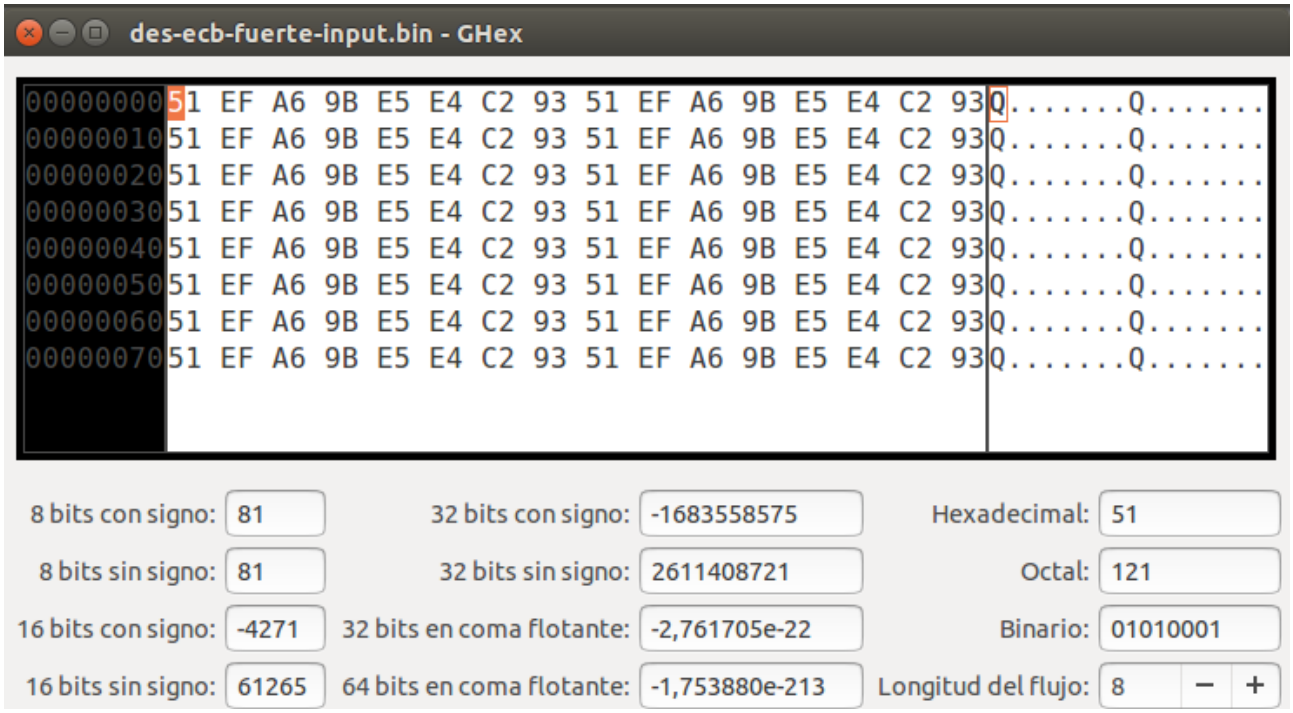


Imagen 15: Cifrado DES en modo ECB con clave fuerte sobre el fichero input.bin.

```
$$ openssl enc -des-ecb -in input1.bin -out des-ecb-fuerte-input1.bin -K 1A2B3C4D5E6F7890
-nopad
```

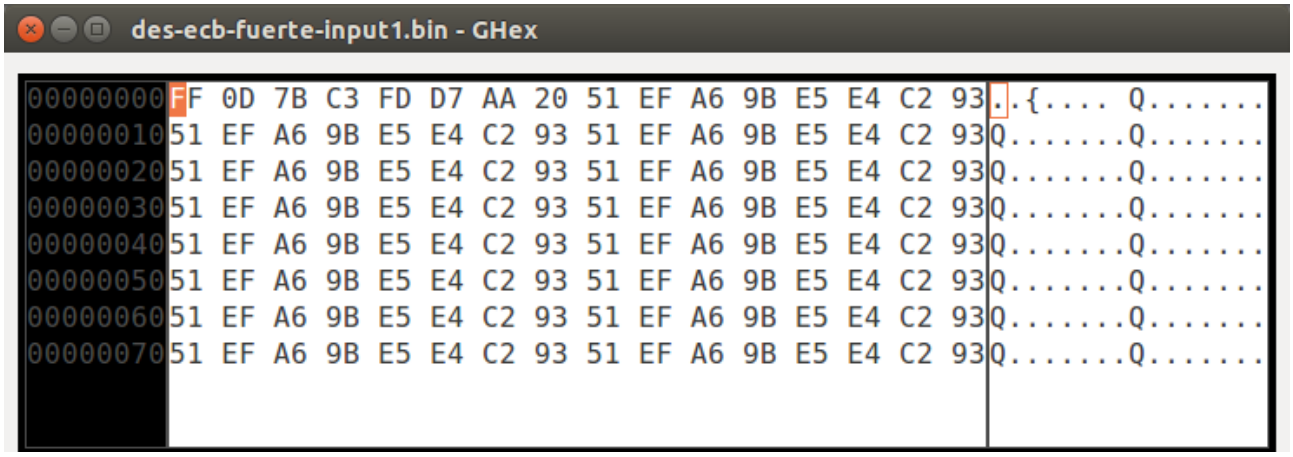


Imagen 16: Cifrado DES en modo ECB con clave fuerte sobre el fichero input1.bin.

De nuevo, una vez más se puede ver (en la Imagen 15) que el simple funcionamiento del modo ECB hace que, independientemente de que la clave elegida sea fuerte, los bloques de 64 bits se repiten de nuevo. En cambio, se puede apreciar en la Imagen 16 que esto no ocurre igual en el cifrado del fichero input1.bin. En éste, se obtiene un primer bloque completamente distinto y los demás ya si vuelven a su estructura de repetición, incluso con los del fichero cifrado de input.bin. El hecho de que se repitan los bloques tanto en uno como en otro es, otra vez, por el funcionamiento del ECB que cifra por bloques independientemente de los demás. El cambio en el primer bloque viene dado porque en el texto plano del fichero input1.bin hay un bit que tiene su valor a 1 entre los primeros 64 bits (lo que correspondería al primer bloque). Dado que el mensaje de este bloque es distinto del mensaje formado por un bloque completamente de ceros, el resultado obtenido tras cifrarlo es también distinto. Pero, ¿por qué afecta este cambio solo al primer bloque? Esto es claro, es porque el tratamiento de los bloques en ECB son independientes del resto. Esto hace referencia justamente a las propiedades de difusión y confusión de un criptosistema. Como decía C. E. Shannon, “un criptosistema debía tener esas dos propiedades para no ser vulnerable a ataques estadísticos y de frecuencias”. Pero esas dos propiedades debían ser globales al criptosistema. En cambio, en el modo ECB, la difusión y la confusión son propiedades locales en cada bloque. Esto quiere decir que un pequeño cambio en un bloque hará que se altere el cifrado de ese bloque, pero solamente el de ése. Un buen método de cifrado debería hacer que todos los bloques se vieran afectados por ese cambio.

Ejercicio 5

Cifrad input.bin e input1.bin con DES en modo CBC, clave y vector de inicialización a elegir. Comparad con los resultados obtenidos en el apartado anterior.

De nuevo voy a ejecutar el mismo comando otro par de veces para ver qué resultados se obtienen.

```
$$ openssl enc -des-cbc -in input.bin -out des-cbc-fuerte-input.bin -K 1A2B3C4D5E6F7890 -nopad
-iv 1234567890ABCDEF
```

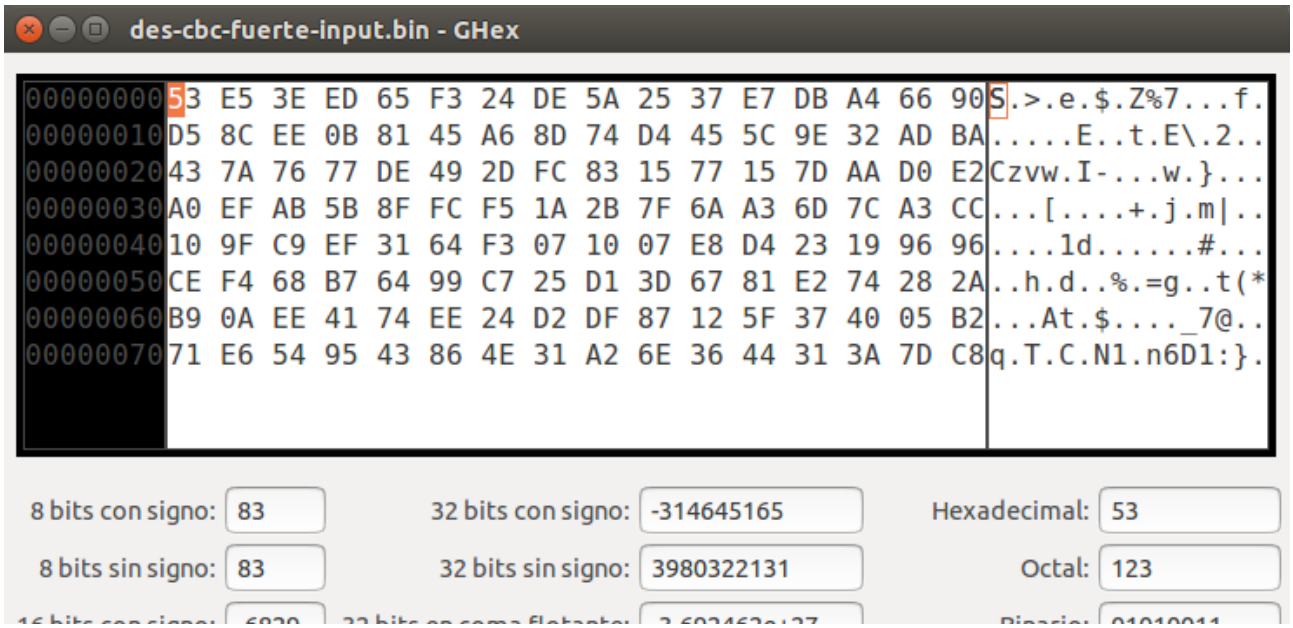


Imagen 17: Cifrado DES en modo CBC con clave fuerte sobre el fichero input.bin.

```
$$ openssl enc -des-cbc -in input1.bin -out des-cbc-fuerte-input1.bin -K 1A2B3C4D5E6F7890 -nopad -iv 1234567890ABCDEF
```

