

UNIVERSIDAD AUTÓNOMA DE MADRID



**FUNDAMENTOS DE CRIPTOGRAFÍA Y SEGURIDAD
INFORMÁTICA
(2019 - 2020)**

PRÁCTICA 2

Aitor Melero Picón
Ana Roa González
Grupo: 1461

Madrid, 25/11/2020

ORGANIZACIÓN Y FUNCIONAMIENTO DEL CÓDIGO.

Antes de comenzar con la parte teórica de la práctica hay que describir tanto la organización como el funcionamiento del código. Todos los ficheros de la práctica se encuentran dentro de la carpeta “g03” tal como se indica en la normativa de las propias prácticas. Dentro de la misma carpeta se encuentran todos los ficheros de código, al principio los teníamos separados en subcarpetas por apartados, pero así no venía estipulado en la normativa.

Los ficheros que contienen la función principal para cada ejercicio son:

- seg_perf.c: Main para el apartado 1.
- desCBC.c: Main para el apartado 2a.
- CBC_TDA.c: Main para el apartado 2b.
- no_linealidad_SBoxes_DES.c: Main para el apartado 3a.
- avalancha_DES.c: Main para el apartado 3b.
- no_linealidad_SBoxes_AES.c: Main para el apartado 4a.
- sbbox_AES.c: Main para el apartado 4b.

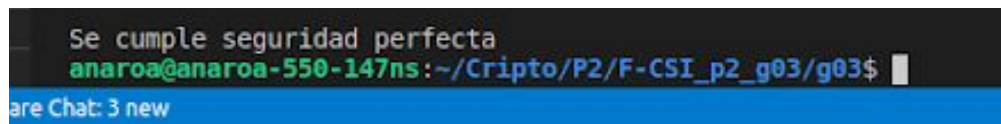
El resto de ficheros que completan la práctica son:

- DES_tables.c: Tablas proporcionadas para el cifrado DES.
- AES_tables.c: Tablas proporcionadas para el cifrado AES.
- CBC.h: Cabecera para el cifrado CBC.
- CBC.c: Implementación del cifrado CBC.
- DES.h: Cabecera para el cifrado DES.
- DES.c: Implementación del cifrado DES.
- TDEA.h: Cabecera para el cifrado del triple DES.
- TDEA.c: Implementación del cifrado del triple DES.
- type.h: Cabecera para recursos personales.
- Makefile: Makefile para compilar y ejecutar los programas de la práctica.
- Textos_Planos/: Carpeta donde se guardan los textos planos a cifrar.
- Textos_Cifrados/: Carpeta donde se guardan los textos cifrados a descifrar.
- Tablas_AES/: Carpeta donde se guardan las tablas resultantes de ejecutar el apartado 4b.

Para compilar el código completo de la práctica tan solo hay que ejecutar “make” y ya se dispondrá de todos los ejecutables. Para ejecutar cada programa se recomienda observar la parte de los ejecutables del make (también se muestra por pantalla cada entrada a cada programa en caso de que se ejecute cada fichero), aún así, veremos ejemplos de ejecución a lo largo del presente documento.

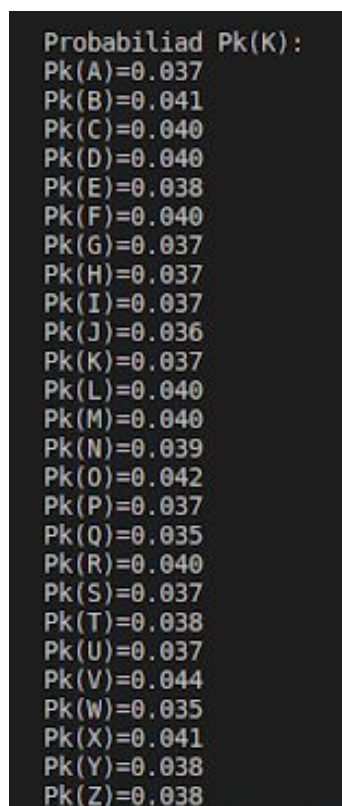
APARTADO 1. Seguridad Perfecta.

