Seguridad y protección de sistemas informáticos (2017-2018)
Grado en Ingeniería Informática
Universidad de Granada

Criptosistemas Simétricos

Juan Ramón Gómez Berzosa

16 de octubre de 2017

Índice

Índice de figuras

Índice de tablas

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.

Para crear el archivo me he ayudado de la herramienta de linux **ghex**, de forma que he creado un archivo llamado "input.bin" y lo he rellenado con 0 usando dicha herramienta.

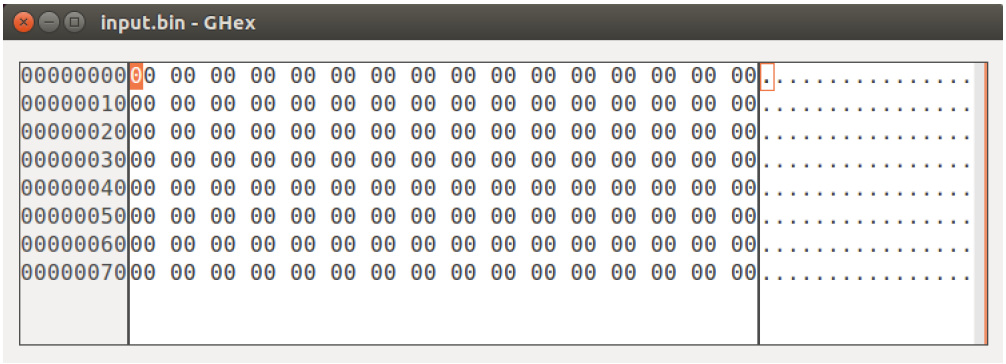

Figura 1.1: Contenido del archivo input.bin

Como podemos observar hemos rellenado desde la columna 0x00 a 0x7f, con lo cual obtenemos un fichero de 128 bytes (1024 bits).

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.

Para crear el archivo me he ayudado de la herramienta de linux **ghex**, de forma que he creado un archivo llamado "input1.bin" y lo he rellenado todo con 0 menos un bit con valor 1 dentro de los primeros 40 bits usando dicha herramienta.

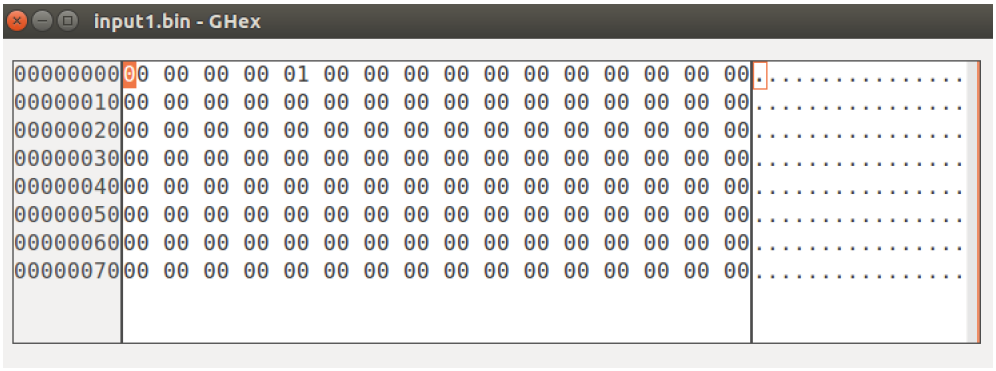

Figura 2.1: Contenido del archivo input1.bin

Como podemos observar hemos rellenado desde la columna 0x00 a 0x7f, introduciendo un bit con valor 1 en los primeros 40 bits, con lo cual obtenemos un fichero de 128 bytes (1024 bits).

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.

3.1. Modo ECB

3.1.1. Clave débil

En primer lugar he cifrado el fichero ”input.bin” con DES en modo ECB usando:

- **Clave:** 0101010101010101 (Débil)

Para ello he ejecutado el siguiente comando:
openssl des-ecb -K 0101010101010101 -in input.bin -out inputDES_ECB_Debil.encrypted

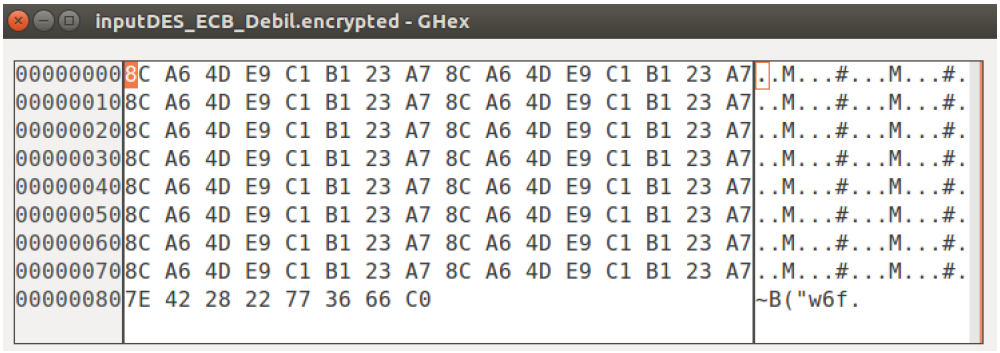

Figura 3.1: Contenido del archivo inputDES_ECB_Debil.encrypted

En la que podemos comprobar como se repiten cada 64 bits (cada bloque, ya que en DES es de 64 bits el tamaño del bloque) la misma secuencia de dígitos hexadecimales, es decir, se repiten cada uno de los bloques cifrados debido al funcionamiento del modo CBC. Esto se debe a que el fichero que estamos cifrando está relleno completamente de 0, por lo que todos los bloques en los que se divide el mensaje al ser cifrado con DES en modo CBC se cifran con el mismo valor al no haber realimentación y cifrar cada bloque por su cuenta.

Debido a este modo de funcionamiento, la confusión y la difusión se mantiene localmente en cada bloque, por lo que un atacante podría sustituir un bloque del cifrado por otro sin darnos cuenta.

El último bloque es de relleno, llamado **padding**, el cual se añade cuando ciframos en los modos **ecb** y **cbc**. Esto se hace porque la longitud del texto plano tiene que ser múltiplo del tamaño del bloque, en DES 64 bits. [5] Si hubiésemos añadido la opción "-nopad" no se hubiera añadido este último bloque.

3.1.2. Clave semidébil

En primer lugar he cifrado el fichero "input.bin" con DES en modo ECB usando:

- **Clave:** 01FE01FE01FE01FE (Semidébil)

Para ello he ejecutado el siguiente comando:
openssl des-ecb -K 01FE01FE01FE01FE -in input.bin -out inputDES_ECB_SemiDebil.encrypted

El resultado obtenido ha sido este:

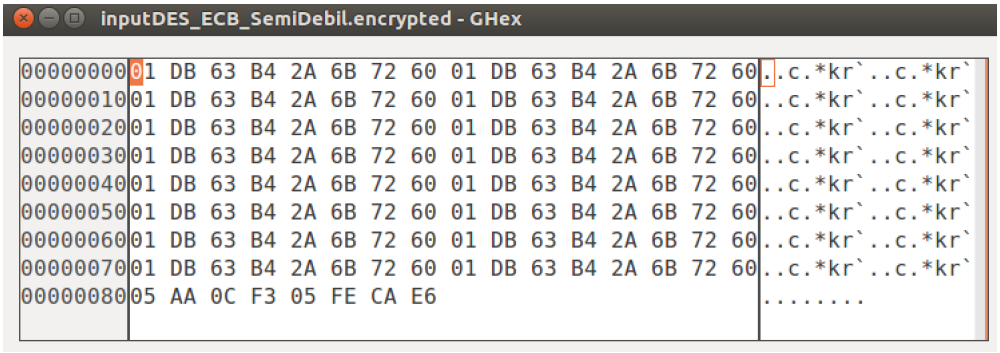

Figura 3.2: Contenido del archivo inputDES_ECB_SemiDebil.encrypted

En este caso podemos ver exactamente lo mismo que pasaba cuando ciframos con una clave débil, aunque como es lógico el criptograma es distinto debido a que la clave que hemos utilizado es distinta. También podemos observar como está el bloque de relleno.