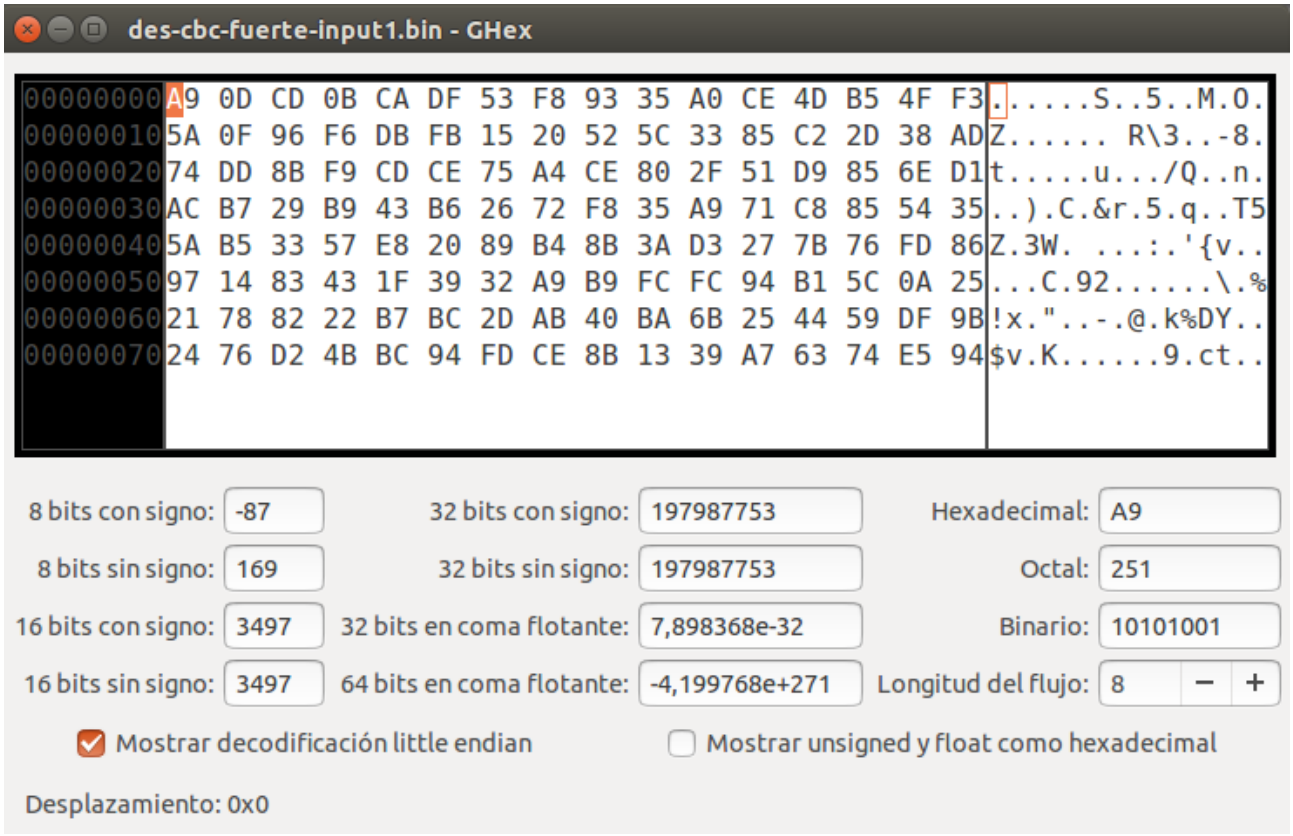


Imagen 18: Cifrado DES en modo CBC con clave fuerte sobre el fichero input1.bin.

En este caso vemos que el CBC es un método de cifrado mucho más seguro que los anteriores. El uso de una clave fuerte hace que sus subclaves de ronda sean todas distintas y por tanto los sucesivos cifrados en cadena que se realizan en sus distintas rondas generen bloques completamente distintos (Imagen 17). Además, las propiedades de difusión y confusión en este caso son globales. Como se puede ver en la Imagen 18, el pequeño cambio introducido en el primer bloque se ve encadenado a lo largo del cifrado del archivo entero generando así un fichero cifrado completamente distinto del anterior.

Ejercicio 6

Repetid los puntos 4 a 5 con AES-128 y AES-256.

Ahora voy a repetir los ejercicios anteriores pero cambiando el algoritmo de encriptación. En primer lugar voy a usar el AES-128. Para ello hay que saber que AES-128 hace referencia al tamaño de la clave, que debe ser de 128 bits. Además, hay que saber también que AES siempre utiliza bloques de tamaño 128 bits, por tanto, el vector de inicialización en los casos en que sea necesario también tiene que tener tamaño 128 bits (igual que los del bloque).

```
$$ openssl enc -aes-128-ecb -in input.bin -out aes-128-ecb-fuerte-input.bin -K 1A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad
```

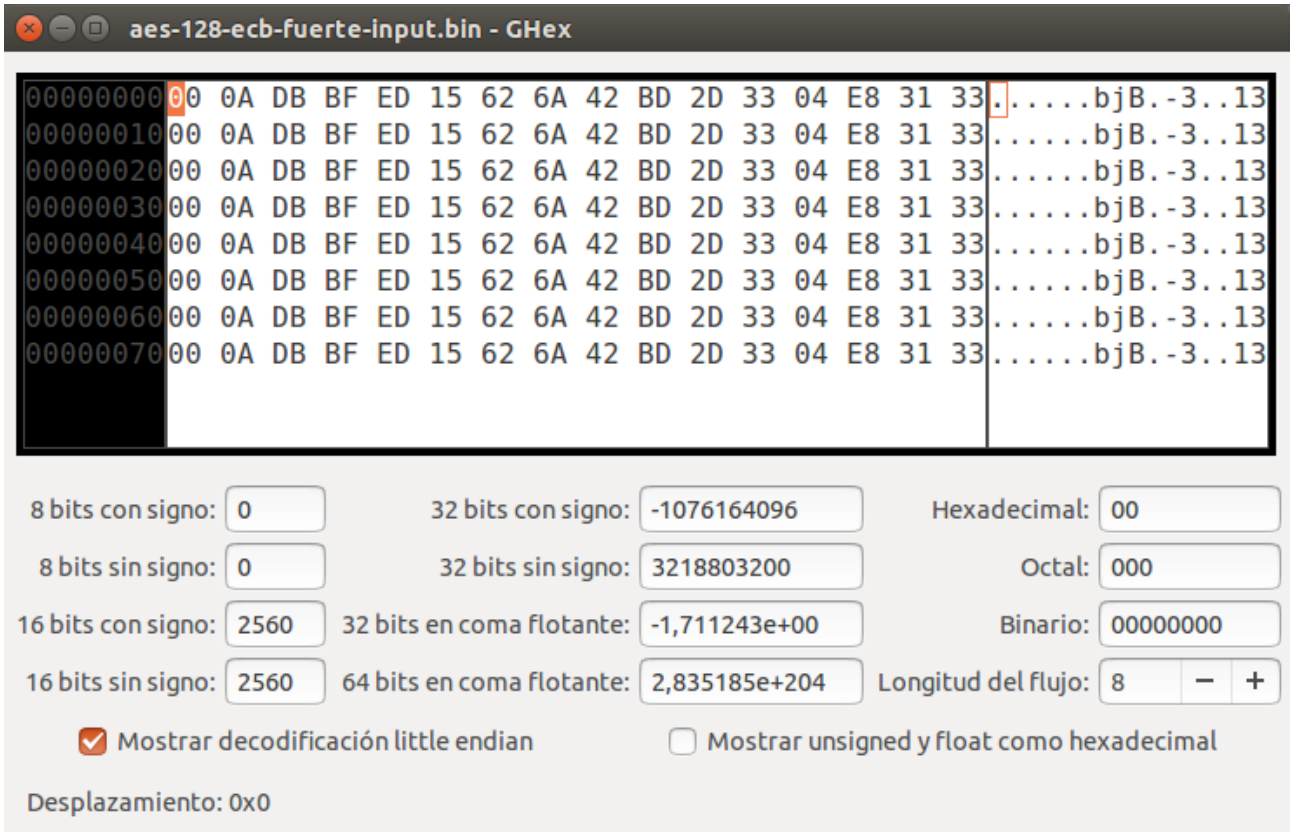


Imagen 19: Cifrado AES-128 en modo ECB con clave fuerte sobre el fichero input.bin.

```
$$ openssl enc -aes-128-ecb -in input1.bin -out aes-128-ecb-fuerte-input1.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad
```