Para poder ejecutar el ejercicio ejecutamos “make apartado_1”, esto cifra un párrafo del Quijote con una clave equiprobable. Para ejecutarlo con una clave no equiprobable, tenemos que ejecutar ./seg_perf -l -i don_quixote.txt -o apartado1.txt. El resultado de ejecutar la opción por defecto del makefile es el siguiente:



```
Se cumple seguridad perfecta
anaroa@anaroa-550-147ns:~/Cripto/P2/F-CST_p2_g03/g03$
```

Para la opción de cifrar con una clave equiprobable, la probabilidad de la clave sale más o menos igual para todas las claves, como podemos observar en la siguiente imagen:



```
Probabilidad Pk(K):
Pk(A)=0.037
Pk(B)=0.041
Pk(C)=0.040
Pk(D)=0.040
Pk(E)=0.038
Pk(F)=0.040
Pk(G)=0.037
Pk(H)=0.037
Pk(I)=0.037
Pk(J)=0.036
Pk(K)=0.037
Pk(L)=0.040
Pk(M)=0.040
Pk(N)=0.039
Pk(O)=0.042
Pk(P)=0.037
Pk(Q)=0.035
Pk(R)=0.040
Pk(S)=0.037
Pk(T)=0.038
Pk(U)=0.037
Pk(V)=0.044
Pk(W)=0.035
Pk(X)=0.041
Pk(Y)=0.038
Pk(Z)=0.038
```

La probabilidad de la clave ($P_k(K)$), la probabilidad del texto plano ($P_p(X)$) y la probabilidad del texto cifrado ($P_c(Y)$) los calculamos contando cuantas veces se genera la clave aleatoriamente, el carácter plano y el carácter cifrado en el texto, respectivamente y luego lo dividimos entre el número de caracteres totales del texto. La probabilidad $P_c(Y|X)$ la calculamos sumando las $P_k(K)$ de la clave usada para cifrar de X a Y.

Para saber si es seguridad perfecta tenemos que comprobar $P_p(X|Y) = P_p(X)$, calculamos $P_p(X|Y) = (P_c(Y|X) * P_p(X)) / P_c(Y)$. Como las probabilidades dependen de la longitud del texto le damos un margen de 0.05 para saber si $P_p(X|Y) = P_p(X)$ se cumple. Como se muestra en la primera imagen, el propio programa nos devuelve si la condición se cumple, pero además se imprimen por pantalla las probabilidades $P_p(X)$, $P_c(Y)$ y $P_p(X|Y)$.

Para la opción de cifrar con una clave no equiprobable, utilizamos un conjunto de 40 claves aleatorias de 0 25, de esta forma la $P_k(K)$ no es igual para todas las letras y no se cumple la condición de la seguridad perfecta.

```
Probabilidad  $P_k(K)$ :  
Pk(A)=0.139  
Pk(B)=0.072  
Pk(C)=0.000  
Pk(D)=0.120  
Pk(E)=0.024  
Pk(F)=0.070  
Pk(G)=0.024  
Pk(H)=0.048  
Pk(I)=0.072  
Pk(J)=0.024  
Pk(K)=0.024  
Pk(L)=0.049  
Pk(M)=0.026  
Pk(N)=0.000  
Pk(O)=0.045  
Pk(P)=0.047  
Pk(Q)=0.050  
Pk(R)=0.024  
Pk(S)=0.000  
Pk(T)=0.025  
Pk(U)=0.000  
Pk(V)=0.022  
Pk(W)=0.025  
Pk(X)=0.000  
Pk(Y)=0.023  
Pk(Z)=0.047
```

Y como antes el programa nos contesta diciendo que no se cumple la condición de seguridad perfecta.

```
NO se cumple seguridad perfecta  
anaroo@anaroo-550-147ns:~/Cripto/P2/F-COI_p2_g03/g03$
```

APARTADO 2. Implementación del DES.

Apartado A. Programación del DES.

Primero vamos a hablar de la ejecución del programa `./desCBC`, el programa que cifra o descifra un fichero usando el cifrado DES con el modo de operación CBC. Para cifrar un fichero de entrada (`prueba.txt`) y dejar el resultado cifrado en un fichero de salida (`CBC_cifrado.dec`) se debe ejecutar “make apartado_2a_1” en la terminal para la ejecución con los datos por defecto o ejecutar “`./desCBC -C -iv <iv> -i <fichero_entrada> -o <fichero_salida>`”. El resultado de cifrar con las opciones por defecto del makefile es el siguiente:

```
aitor@mele27:~/Escritorio/Examen_Cripto/F-CSI_p2_g03/g03$ make apartado_2a_1
./desCBC -C -iv 12345678 -i prueba.txt -o CBC_cifrado.dec
#####
CLAVE DE CIFRADO: 25AD9DE06E8C8051
#####
```