3.2. Modo CBC

3.2.1. Clave débil

En primer lugar he cifrado el fichero "input.bin" con DES en modo CBC usando:

- **Clave:** 010101010101010101 (Débil)
- **Vector de inicialización:** 4FA92C5873672E20

Para ello he ejecutado el siguiente comando:
openssl des-cbc -K 0101010101010101 -iv 4FA92C5873672E20
-in input.bin -out inputDES_CBC_Debil.encrypted
El resultado obtenido ha sido este:

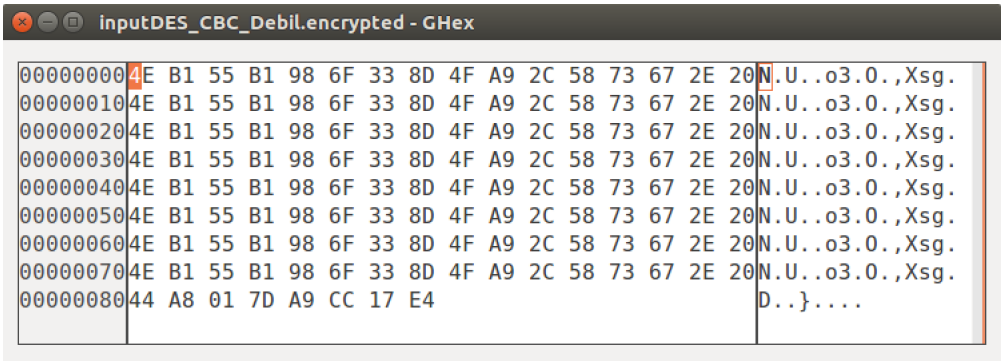

Figura 3.3: Contenido del archivo inputDES_CBC_Debil.encrypted

En este caso hemos empleado el modo cbc, en el que cada bloque no se cifra de forma independiente si no que depende siempre de los bloques anteriores. En primer lugar se

hace la suma OR exclusiva (XOR) entre el bloque que queremos cifrar y el bloque anterior y después se aplica el algoritmo de cifrado sobre este. Al primer bloque se le suma el contenido del vector de inicialización que será elegido sin ningún criterio específico, aunque sí cumpliendo tener el mismo tamaño del bloque. De esta forma, ahora este último bloque cifrado se usará para cifrar el siguiente bloque.

Si nos fijamos, podemos ver como siempre se repite la misma fila de 128 bits siendo la segunda mitad (el bloque segundo bloque de 64 bits) el vector de inicialización. Esto se debe a dos cosas:

- La clave que estamos empleando es una clave débil, por lo que sabemos que cada una de las subclaves de ronda que se generan en cada una de las 16 rondas de la red de Feistel del cifrado DES son exactamente la misma.
- El fichero esta completamente relleno de 0, por lo que todos los bloques en los que se subdivide el mensaje para ser cifrado son bloques de 64 bits rellenos de 0.

Al darse ambos casos y debido al funcionamiento de CBC, estamos todo el rato "cifrando y descifrando" el vector de inicialización. En primer lugar, se hace una operación XOR entre bloque inicial (relleno de 0) y el vector de inicialización, lo que hace que se quede igual y después se le aplica el cifrado de DES con la clave débil. En segundo lugar, este bloque se le aplicará una operación XOR con el siguiente bloque a cifrar (también relleno de 0) , por lo que todo volverá a quedarse igual. Por último, al volverle a aplicar el cifrado DES con la clave débil es como si estuviésemos descifrando ya que en verdad el cifrado y descifrado del modo DES es idéntico excepto en el orden de las subclaves, pero como con una clave débil se generan siempre las mismas subclaves no necesitaríamos el orden y nos volverá a dar el vector de inicialización.

En este caso podemos comprobar como la difusión y la confusión no son locales al bloque y afectan a los demás. También podemos observar que la última línea es de padding.

3.2.2. Clave semidébil

En primer lugar he cifrado el fichero "input.bin" con DES en modo CBC usando:

- **Clave:** 01FE01FE01FE01FE (Semidébil)
- **Vector de inicialización:** 4FA92C5873672E20

Para ello he ejecutado el siguiente comando:
openssl des-cbc -K 01FE01FE01FE01FE -iv 4FA92C5873672E20
-in input.bin -out inputDES_CBC_SemiDebil.encrypted
El resultado obtenido ha sido este:

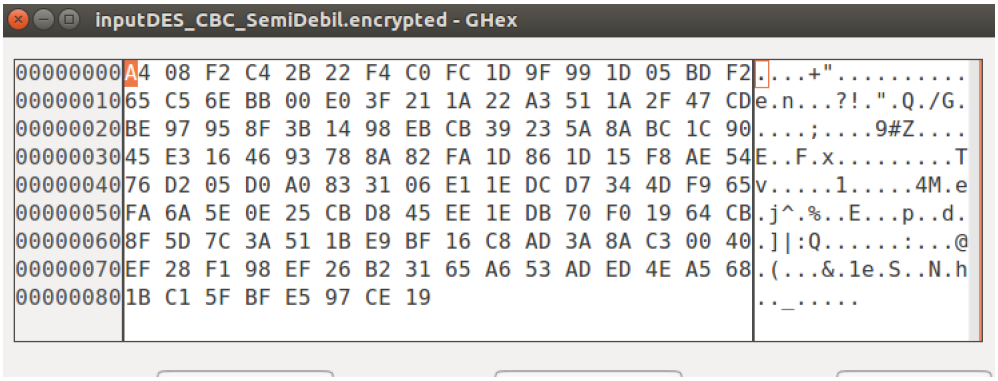

Figura 3.4: Contenido del archivo inputDES_CBC_SemiDebil.encrypted

Como podemos observar, con una clave semidébil no encontramos el vector de inicialización como con la clave débil. Esto se debe a que en este caso las subclaves de ronda generadas al usar una clave semidébil si que varían dos o cuatro, por lo que ahora si que importaría el orden de las subclaves en la red de Feistel y por tanto al aplicar otra vez el cifrado sobre el bloque nos daría un nuevo bloque cifrado y no el valor del vector de inicialización. También podemos observar el padding.

3.3. Modo OFB

3.3.1. Clave débil

En primer lugar he cifrado el fichero "input.bin" con DES en modo OFB usando:

- **Clave:** 0101010101010101 (Débil)
- **Vector de inicialización:** 4FA92C5873672E20

Para ello he ejecutado el siguiente comando:
openssl des-ofb -K 0101010101010101 -iv 4FA92C5873672E20 -in input.bin -out inputDES_OFB_Debil.encrypted
El resultado obtenido ha sido este:

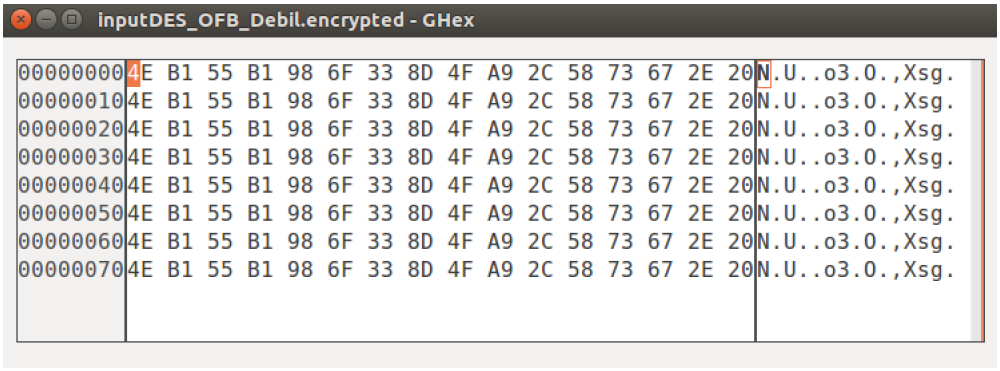

Figura 3.5: Contenido del archivo inputDES_OFB_Debil.encrypted

En este caso el modo que emplearemos será el modo OFB. En este caso, cuando vayamos a cifrar un bloque lo que haremos será que en primer lugar el cifrado se aplique sobre el bloque anterior y que después se haga una operación XOR entre el resultado del cifrado y el bloque que queríamos cifrar.