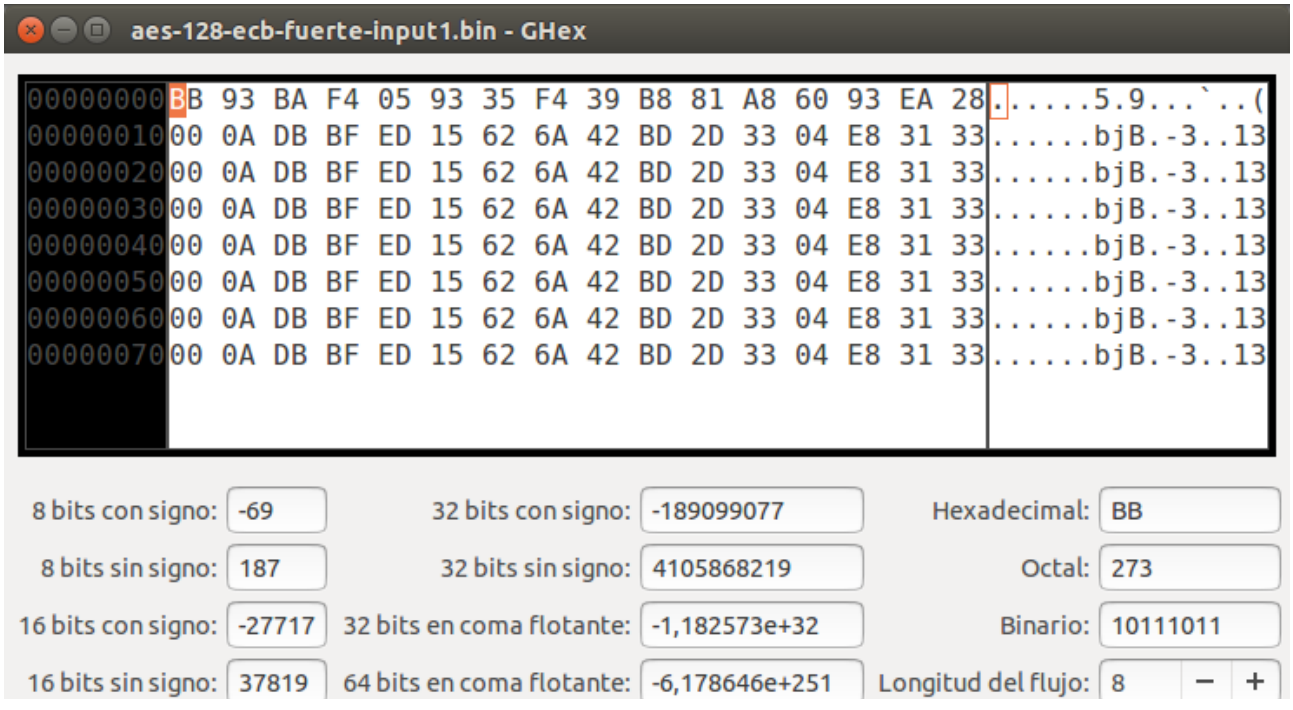


Imagen 20: Cifrado AES-128 en modo ECB con clave fuerte sobre el fichero input1.bin.

```
$$ openssl enc -aes-128-cbc -in input.bin -out aes-128-cbc-fuerte-input.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad -iv
1234567890ABCDEF1234567890ABCDEF
```

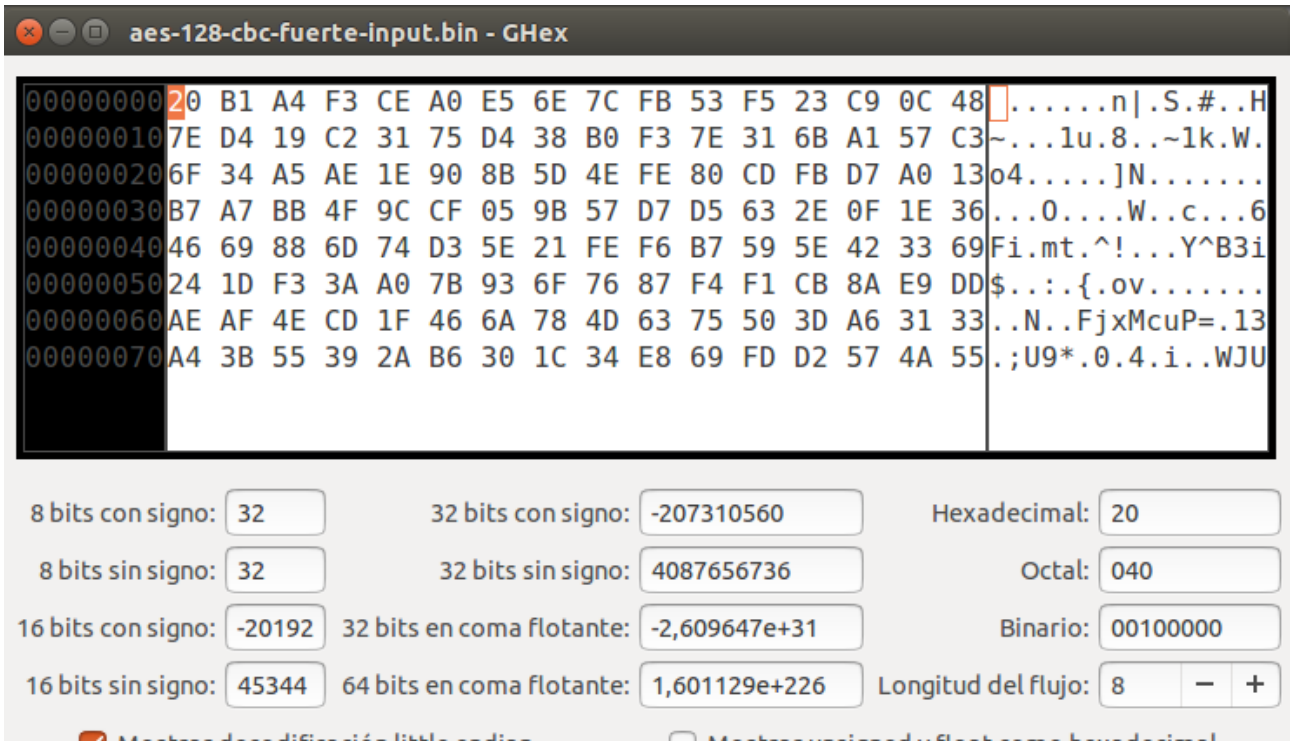


Imagen 21: Cifrado AES-128 en modo CBC con clave fuerte sobre el fichero input.bin.

```
$$ openssl enc -aes-128-cbc -in input1.bin -out aes-128-cbc-fuerte-input1.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad -iv
1234567890ABCDEF1234567890ABCDEF
```

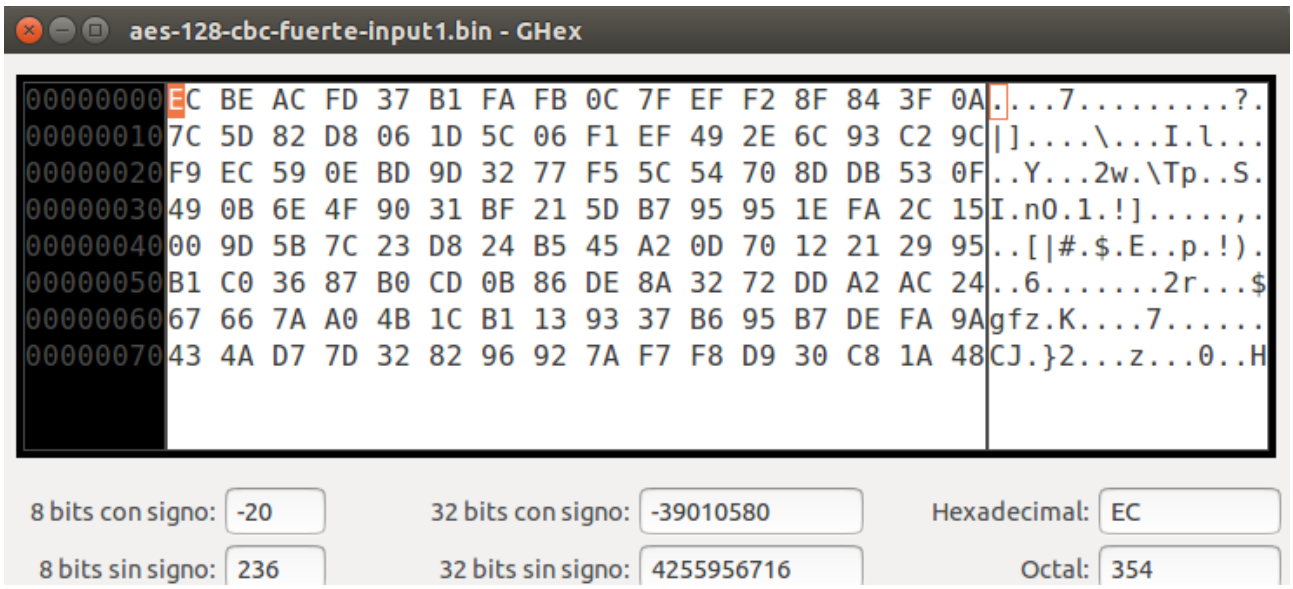


Imagen 22: Cifrado AES-128 en modo CBC con clave fuerte sobre el fichero input1.bin.

En este ejercicio, se puede observar que el funcionamiento de AES-128 en modo ECB es exactamente el mismo que para el DES, con la única salvedad de que en este caso los bloques son de 128 bits en lugar de 64 bits, y por tanto los bloques se repiten de 128 en 128 bits (Imagen 19). Otra vez, un pequeño cambio introducido en el primer bloque afectará solo al primer bloque por la misma razón que en el ejercicio anterior (Imagen 20).