Como puede verse en la imagen, hemos cifrado el contenido del fichero “prueba.txt” con el vector de inicialización “12345678” y hemos guardado el resultado en el fichero “CBC_cifrado.dec”. Como resultado puede verse por pantalla la clave de 64 bits generada en hexadecimal de manera aleatoria y con paridad impar en los bits especificados en el enunciado de la práctica. Ahora, si queremos descifrar el fichero “CBC_cifrado.dec” debemos introducir “./desCBC -D -k <clave_64-bits> -iv <iv> -i <fichero_cifrado> -o <fichero_salida>”, en este caso no es recomendable usar la opción por defecto del makefile porque tiene una clave por defecto que no es la generada aleatoriamente en el caso anterior. La ejecución es la siguiente:

```
aitor@mele27:~/Escritorio/Examen_Cripto/F-CSI_p2_g03/g03$ ./desCBC -D -k 25AD9DE06E8C8051 -iv 12345678 -i CBC_cifrado.dec -o CBC_solucion.txt
aitor@mele27:~/Escritorio/Examen_Cripto/F-CSI_p2_g03/g03$ cat Textos_Planos/CBC_solucion.txt
EN UN LUGAR DE LA MANCHA, CUYO
NOMBRE
NO QUIERO ACORDARME...
```

Como puede verse en la imagen, si mostramos el texto plano generado (guardado en la carpeta “Textos_Planos/” automáticamente por el programa, el resultado es el correcto.

Ahora vamos a hablar de la implementación del programa. En primer lugar, a la hora de tratar los bits nos las hemos tenido que ingeniar con el tipo de dato unsigned char de C. Un unsigned char de C es un tipo de dato formado por 8 bits sin signo. Con esto, hemos ido realizando las correspondientes operaciones de bits jugando de 8 en 8 bits. Las operaciones relevantes de bits se encuentran en el fichero “operaciones_bit.c”.

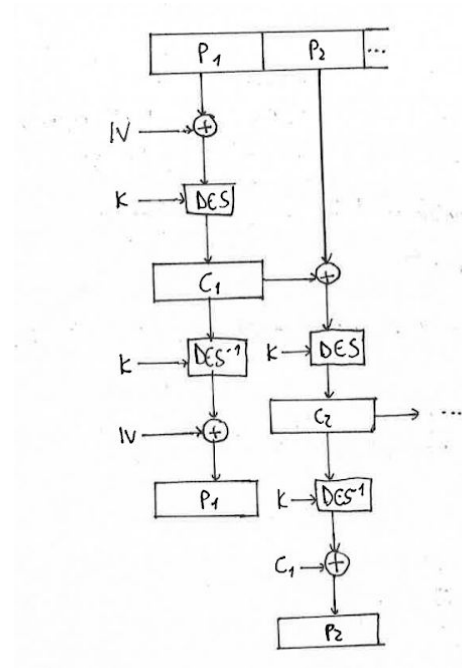
En cuanto al algoritmo de cifrado DES, este está implementado en el fichero “DES.c”. Básicamente, hemos implementado una función para cada transformación, por ejemplo, una función para la transformación IP, otra función para el SWAP, otra función para E, ..., y así con todas.

Hay que destacar el cómo hemos conseguido realizar los cambios para cada tabla prestada en “DES_tables.c”. Lo que hemos hecho es ir guardando el bit de la posición correspondiente a guardar en una variable con la operación AND, de tal manera que la variable tendrá todos los bits a 0 menos el bit de la posición correspondiente que puede valer 1 en caso de que ese sea su valor. Después hacemos un XOR con el bloque solución (bloque con todos los cambios), lo que hará que se vayan escribiendo uno a uno los bits con su valor y en su posición correspondientes. Esta operación, la hemos realizado para la transformación IP, IP^{-1} , E, P, PC-1 y PC-2. Cabe destacar que también hemos seguido este proceso para el desplazamiento de LCS, para ello hemos creado las tablas con la posición de los bits desplazados.