Si nos fijamos bien y volvemos al criptograma generado por el modo CBC con clave débil, podemos observar que son los ficheros idénticos excepto en un detalle que comentaré después. Esto se debe a que aquí pasa lo mismo que he comentado en el apartado de dicha salida, la única diferencia está en que en este caso el vector de inicialización se cifrá usando DES y la clave débil y a continuación se le aplicará la operación de XOR con el bloque que queríamos cifrar. Vuelve a pasar lo mismo con el ”cifrado y descifrado” que se produce para que aparezca el vector de inicialización también, al igual que en modo cbc aplicando DES.

La diferencia entre estos dos archivos es que en este modo no se ha introducido el bloque de relleno aún sin haber utilizado la opción -nopad. Esto se debe a que el modo CBC no requiere métodos para manejar los mensajes que no tienen longitudes que sea múltiplos del tamaño de bloque, dado que trabajan realizando una operación XOR entre el texto plano con la salida creada por el cifrador por bloques, lo que le es suficiente para que el texto cifrado resultante sea del mismo tamaño del último texto plano. [5]

3.3.2. Clave semidébil

En primer lugar he cifrado el fichero ”input.bin” con DES en modo OFB usando:

- **Clave:** 01FE01FE01FE01FE (Semidébil)
- **Vector de inicialización:** 4FA92C5873672E20

Para ello he ejecutado el siguiente comando:
openssl des-ofb -K 01FE01FE01FE01FE -iv 4FA92C5873672E20
-in input.bin -out inputDES_OFB_SemiDebil.encrypted

El resultado obtenido ha sido este:

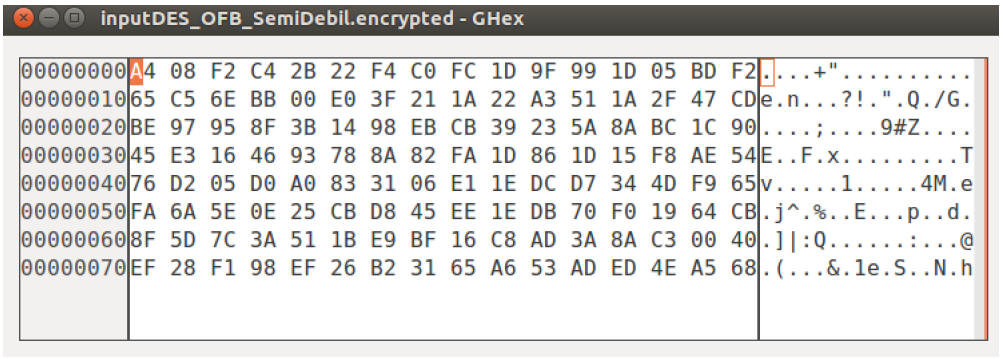

Figura 3.6: Contenido del archivo inputDES_OFB_SemiDebil.encrypted

En este caso, podemos comprobar que el criptograma es idéntico al obtenido al cifrar con DES el modo CBC. Esto se debe a lo mismo que he comentado antes del funcionamiento del modo OFB y de que en este caso el cifrado y descifrado de DES no es idéntico al utilizar una clave semidébil, ya que aquí si que importa el orden de las subclaves de ronda.

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.

4.1. Archivo "Input.bin"

En primer lugar he cifrado el fichero "input.bin" con DES en modo ECB usando:

- **Clave:** 0CC175B9C0F1B6A8

Para ello he ejecutado el siguiente comando:
openssl des-ecb -K 0CC175B9C0F1B6A8 -in input.bin -out inputDES_ECB_ej4.encrypted

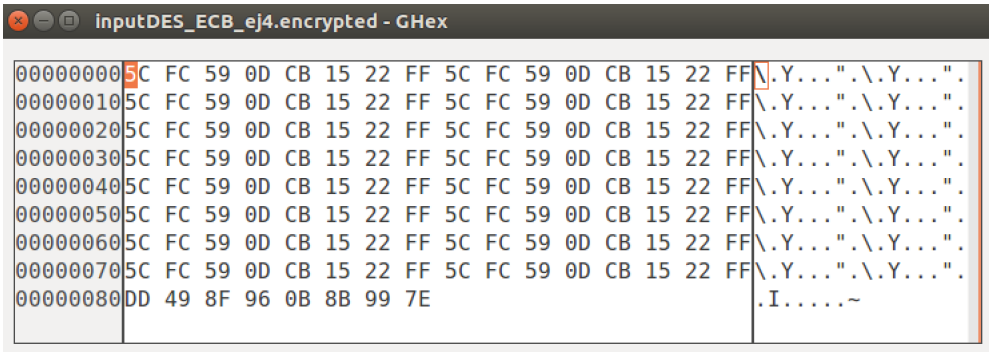

Figura 4.1: Contenido del archivo inputDES_ECB_ej4.encrypted

Podemos observar como se vuelven a repetir las secuencias de valores hexadecimales cada bloque de 64 bits. Al final podemos ver también la linea de padding.

4.2. Archivo "Input1.bin"

En primer lugar he cifrado el fichero "input.bin" con DES en modo ECB usando:

- **Clave:** 0CC175B9C0F1B6A8

Para ello he ejecutado el siguiente comando:
openssl des-ecb -K 0CC175B9C0F1B6A8 -in input1.bin -out input1DES_ECB_ej4.encrypted

El resultado obtenido ha sido este:

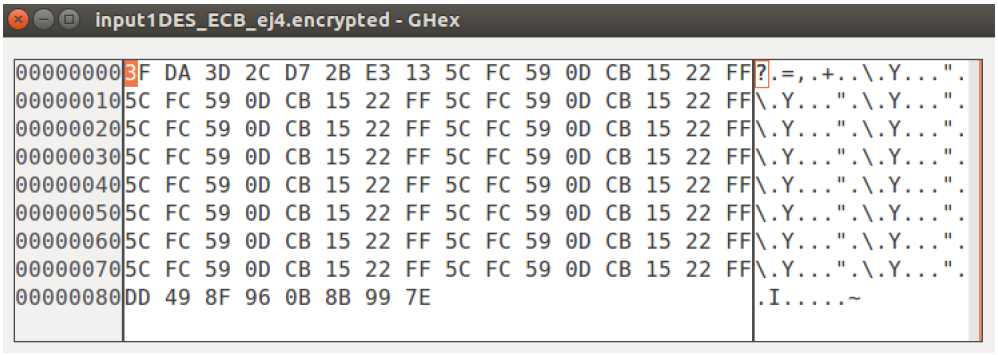

Figura 4.2: Contenido del archivo input1DES_ECB_ej4.encrypted

En este caso, el archivo "input1.bin" tiene un dígito con valor 1 entre los 40 primeros bits. Por tanto, esto ha afectado al valor del primer bloque como podemos observar, ya que el resto de bloques si están cifrados totalmente idénticos. Como la difusión y la confusión se mantienen locales al bloque, aunque varíe ese primer bloque el resto siguen totalmente iguales.