En el caso de CBC con clave fuerte, DES ya funcionaba bastante bien. Lo único que cambia es el método de cifrado de los bloques además de su tamaño. Los bloques en AES se cifran utilizando un algoritmo basado en rondas con operaciones que, en general, no son conmutativas y que se deben realizar en un orden determinado. El cifrado de un bloque da como resultado un bloque cifrado, al que si se le vuelve a aplicar de nuevo otro cifrado para la siguiente ronda se obtendrá otro bloque cifrado. En este caso no ocurre como con las claves débiles, que si cifrabas dos veces obtenías el descifrado. Para eso habría que realizar las operaciones inversas y además en el orden inverso, cosa que no sucede aquí. Por tanto, CBC irá cifrando encadenadamente los distintos bloques, cada uno en función del anterior. Por tanto, todos los bloques obtenidos en el modo CBC son distintos (Imagen 21). Igualmente, si se introduce un pequeño cambio (Imagen 22), como éste es arrastrado por toda la cadena de cifrados, cada bloque generado será completamente distinto del correspondiente bloque en el fichero original cifrado.

A continuación realizo de nuevo otro ejercicio similar, esta vez con el algoritmo AES-256. En este caso el tamaño de los bloques no cambian. Únicamente cambia el tamaño de las claves, pero esto en principio no afecta más que para añadir dificultad a la hora de descifrar la relación existente entre un bloque sin cifrar y su correspondiente cifrado.

```
$$ openssl enc -aes-256-ecb -in input.bin -out aes-256-ecb-fuerte-input.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad
```

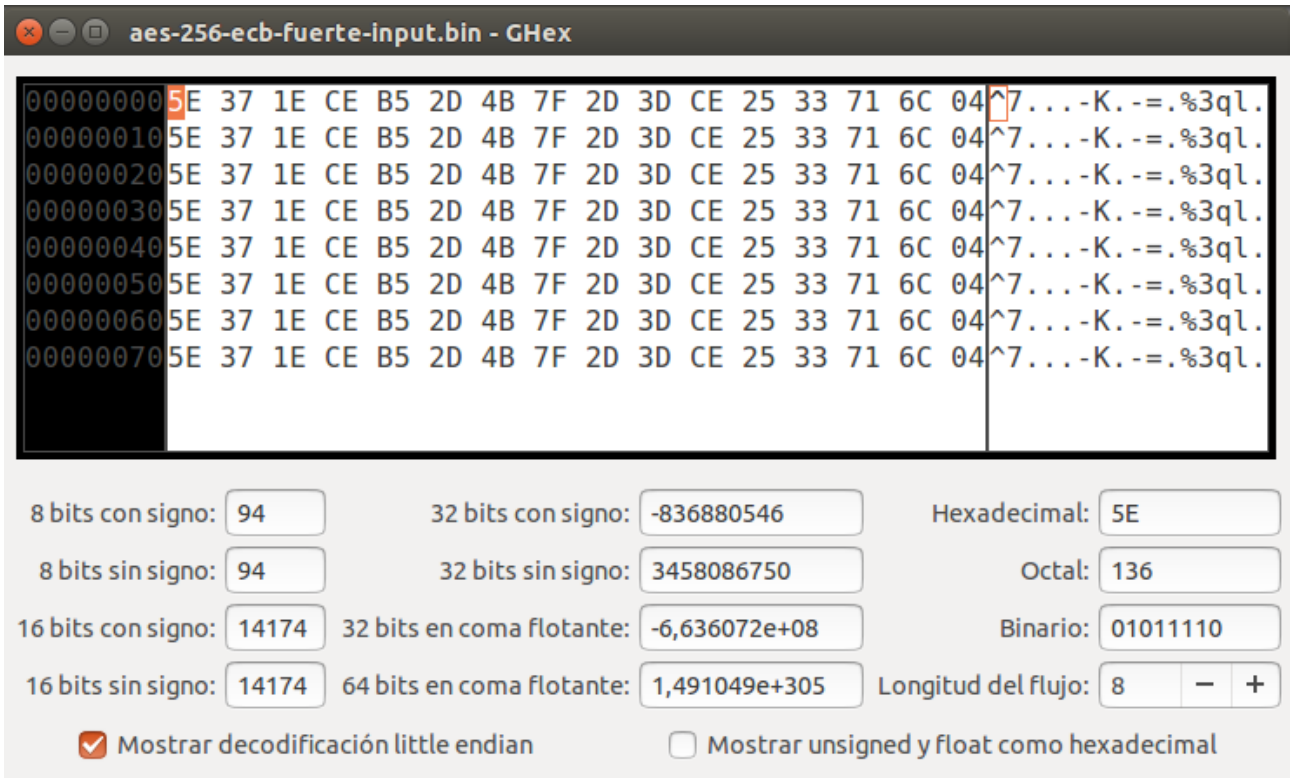


Imagen 23: Cifrado AES-256 en modo ECB con clave fuerte sobre el fichero input.bin.

```
$$ openssl enc -aes-256-ecb -in input1.bin -out aes-256-ecb-fuerte-input1.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad
```

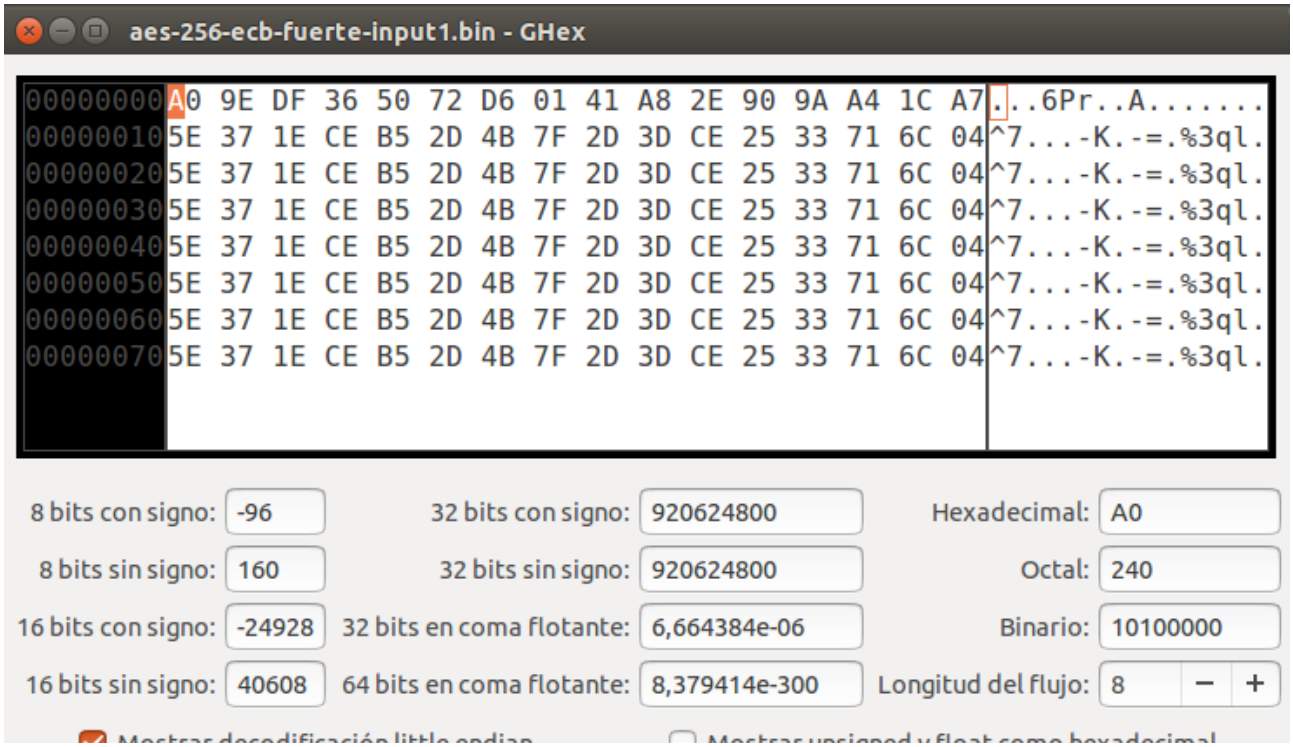


Imagen 24: Cifrado AES-256 en modo ECB con clave fuerte sobre el fichero input1.bin.

```
$$ openssl enc -aes-256-cbc -in input.bin -out aes-256-cbc-fuerte-input.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad
-iv 1234567890ABCDEF1234567890ABCDEF
```