Además de ir cambiando los bits de la forma descrita con las tablas, también nos las hemos tenido que ingeniar con el cambio de las S-BOXES. Para ello íbamos sacando la fila y la columna de cada bloque de 6 bits con una operación AND poniendo a 0 los bits que no nos interesaban y finalmente íbamos guardando los valores de las S-BOXES y desplazándolos a la izquierda hasta obtener el valor adecuado.

La función cifrar_DES, se encarga de ir llamando a todas las funciones del proceso de cifrado del DES, por lo que esta función, junto a la de descifrar_DES, no han tenido mucha complicación. Todo el cifrado del DES se probó observando la secuencia de bits generada en cada fase con <http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm>. Imprimíamos los bits con la función print_binint de "operaciones_bit.c".

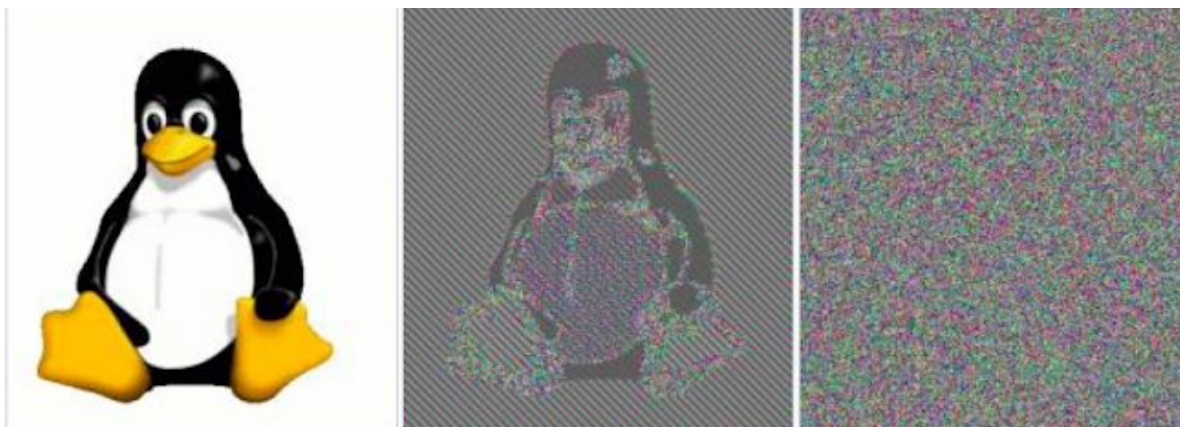
Para cifrar con el modo de operación CBC tenemos la función cifrar_CBC y descifrar_CBC (fichero CBC.c) que siguen la estructura de la imagen en los apuntes de la práctica.



Como puede verse en la imagen, se va cifrando cada bloque de texto plano con el XOR del bloque cifrado anterior, salvo el primer bloque de texto plano, se cifra el XOR del bloque de texto plano con el vector de inicialización. En el caso del descifrado, cada bloque de texto plano será el resultado del XOR del bloque cifrado anterior con el resultado del descifrado, salvo el primer bloque en el que de nuevo se hace el XOR con el vector de inicialización.

Por lo tanto, para cifrar con DES en el modo CBC, nuestro programa principal (desCBC.c) llama a las funciones del CBC (CBC.c) que llaman a su vez a las funciones del DES (DES.c).

Con respecto a las diferencias entre el cifrado DES con ECB y DES con CBC, se pueden observar y explicar mejor con la siguiente imagen de https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation:



La primera fotografía es la original, la segunda es el resultado de cifrar la primera imagen usando el modo ECB y la última el resultado de cifrar la primera imagen usando otro modo más aleatorio. Se puede ver que la segunda imagen, la del ECB, muestra un contenido que

se puede apreciar, se puede intuir la imagen en claro. Esto se debe a que el modo ECB va cifrando cada bloque plano con la misma clave y con independencia del resto de bloques planos, lo que en el caso de que haya repeticiones, es decir, en caso de que existan bloques planos iguales, el resultado del cifrado va a ser el mismo. En el caso de la fotografía de arriba, los bloques planos que representen un mismo color como el blanco, el negro o el amarillo obtendrán el mismo bloque cifrado porque representan el mismo bloque plano del fichero. Es por ello que se pueden observar trazos con un diferente aspecto pero que permiten visualizar la forma de la figura. Todo esto ocurre porque se repite texto plano y es por ello que no es recomendable usar el modo ECB con textos grandes y estructurados, porque cuanto más grande sea el texto, más repeticiones tendrá y más similitudes se observarán en el cifrado final. Con CBC no ocurre esto ya que CBC crea dependencia entre cada bloque plano porque cada bloque cifrado es el resultado de cifrar un bloque plano con el bloque cifrado anterior (operación XOR). Esto hace que se eliminen las repeticiones a la hora de cifrar y por lo tanto también hace que no se puedan encontrar bloques cifrados idénticos, obtenemos un resultado similar al de la última fotografía.