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.

5.1. Archivo "Input.bin"

En primer lugar he cifrado el fichero "input.bin" con DES en modo CBC usando:

- **Clave:** 0CC175B9C0F1B6A8
- **Vector de inicialización: 4FA92C5873672E20**

Para ello he ejecutado el siguiente comando:
openssl des-cbc -K 0CC175B9C0F1B6A8 -iv 4FA92C5873672E20 -in input.bin -out inputDES_CBC_ej4.encrypted

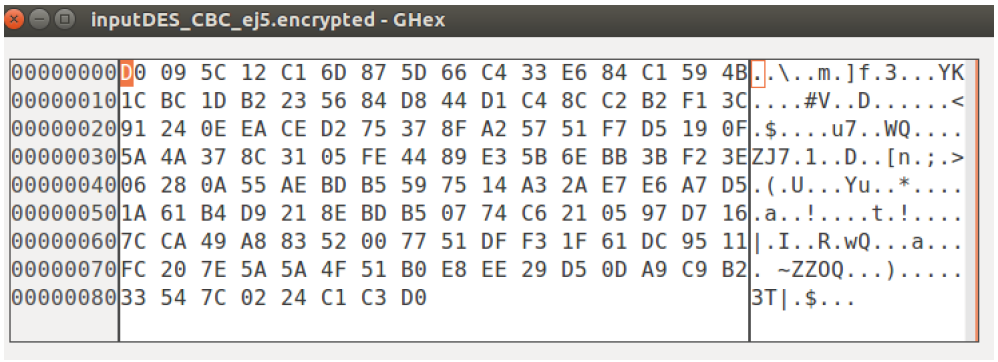

Figura 5.1: Contenido del archivo inputDES_CBC_ej4.encrypted

5.2. Archivo "Input1.bin"

En primer lugar he cifrado el fichero "input.bin" con DES en modo CBC usando:

- **Clave:** 0CC175B9C0F1B6A8
- **Vector de inicialización: 4FA92C5873672E20**

Para ello he ejecutado el siguiente comando:
openssl des-cbc -K 0CC175B9C0F1B6A8 -iv 4FA92C5873672E20 -in input1.bin -out input1DES_CBC_ej4.encrypted

El resultado obtenido ha sido este:

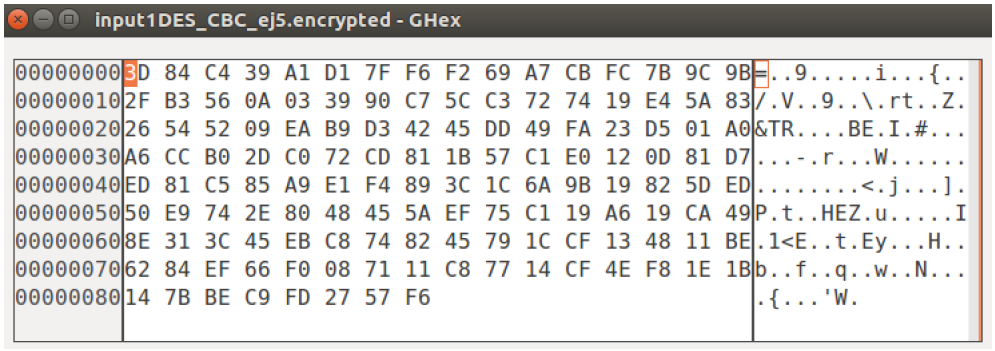

Figura 5.2: Contenido del archivo input1DES_CBC_ej4.encrypted

En estos ambos casos, ya no se repiten ningún bloque como con las claves débiles, ni aparece el vector de inicialización. Esto se debe a lo mismo que hemos comentado antes en las claves semidébiles del modo CBC con el cifrado DES, lo único que al no tratarse de una clave semidébil nos aseguramos que las subclaves de ronda sean distintas en el cifrado. Los criptogramas son distintos ya que en el "input1.bin" tenemos un dígito a 1 en los primeros 40 bits (primer bloque), lo cual provoca que se altere el resto de bloques del criptograma.

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

En este nuevo ejercicio vamos a utilizar como algoritmo criptográfico el AES, concretamente el de 128 y el de 256 bits, lo que nos indica el tamaño de la clave. El AES utiliza tamaños de bloque fijo de 128 bits. A diferencia de DES que se basaba en una red de Feistel, ahora podemos decir que se basa en una red de sustitución-permutación basada en un grupo multiplicativo, exactamente el grupo F_{256}. Surgió a partir del algoritmo de Rijndael. [1]

El algoritmo realiza una serie de operaciones intermedias en cada ronda, donde se aplican ya la difusión y la confusión. El número de rondas viene determinado por el número de bits del tamaño del bloque y el número de bits del tamaño de la clave, concretamente en nuestro caso serán 10 rondas para el aes-128 y 14 rondas para el aes-256. El tamaño del bloque es siempre de 128 bits en AES. Podemos observar como se vuelven a repetir las secuencias de valores hexadecimales cada bloque de 64 bits. Al final podemos ver también la linea de padding.

6.1. Modo ECB

6.1.1. Archivo "Input.bin"

En primer lugar he cifrado el fichero "input.bin" con AES-128 en modo ECB usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661

Para ello he ejecutado el siguiente comando:
openssl aes-128-ecb -K 0CC175B9C0F1B6A831C399E269772661 -in input.bin -out input_aes_128_ECB.encrypted

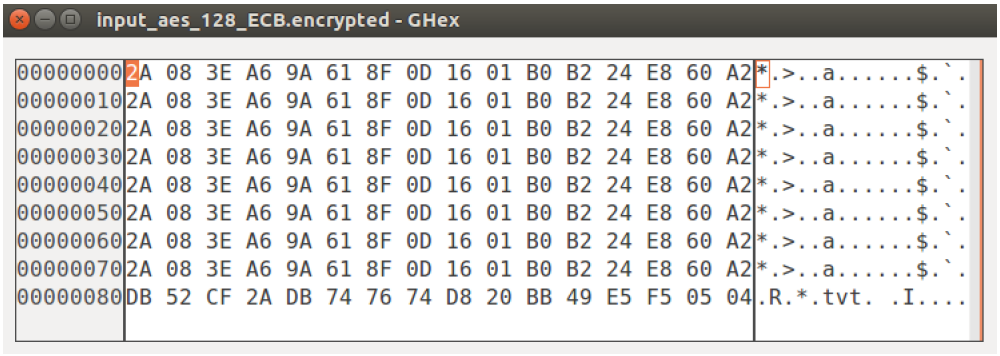

Figura 6.1: Contenido del archivo input_aes_128_ECB.encrypted

Ahora he cifrado el fichero ”input.bin” con AES-256 en modo ECB usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661CEC520EA51EA0A47E872 95FA3245A605

Para ello he ejecutado el siguiente comando:
openssl aes-256-ecb -K 0CC175B9C0F1B6A831C399E269772661CEC520EA5 1EA0A47E87295FA3245A605 -in input.bin -out input_aes_256_ECB.encrypted

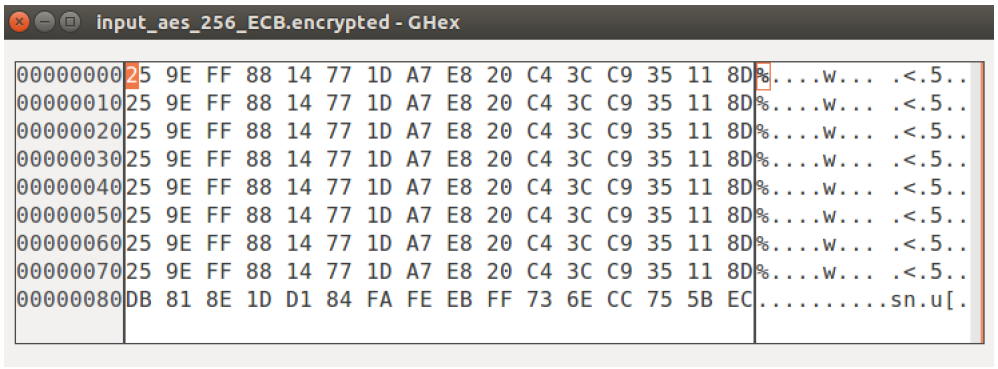

Figura 6.2: Contenido del archivo input_aes_256_ECB.encrypted

En el fichero de aes-128 hemos podido comprobar como al igual que pasaba en el modo ECB con el algoritmo DES y el cifrado del ”input.bin” se repite constantemente el mismo bloque de 128 bits como resultado del funcionamiento del modo ECB. Cabe destacar que a diferencia del DES ahora se repite cada bloque de 128 bits, debido a que el AES maneja

bloques de 128 bits siempre.