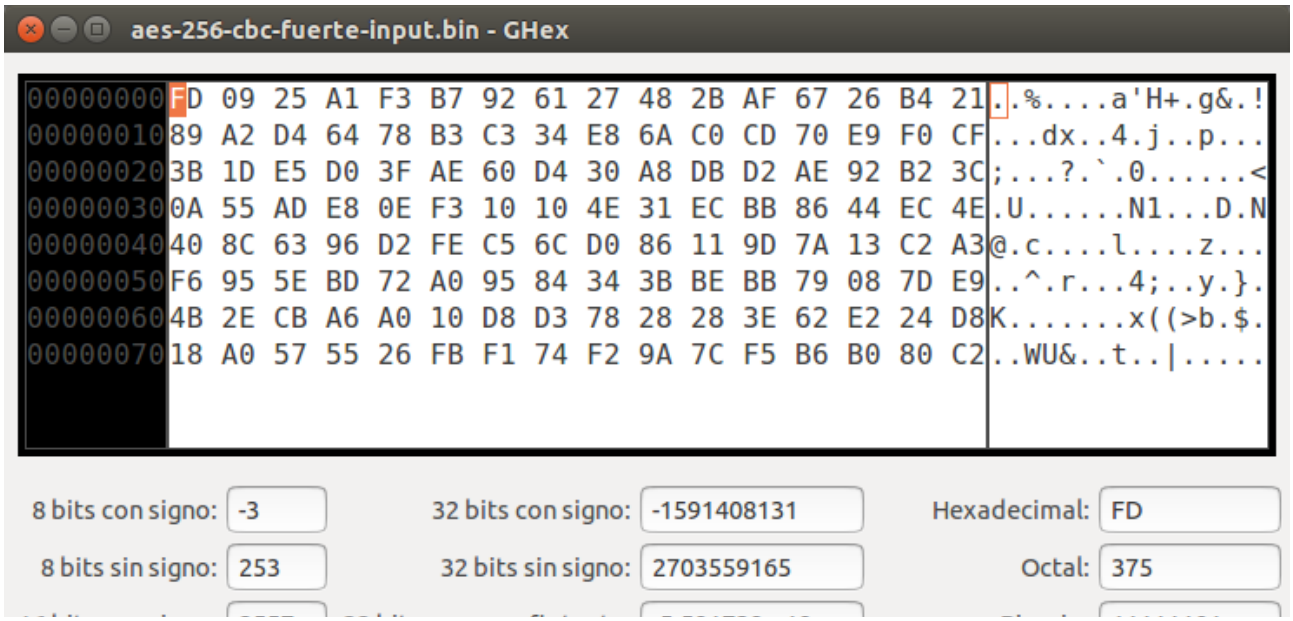


Imagen 25: Cifrado AES-256 en modo CBC con clave fuerte sobre el fichero input.bin.

```
$$ openssl enc -aes-256-cbc -in input1.bin -out aes-256-cbc-fuerte-input1.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad
-iv 1234567890ABCDEF1234567890ABCDEF
```

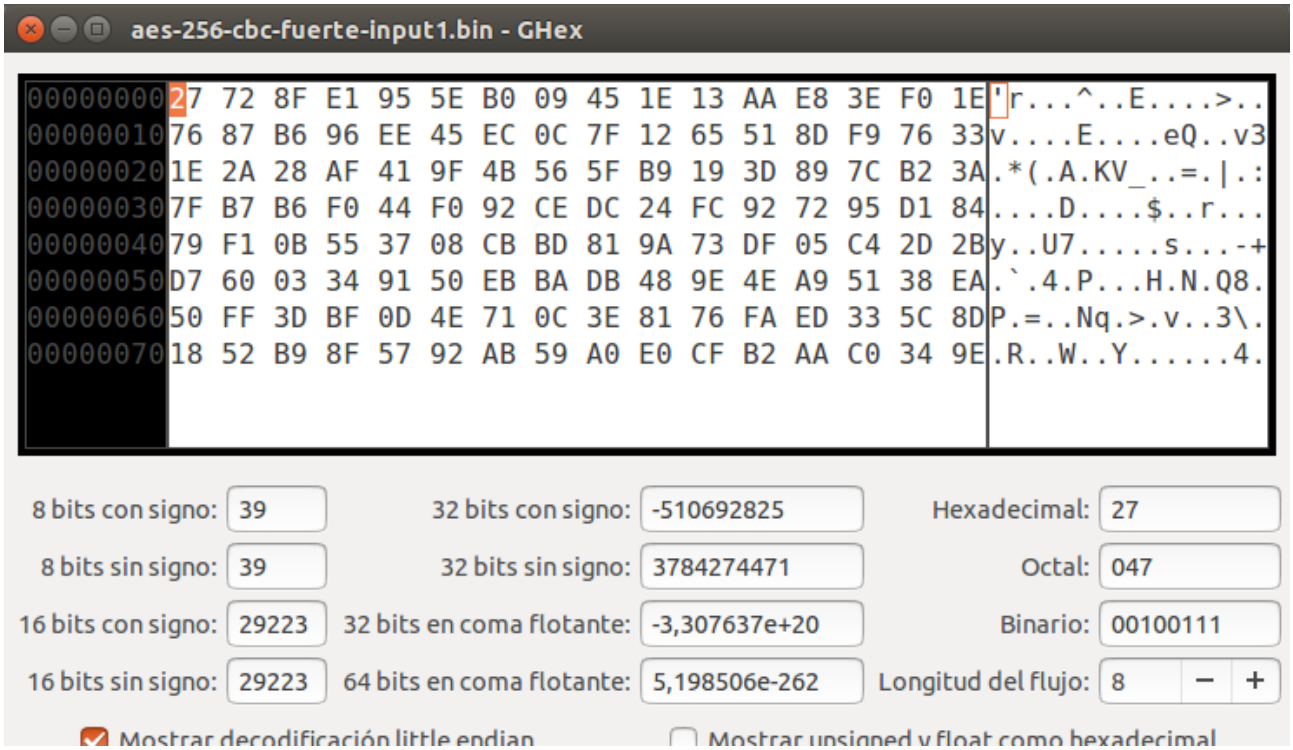


Imagen 26: Cifrado AES-256 en modo CBC con clave fuerte sobre el fichero input1.bin.

Las conclusiones que se pueden extraer para este caso son exactamente las mismas que para el caso de AES-128. Una mente poco aguda podría preguntarse, por ejemplo en el caso de ECB: si antes el cambio introducido afectaba al primer bloque de 128 bits, ¿por qué ahora no afecta al primero de 256 bits? Esto es debido a que, como he dicho anteriormente, AES-256 indica el tamaño de la clave, no el del bloque que se sigue manteniendo igual. Es decir, los bloques aquí también son de 128 bits, por eso cuando se producen las repeticiones son en bloques de 128 bits (Imagen 23) y el cambio afecta solo a un primer bloque de ese tamaño (Imagen 24). Cabe destacar que el uso de AES-256 y de AES-128 para CBC genera dos ficheros distintos, ya que el tamaño de la clave hace que las claves sean distintas y, por tanto, los cifrados de los bloques sean distintos (ver Imágenes 25 y 26).

Ejercicio 7

Cifrad input.bin con AES-192 en modo OFB, clave y vector de inicialización a elegir. Supongamos que la salida es output.bin.

En sintonía con lo explicado en el ejercicio anterior, cabe remarcar que AES-192 lo único que varía es el uso de claves de tamaño 192 bits. El tamaño de los bloques se sigue manteniendo en 128 bits así como las órdenes utilizadas para generar el cifrado. El fichero cifrado se puede observar en la Imagen 27.

```
$$ openssl enc -aes-192-ofb -in input.bin -out output.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad -iv
```

1234567890ABCDEF1234567890ABCDEF

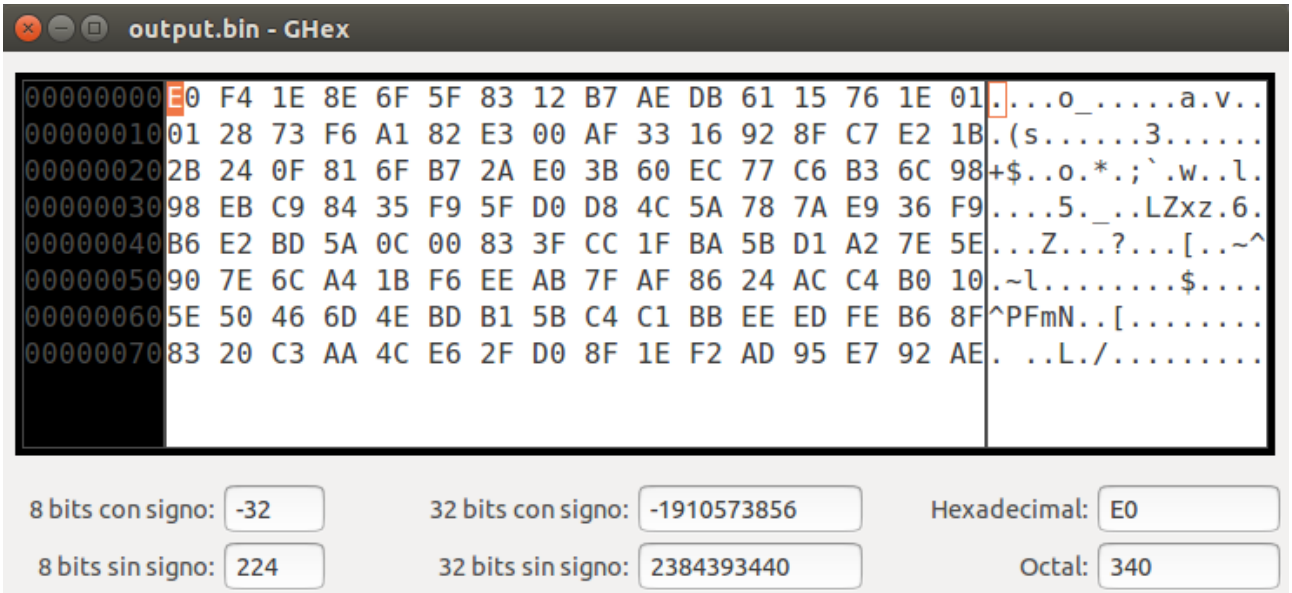


Imagen 27: Cifrado AES-192 en modo OFB con clave fuerte sobre el fichero input.bin.

Ejercicio 8

Descifra output.bin utilizando la misma clave y vector de inicialización que en 7.

Para descifrar un fichero cifrado hay que conocer el modo de cifrado, sus claves y su vector de inicialización si los tuvieran. En este caso, como lo hemos hecho nosotros mismos y son conocidos, basta con aplicar la funcionalidad de OpenSSL para descifrar, -d, y pasar como parámetros la clave y el IV utilizados.

```
$$ openssl aes-192-ofb -d -in output.bin -out output-descifrado.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad -iv
1234567890ABCDEF1234567890ABCDEF
```

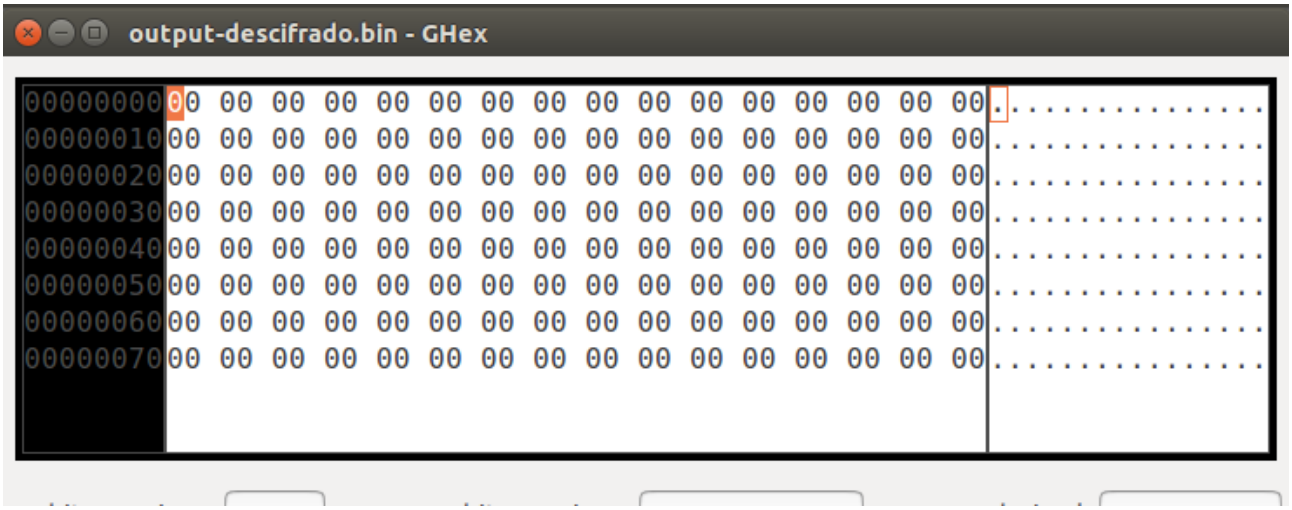


Imagen 27: Descifrado AES-192 en modo OFB con clave fuerte sobre el fichero output.bin.

Como se puede observar (Imagen 27), el fichero descifrado coincide con el fichero original.

Ejercicio 9

Vuelve a cifrar output.bin con AES-192 en modo OFB, clave y vector de inicialización del punto 7. Compara el resultado obtenido con el punto 8, explicando el resultado.

Lo primero que voy a hacer va a ser ejecutar la orden de cifrado para ver el resultado obtenido.

```
$$ openssl enc -aes-192-ofb -in output.bin -out output-cifrado.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad -iv
1234567890ABCDEF1234567890ABCDEF
```

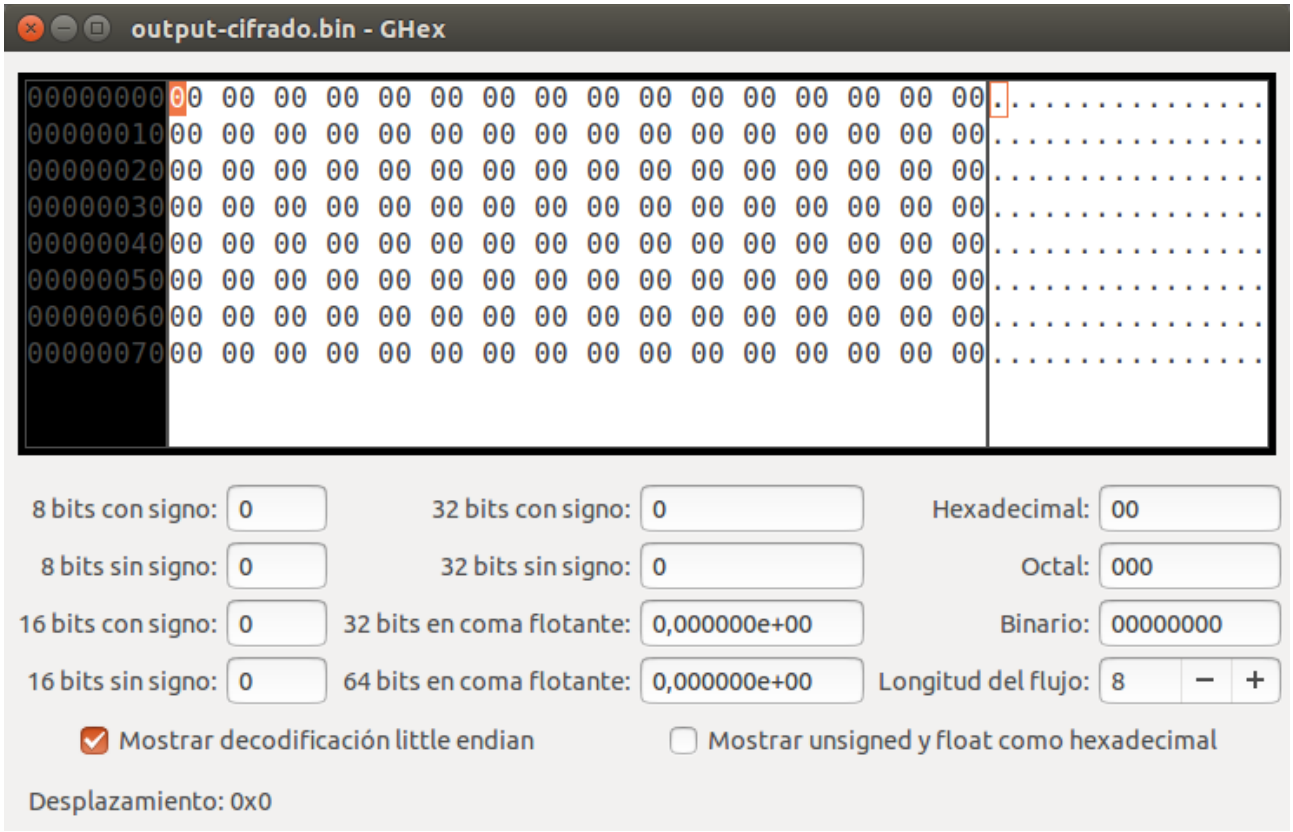


Imagen 28: Cifrado AES-192 en modo OFB con clave fuerte sobre el fichero output.bin.

Vemos que cifrando el fichero se obtiene de nuevo el fichero original, es decir, se ha descifrado (Imagen 28).

Esto ocurre por el funcionamiento del modo OFB. Este modo de cifrado coge un IV y lo cifra con el algoritmo elegido. El resultado lo suma con el mensaje que se desea cifrar y se obtiene el primer bloque cifrado. Para el segundo bloque cifrado, hay que realizar la suma del segundo bloque de mensaje junto con un nuevo cifrado del cifrado del IV anterior, y así sucesivamente (Imagen 29).

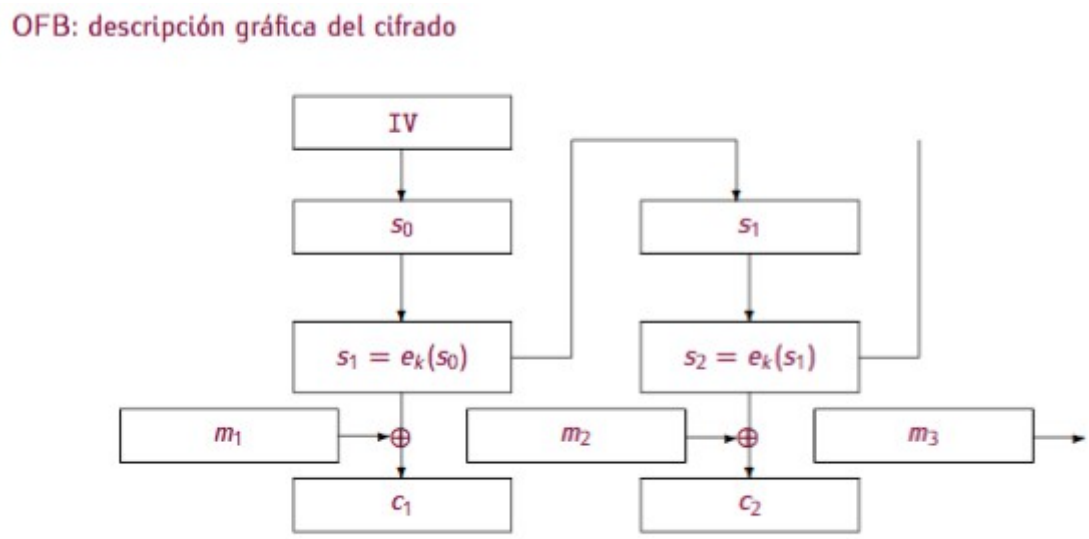


Imagen 29: Descripción gráfica del cifrado OFB.