En el fichero de aes-256 podemos ver como también se repite el mismo bloque de 128 bits, la diferencia está en que aquí al tener una clave de 256 bits el número de rondas es también mayor, por lo que el criptograma es distinto.

Podemos ver en ambos ficheros como al final también se añade el bloque de relleno, debido al modo ECB y lo explicado anteriormente.

6.1.2. Archivo "Input1.bin"

En primer lugar he cifrado el fichero "input1.bin" con AES-128 en modo ECB usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661

Para ello he ejecutado el siguiente comando:
openssl aes-128-ecb -K 0CC175B9C0F1B6A831C399E269772661 -in input1.bin -out input1_aes_128_ECB.encrypted

El resultado obtenido ha sido este:

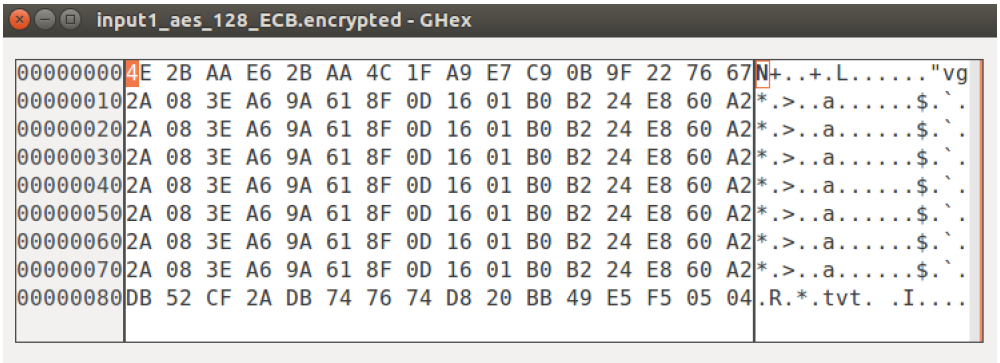

Figura 6.3: Contenido del archivo input1_aes_ECB.encrypted

Ahora he cifrado el fichero "input1.bin" con AES-256 en modo CBC usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661CEC520EA51EA0A47E872 95FA3245A605

Para ello he ejecutado el siguiente comando:
openssl aes-256-ecb -K 0CC175B9C0F1B6A831C399E269772661CEC520EA5 1EA0A47E87295FA3245A605 -in input1.bin -out input1_aes_256_ECB.encrypted

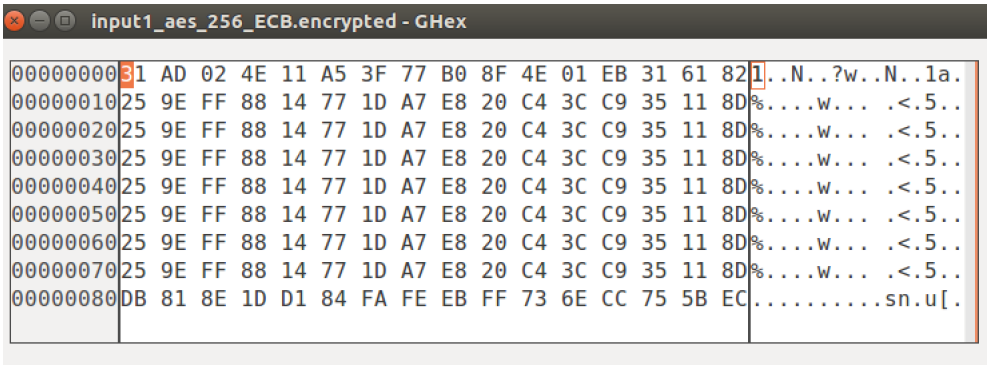

Figura 6.4: Contenido del archivo input1_aes_256_ECB.encrypted

En el fichero de aes-128 hemos podido comprobar como al igual que pasaba en el modo ECB con el algoritmo DES y el cifrado del "input1.bin" se repite constantemente el mismo bloque de 128 bits como resultado del funcionamiento del modo ECB, menos el primer bloque ya que este se corresponde con el bloque donde se encuentra el dígito 1 introducido en el texto plano.

Como en el modo ECB la difusión y la confusión se mantienen locales al bloque, aunque varíe ese primer bloque no afecta a los demás bloques.

En el fichero de aes-256 podemos ver como también se repite el mismo bloque de 128 bits menos el primero por lo comentado anteriormente, la diferencia está en que aquí al tener una clave de 256 bits el número de rondas es también mayor, por lo que el criptograma es distinto.

Podemos ver en ambos ficheros como al final también se añade el bloque de relleno, debido al modo ECB y lo explicado anteriormente.

6.2. Modo CBC

6.2.1. Archivo "Input.bin"

En primer lugar he cifrado el fichero "input.bin" con AES-128 en modo CBC usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661
- **Vector de inicialización:** 4FA92C5873672E20FB163A0BCB2BB4A4

Para ello he ejecutado el siguiente comando:
openssl aes-128-cbc -K 0CC175B9C0F1B6A831C399E269772661 -iv 4FA92C58 73672E20FB163A0BCB2BB4A4 -in input.bin -out input_aes_128_CBC.encrypted

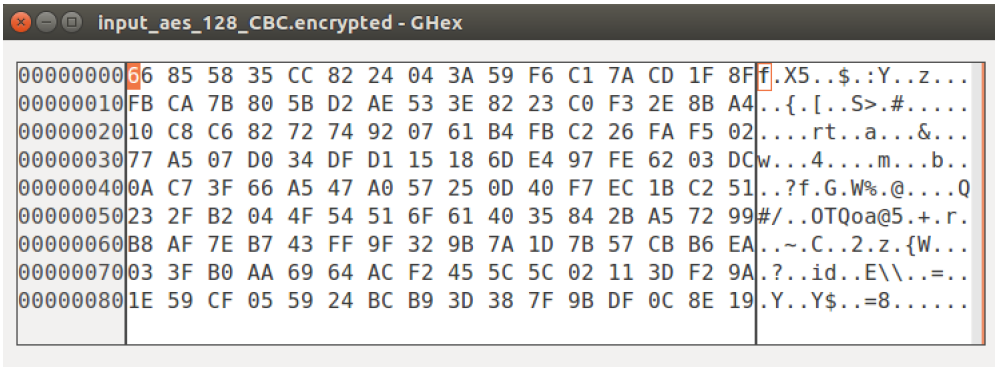

Figura 6.5: Contenido del archivo input_aes_128_CBC.encrypted

Ahora he cifrado el fichero "input.bin" con AES-256 en modo CBC usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661CEC520EA5 1EA0A47E87295FA3245A605
- **Vector de inicialización:** 4FA92C5873672E20FB163A0BCB2BB4A4