Si ahora sustituimos y en lugar de pasar como mensaje el propio mensaje original le pasamos el mensaje cifrado, por las características de OFB se obtendrá en cada paso el bloque correspondiente del mensaje original (Imagen 30). Por tanto, descifrar usando OFB o cifrar dos veces produce el mismo resultado y es justo lo que se ha obtenido como conclusión de los ejercicios 8 y 9.

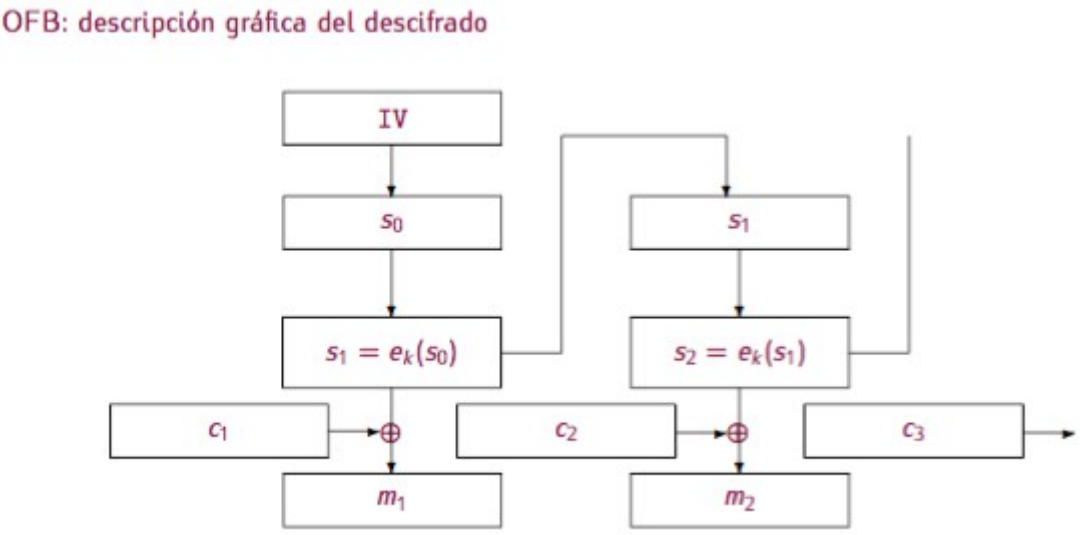


Imagen 30: Descripción gráfica del descifrado OFB.

Ejercicio 10

Presentad la descripción de otro cifrado simétrico que aparezca en vuestra implementación de OpenSSL.

En mi caso voy a elegir el algoritmo de encriptación CAST-128 (o CAST5 – Imagen 31). Toda la información relativa a este método se puede encontrar en el Request for Comments de [3].

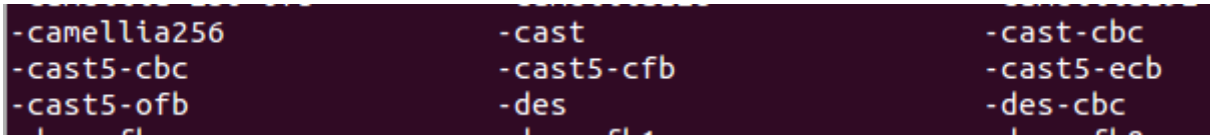


Imagen 31: Algunos de los cifrados implementados en mi versión de OpenSSL.

CAST-128 es un método de cifrado por bloques que se utiliza en algunas versiones de GPG y PGP. Fue creado por Carlisle Adams y Stafford Tavares (de ahí su nombre) en 1996, en la empresa Entrust Technologies. Este método está basado en las redes de Feistel, de hasta 16 rondas, y su funcionamiento es similar al DES. Utiliza bloques de tamaño 64 bits y claves de hasta 128 bits. Además, este método no posee claves débiles ni semidébiles.

Ejercicio 11

Repetid los puntos 3 a 5 con el cifrado presentado en el punto 10 (el 3 si el cifrado elegido tuviese claves débiles o semidébiles).

En este ejercicio voy a realizar el cifrado de los ficheros input.bin e input1.bin utilizando el método CAST-128 en modos ECB y CBC. Es de esperar que los resultados sean similares a los obtenidos cuando se utilizó DES. Hay que tener en cuenta que la clave tiene que ser de 128 bits y el vector de inicialización de 64 bits (igual al tamaño del bloque).

```
$$ openssl enc -cast5-ecb -in input.bin -out cast5-ecb-input.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad -iv 1234567890ABCDEF
```

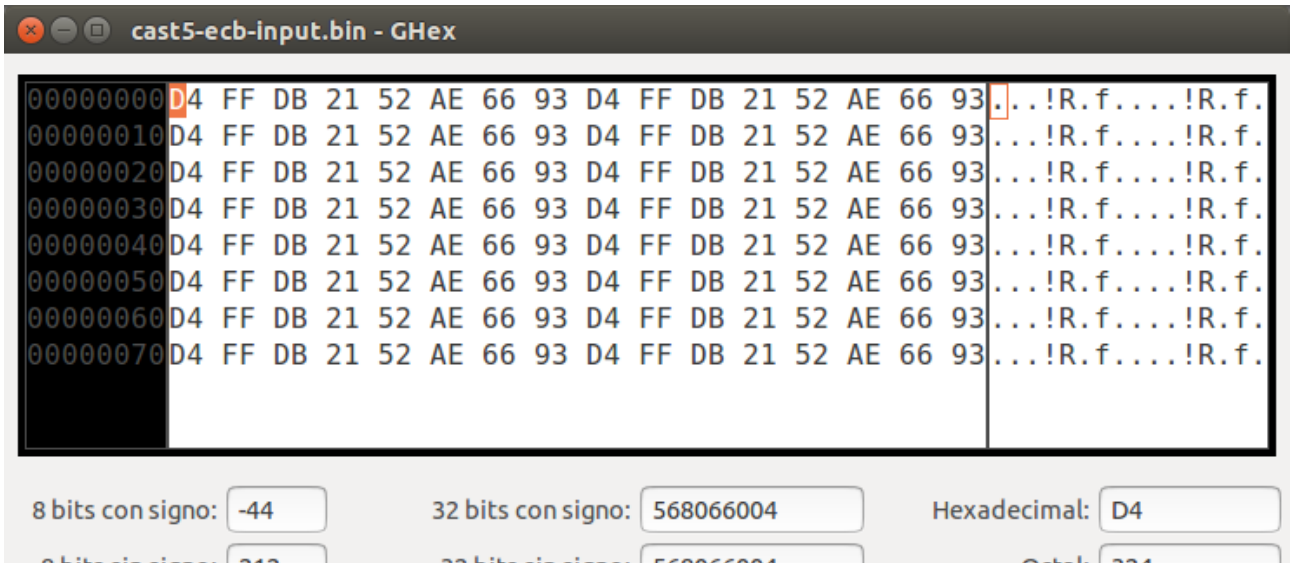


Imagen 32: Cifrado CAT-128 en modo ECB con clave fuerte sobre el fichero input.bin.

```
$$ openssl enc -cast5-ecb -in input1.bin -out cast5-ecb-input1.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad -iv 1234567890ABCDEF
```

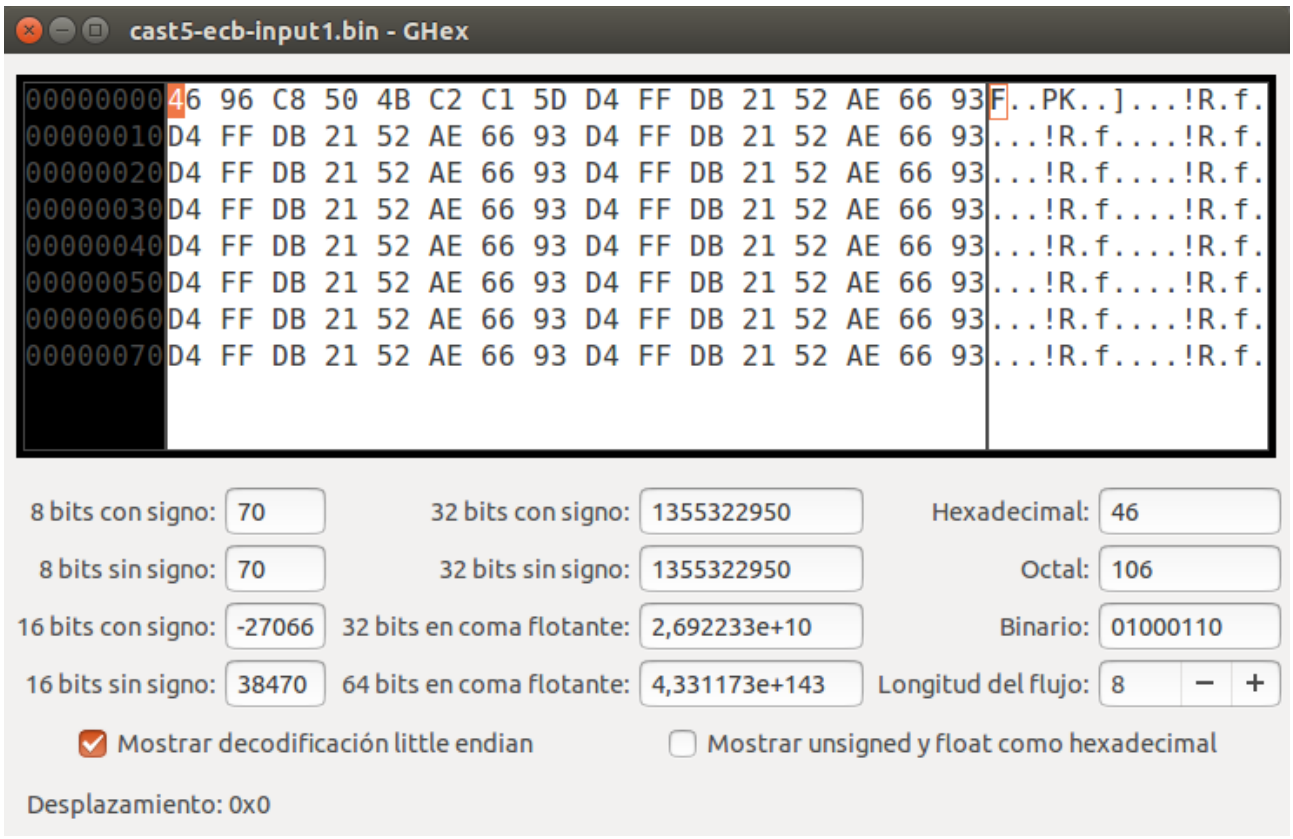


Imagen 33: Cifrado CAST-128 en modo ECB con clave fuerte sobre el fichero input1.bin.

```
$$ openssl enc -cast5-cbc -in input.bin -out cast5-cbc-input.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad -iv 1234567890ABCDEF
```