Para ello he ejecutado el siguiente comando:
openssl aes-256-ecb -K 0CC175B9C0F1B6A831C399E269772661CEC520EA5 1EA0A47E87295FA3245A605 -iv 4FA92C5873672E20FB163A0BCB2BB4A4 -in input.bin -out input_aes256_CBC.encrypted

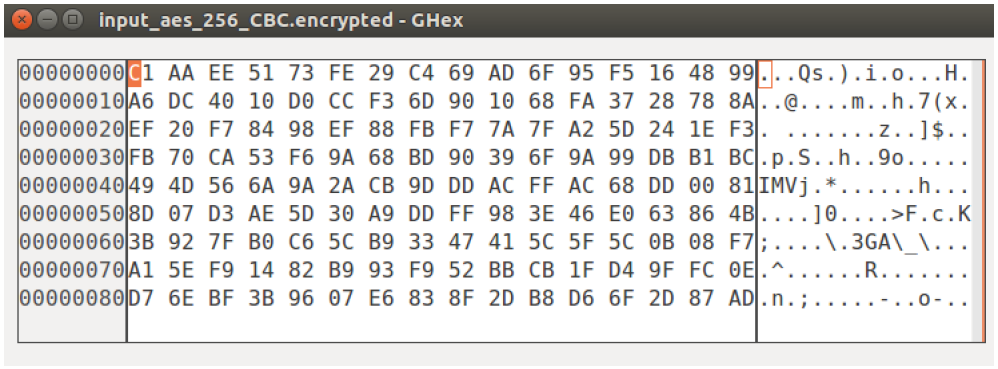

Figura 6.6: Contenido del archivo input_aes_256_CBC.encrypted

En el fichero de aes-128 hemos podido comprobar como al igual que pasaba en el modo CBC con el algoritmo DES con clave semidébil y el cifrado del "input.bin", el encadenamiento de bloques se realiza correctamente y no se repite nada ni aparece el vector de inicialización como cuando cifrábamos usando clave débil en el algoritmo DES, apareciendo ahora cada bloque con distinto valor hexadecimal.

En el fichero de aes-256 podemos ver como tampoco se repite nada, la diferencia está en que aquí al tener una clave de 256 bits el número de rondas es también mayor, por lo que el criptograma es distinto.

Podemos ver en ambos ficheros como al final también se añade el bloque de relleno, debido al modo CBC y lo explicado anteriormente.

6.2.2. Archivo "Input1.bin"

En primer lugar he cifrado el fichero "input1.bin" con AES-128 en modo CBC usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661
- **Vector de inicialización:** 4FA92C5873672E20FB163A0BCB2BB4A4

Para ello he ejecutado el siguiente comando:
openssl aes-128-cbc -K 0CC175B9C0F1B6A831C399E269772661 -iv 4FA92C5873672E20FB163A0BCB2BB4A4 -in input1.bin -out input1_aes_128_CBC.encrypted

El resultado obtenido ha sido este:

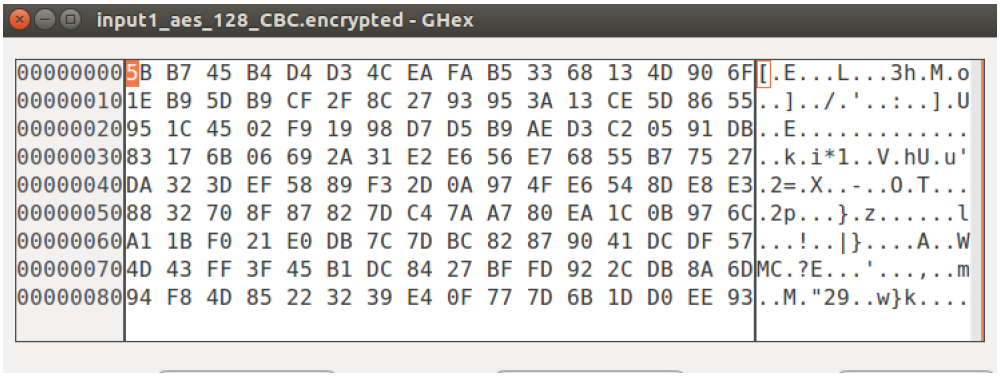

Figura 6.7: Contenido del archivo input1_aes_CBC.encrypted

Ahora he cifrado el fichero "input1.bin" con AES-256 en modo CBC usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661CEC520EA5 1EA0A47E87295FA3245A605
- **Vector de inicialización:** 4FA92C5873672E20FB163A0BCB2BB4A4

Para ello he ejecutado el siguiente comando:
openssl aes-256-cbc -K 0CC175B9C0F1B6A831C399E269772661CEC520EA51E A0A47E87295FA3245A605 -iv 4FA92C5873672E20FB163A0BCB2BB4A4 -in input1.bin -out input1_aes_256_CBC.encrypted

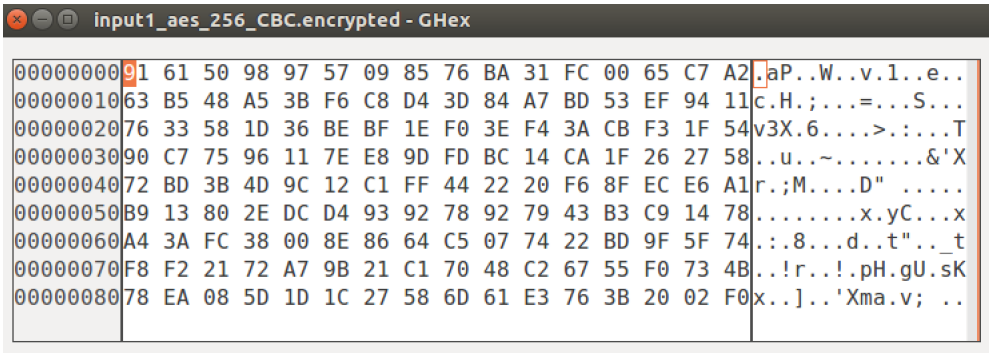

Figura 6.8: Contenido del archivo input1_aes_256_CBC.encrypted

En estos ambos casos aunque hayamos introducido un dígito con valor 1 dentro de los primeros 40 bits, afecta al cifrado del bloque y al siguiente encadenamiento con el resto, pero no nos dice nada a simple vista viendo el criptograma.

En el fichero de aes-128 hemos podido comprobar como al igual que pasaba en el modo CBC con el algoritmo DES con clave semidébil y el cifrado del "input1.bin", el encadenamiento de bloques se realiza correctamente y no se repite nada ni aparece el vector de inicialización como cuando cifrábamos usando clave débil en el algoritmo DES, apareciendo ahora cada bloque con distinto valor hexadecimal.

En el fichero de aes-256 podemos ver como tampoco se repite nada, la diferencia está en que aquí al tener una clave de 256 bits el número de rondas es también mayor, por lo que el criptograma es distinto.

Podemos ver en ambos ficheros como al final también se añade el bloque de relleno, debido al modo CBC y lo explicado anteriormente.

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 primer lugar he cifrado el fichero "input.bin" con AES-192 en modo OFB usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661CEC520EA51EA0A47
- **Vector de inicialización:** 4FA92C5873672E20FB163A0BCB2BB4A4

Para ello he ejecutado el siguiente comando:
openssl aes-192-ofb -K 0CC175B9C0F1B6A831C399E269772661CEC520EA51EA 0A47 -iv 4FA92C5873672E20FB163A0BCB2BB4A4 -in input.bin

-out output.bin

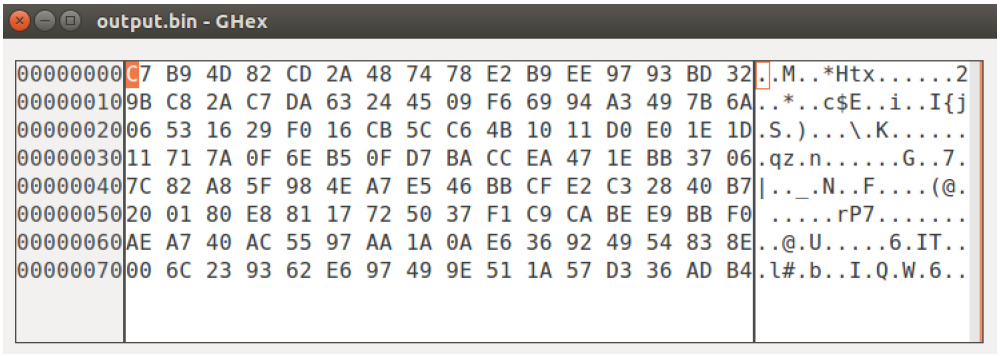

Figura 7.1: Contenido del archivo output.bin

Como podemos apreciar, se aplica el cifrado AES pero ahora con 12 rondas debido al tamaño de 192 bits de la clave. No se añade bloque de relleno como característica del modo OCB, como ya comentamos anteriormente.

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

En primer lugar he descifrado el fichero "output.bin" con AES-192 en modo OFB usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661CEC520EA51EA0A47
- **Vector de inicialización:** 4FA92C5873672E20FB163A0BCB2BB4A4
- Para descifrar hemos usado la opción **-d**.

Para ello he ejecutado el siguiente comando:
openssl aes-192-ofb -d -K 0CC175B9C0F1B6A831C399E269772661CEC520EA 51EA0A47 -iv 4FA92C5873672E20FB163A0BCB2BB4A4 -in output.bin -out outputDecrypt.bin

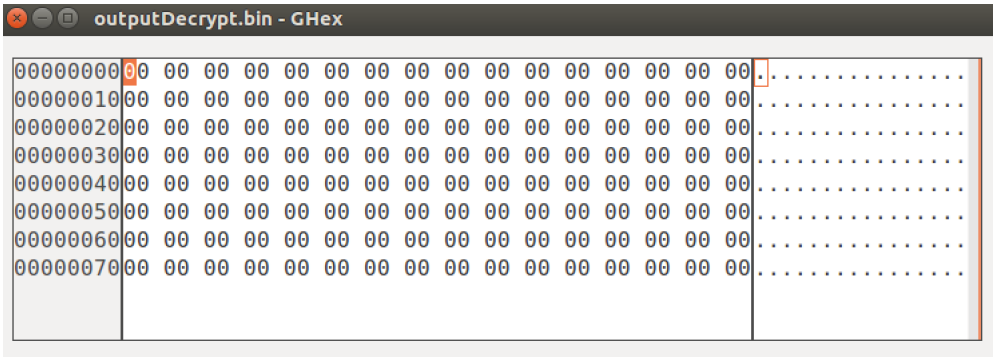

Figura 8.1: Contenido del archivo outputDecrypt.bin

Como vemos, el archivo ha vuelto a quedarse relleno de 0, se ha descifrado correctamente.

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.

En primer lugar he cifrado el fichero ”output.bin” con AES-192 en modo OFB usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661CEC520EA51EA0A47
- **Vector de inicialización:** 4FA92C5873672E20FB163A0BCB2BB4A4

Para ello he ejecutado el siguiente comando:
openssl aes-192-ofb -K 0CC175B9C0F1B6A831C399E269772661CEC520EA51EA 0A47 -iv 4FA92C5873672E20FB163A0BCB2BB4A4 -in output.bin -out outputEj9.bin

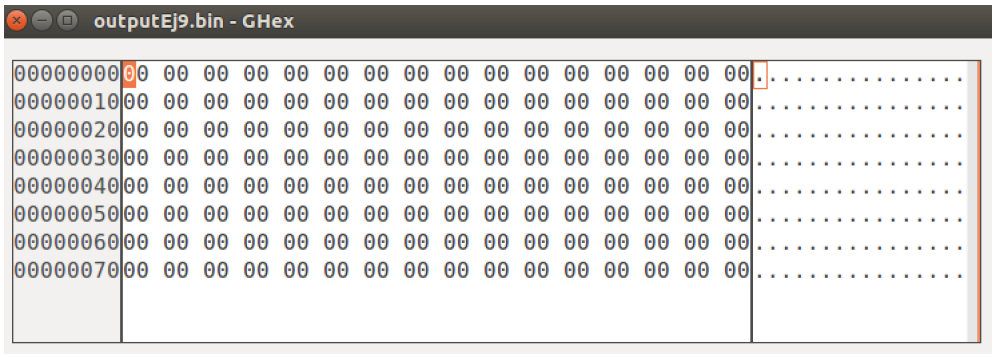

Figura 9.1: Contenido del archivo outputEj9.bin

Como podemos ver, el archivo también aparece entero relleno de 0. El primer archivo es lógico ya que hemos aplicado el descifrado en las mismas condiciones (misma clave e iv). El segundo archivo, como hemos comentado, también esta relleno de 0, esto se debe a que en el modo OFB el cifrado como el descifrado se hace de la misma forma por lo que es normal que obtengamos la misma salida que el fichero de texto original.

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

El algoritmo que he elegido es el **Camellia Cipher**. Fue desarrolloado por Nippon Telegraph, Telephone Corporation y Mitsubish Electric Corporation en el año 2000. Se trata de un cifrado de clave simétrica por bloques, con un tamaño de bloque fijo de 128 bits (al igual que AES) y un tamaño de clave variable entre 128, 192 y 256 bits (mismos que AES). [3] [2]

El cifrado de Camellia tiene niveles de seguridad comparables con los del AES. Fue diseñado para ser adecuado tanto para implementaciones hardware como software. También es utilizado dentro del protocolo IPSEC. [4]

Camellia es una red de Feistel de 18 rondas (con claves de 128 bits) o 24 rondas (cuando se usan claves de 192 o 256 bits). Cada 6 rondas se aplica una transformación lógica y además también se emplean en el cifrado las S-Cajas. También aplica difusión y confusión. [3]

También podemos decir que hasta ahora no se han encontrado claves débiles para este cifrado.[4]

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 primer lugar cabe decir que como no existen claves débiles ni semidébiles conocidas para el cifrado de Camellia el ejercicio 3 no lo vamos a realizar. En los demás ejercicios se aplicará el cifrado de camellia usando claves de 128 bits (camellia-128 en OpenSSL).

11.1. Modo ECB

11.1.1. Archivo "Input.bin"

En primer lugar he cifrado el fichero "input.bin" con camellia-128 en modo ECB usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661

Para ello he ejecutado el siguiente comando:
openssl camellia-128-ecb -K 0CC175B9C0F1B6A831C399E269772661 -in input.bin -out input_128_ecb.encrypted

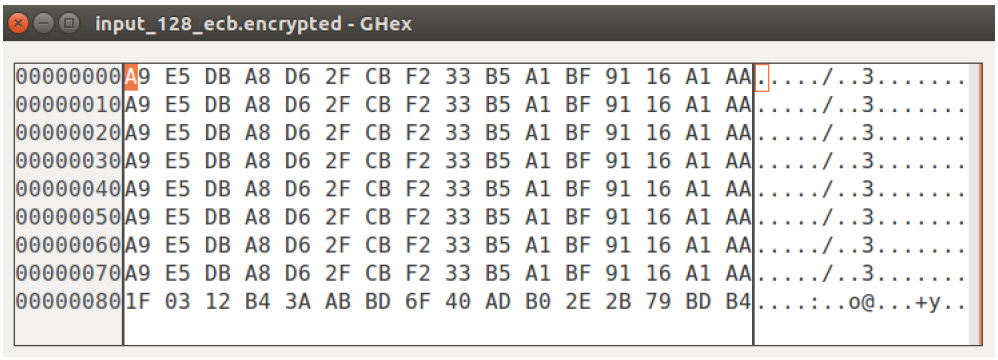

Figura 11.1: Contenido del archivo input_128_ECB.encrypted

Como podemos observar, al igual que pasaba con el cifrado AES o DES, se repiten todos los bloques de 128 bits, debido al funcionamiento del modo ECB ya explicado anteriormente. También podemos ver el bloque de relleno en la última fila.