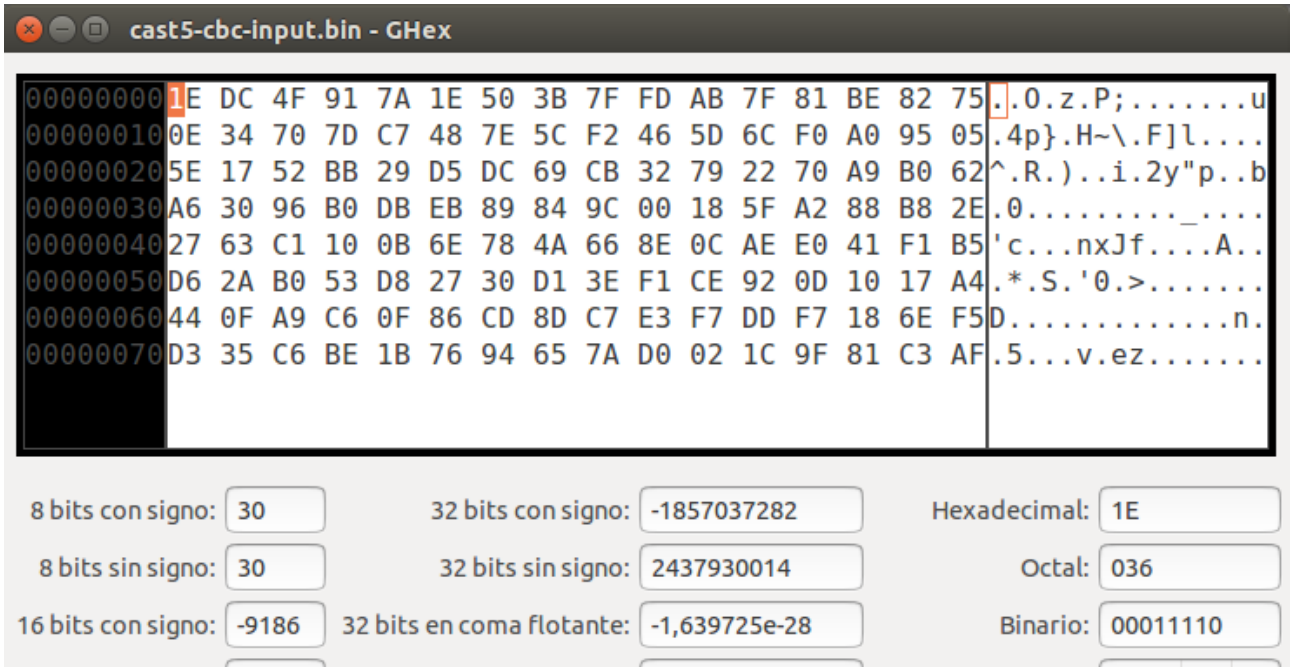


Imagen 34: Cifrado CAST-128 en modo CBC con clave fuerte sobre el fichero input.bin.

```
$$ openssl enc -cast5-cbc -in input1.bin -out cast5-cbc-input1.bin -K
1A2B3C4D5E6F78901A2B3C4D5E6F7890 -nopad -iv 1234567890ABCDEF
```

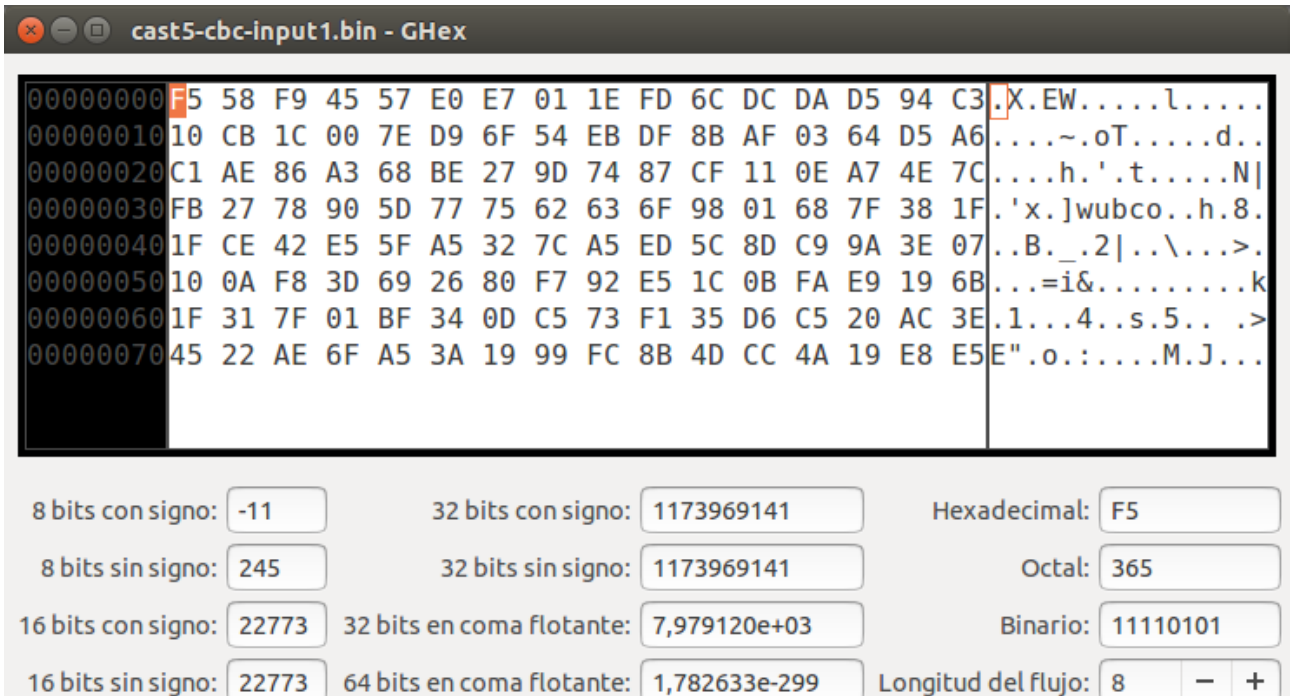


Imagen 35: Cifrado CAST-128 en modo CBC con clave fuerte sobre el fichero input1.bin.

Otra vez, es apreciable en las Imágenes 32 y 33 que el modo ECB hace que los bloques se repitan de nuevo constantemente, todos con el mismo tamaño. Además, de nuevo, una pequeña variación en el archivo original supone una variación únicamente en el bloque correspondiente (de nuevo porque la difusión y la confusión son propiedades locales a los bloques).

Así mismo, en el modo CBC de las Imágenes 34 y 35 se puede apreciar que el cifrado es mejor ya que no se producen repeticiones en los mensajes cifrados. Además, el cambio introducido en input1.bin hace que el fichero cambie por completo. Esto es justamente lo que ocurría en DES con clave fuerte. Es debido a que las subclaves de ronda son todas distintas y el orden en el que se usan para las rondas de la red de Feistel influye en el cifrado de cada bloque.

Conclusión

Con la realización de esta práctica me ha quedado claro el funcionamiento de los distintos algoritmos de cifrado usados, así como los distintos métodos de cifrado en bloques. Se ve que tanto el algoritmo en sí como las claves utilizadas para cifrar son elementos determinantes para decidir si un método es suficientemente seguro.
Está claro que el método ECB es muy malo, independientemente del algoritmo usado para encriptar. En cambio, los métodos que se realimentan son mejores sobre todo cuando se utilizan claves adecuadas, ya que generan archivos cuyos bloques no muestran relación alguna. Además queda clara la importancia de las propiedades de difusión y confusión a nivel global, ya que si su ámbito es local, los cambios introducidos solo afectan a un bloque en concreto.
En cuanto al uso de un algoritmo u otro, no es apreciable a simple vista cual es mejor, pero por lo visto en teoría se puede afirmar que el AES-256 sería el mejor de los utilizados ya que cuanto mayor es el tamaño de la clave menos vulnerable es el sistema.

Índice de imágenes

Bibliografía

[1] Claves débiles y semidébiles para DES
http://spi1.nisu.org/recop/al02/jgargallo/index.html#gclaves

[2] Manual de OpenSSL
https://www.openssl.org/docs/man1.0.2/apps/enc.html

[3] RFC2144 – CAST-128
https://tools.ietf.org/html/rfc2144