11.1.2. Archivo "Input1.bin"

En primer lugar he cifrado el fichero "input1.bin" con camellia-128 en modo ECB usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661

Para ello he ejecutado el siguiente comando:
openssl camellia-128-ecb -K 0CC175B9C0F1B6A831C399E269772661 -in input1.bin -out input1_128_ecb.encrypted

El resultado obtenido ha sido este:

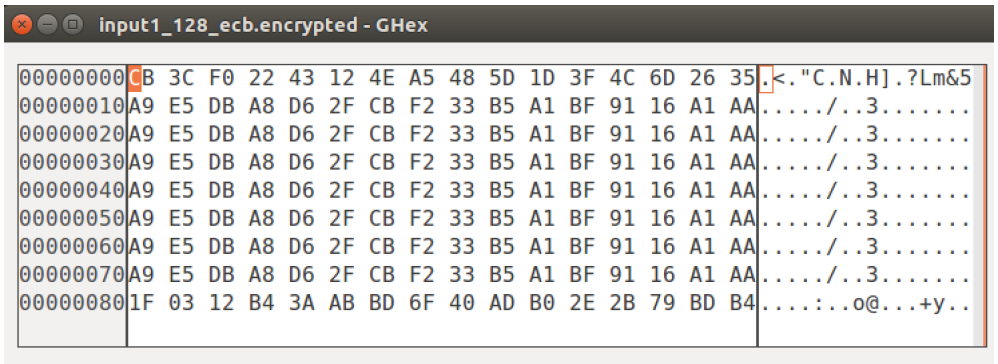

Figura 11.2: Contenido del archivo input1_aes_ECB.encrypted

Tal y como pasaba en AES y DES con este mismo modo de cifrado por bloques, podemos ver como se repiten todos los bloques a partir del segundo, ya que el primero fue donde introdujimos el dígito 1 y observamos como esto no afecta a los siguientes bloques ya que no hay realimentación en el modo CBC. También podemos observar la línea de padding.

11.2. Modo CBC

11.2.1. Archivo "Input.bin"

En primer lugar he cifrado el fichero "input.bin" con camellia-128 en modo CBC usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661
- **Vector de inicialización:** 4FA92C5873672E20FB163A0BCB2BB4A4

Para ello he ejecutado el siguiente comando:
openssl camellia-128-cbc -K 0CC175B9C0F1B6A831C399E269772661 -iv 4FA92C5873672E20FB163A0BCB2BB4A4 -in input.bin -out input_128_cbc.encrypted

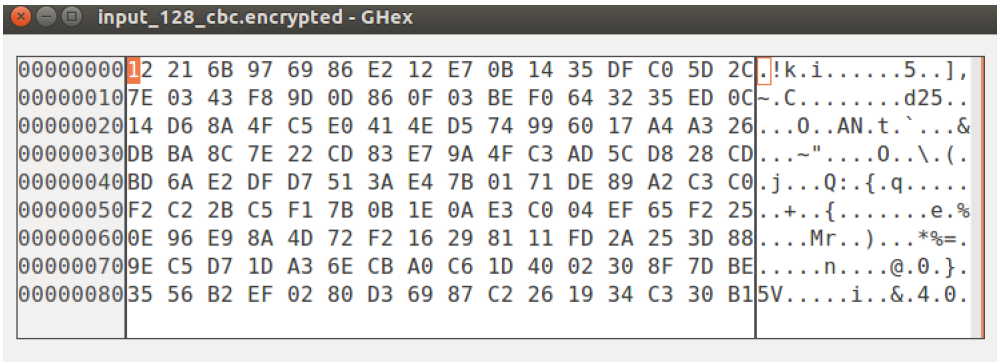

Figura 11.3: Contenido del archivo input_aes_128_CBC.encrypted

Tal y como pasaba en el modo CBC con el algoritmo DES y AES con clave semidébil o cualquier clave que no fuera débil y el cifrado del "input.bin", el encadenamiento de bloques se realiza correctamente y no se repite nada ni aparece el vector de inicialización como cuando cifrábamos usando clave débil en el algoritmo DES, apareciendo ahora cada bloque con distinto valor hexadecimal. Esto se debe al correcto funcionamiento del algoritmo y que en este caso las sublcaves de ronda generadas por la clave al no ser ni débil ni semidébil son distintas.

Podemos ver en el fichero como al final también se añade el bloque de relleno.

11.2.2. Archivo "Input1.bin"

En primer lugar he cifrado el fichero "input1.bin" con AES-128 en modo CBC usando:

- **Clave:** 0CC175B9C0F1B6A831C399E269772661
- **Vector de inicialización:** 4FA92C5873672E20FB163A0BCB2BB4A4

Para ello he ejecutado el siguiente comando:
openssl camellia-128-cbc -K 0CC175B9C0F1B6A831C399E269772661 -iv 4FA92C5873672E20FB163A0BCB2BB4A4 -in input1.bin -out input1_128_cbc.encrypted

El resultado obtenido ha sido este:

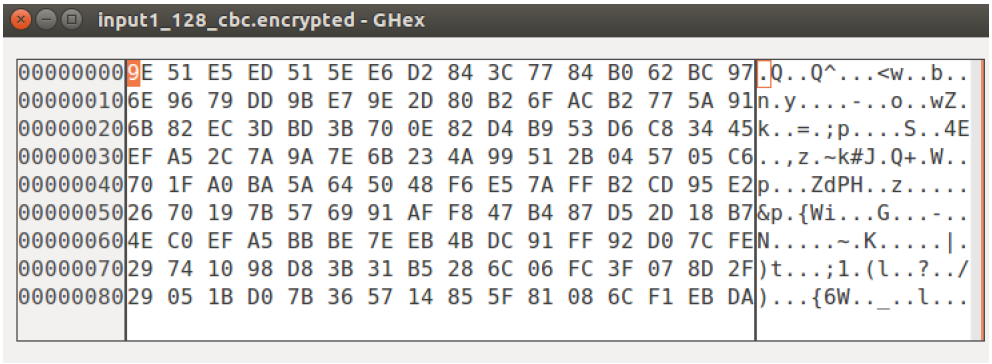

Figura 11.4: Contenido del archivo input1_aes_CBC.encrypted

Podemos aplicar la misma explicación para este cifrado que cuando ciframos el "input.bin". Lo único que le hace distinto es que aquí cambiamos un dígito del primer bloque (introducimos un 1), por lo que el primer bloque es distinto y ya se van encadenando los cambios en los distintos bloques consecuentes.

Referencias

[1] Algoritmo aes. https://es.wikipedia.org/wiki/Advanced_Encryption_Standard.

[2] Camellia cipher - rfc. https://tools.ietf.org/html/rfc3713.

[3] Camellia cipher - wikipedia. https://en.wikipedia.org/wiki/Camellia_(cipher).

[4] Camellia y su uso en el protocolo ipsec. https://tools.ietf.org/html/rfc4312.

[5] Modos de operación de una unidad de cifrado por bloques. https://es.wikipedia.org/wiki/Modos_de_operaci%C3%B3n_de_una_unidad_de_cifrado_por_bloques.