Apartado B. Programación del Triple DES.

Al igual que el programa anterior, la ejecución del `./TDEA-CBC`, el programa que cifra o descifra un fichero usando el cifrado triple DES con el modo de operación CBC, sigue las mismas pautas. Para cifrar un fichero de entrada (`prueba.txt`) y dejar el resultado cifrado en un fichero de salida (`CBC_cifrado.dec`) se debe ejecutar `make apartado_2b_1` en la terminal para la ejecución con los datos por defecto o ejecutar `./TDEA-CBC -C -iv <iv> -i <fichero_entrada> -o <fichero_salida>`. El resultado de cifrar con las opciones por defecto del makefile es el siguiente:

```
anaroa@anaroa-550-147ns:~/Cripto/P2/F-CSE_p2_g03/g03$ make apartado_2b_1
./TDEA-CBC -C -iv 12345678 -i prueba.txt -o CBC_cifrado.dec
#####
CLAVE DE CIFRADO: 6E3434924AA7C7E65445D3E6A70E4631ADD2597C7C1834C
#####
```

Como puede verse en la imagen, el programa cifra el fichero de entrada y devuelve la clave aleatoria de 192 bits (dividida en tres claves de 64 bits cada una) generada y con la paridad en los bits especificados.

Para descifrar es recomendable poner los argumentos a mano y no usar el makefile ya que este tiene una clave predefinida de antemano. Los argumentos para descifrar con el triple DES con el modo CBC son `./TDEA-CBC -D -k <clave_192-bits> -iv <iv> -i <fichero_cifrado> -o <fichero_salida>`. Aquí un ejemplo del descifrado para el fichero cifrado anterior:

```
anaroa@anaroa-550-147ns:~/Cripto/P2/F-CSE_p2_g03/g03$ ./TDEA-CBC -D -k 6E3434924AA7C7E65445D3E6A70E4631ADD2597C7C1834C -iv 12345678 -i CBC_cifrado.dec -o TDEAsolucion.txt
anaroa@anaroa-550-147ns:~/Cripto/P2/F-CSE_p2_g03/g03$ cat TDEAsolucion.txt
cat: TDEAsolucion.txt: No existe el archivo o el directorio
anaroa@anaroa-550-147ns:~/Cripto/P2/F-CSE_p2_g03/g03$ cat Textos_Planos/TDEAsolucion.txt
EN UN LUGAR DE LA MANCHA, CUYO
NOMBRE
NO QUIERO ACORDARME...
```

Para este ejercicio hemos creado la función que cifra con el triple DES en “TDEA.c”. Básicamente el programa “TDEA_CBC.c” llama a cifrar_TDEA o descifra_TDEA que a su vez llama a cifrar_CBC o descifrar_CBC usando 3 claves y con la siguiente estructura:

- **CIFRADO:** $Y = E(K3, D(K2, E(K1, X)))$
- **DESCIFRADO:** $X = D(K1, E(K2, D(K3, Y)))$

donde:

- Y = bloque cifrado
- X = bloque plano
- E = función de cifrado
- D = función de descifrado
- Kn = clave n

APARTADO 3. Principios de diseño del DES.

Apartado A. Estudio de la no linealidad de las S-boxes del DES.

El objetivo de este apartado es probar la no linealidad de las S-boxes del DES y para ello se tiene que cumplir lo siguiente:

$$f(x + y) \neq f(x) + f(y)$$

Esto quiere decir que la salida de una S-box con entrada el xor de dos bloques no es igual que el xor de la salida de la S-box para cada bloque. Esto es una de las claves para que un algoritmo de cifrado sea más o menos seguro. Cuanta mayor no linealidad mayor seguridad presentará un cifrado.

Para probar experimentalmente la no linealidad de las S-boxes del DES, en el fichero “no_linealidad_SBoxes_DES.c” generamos los 64 bloques posibles para 6 bits 2 veces (sin contar repeticiones) y realizamos las operaciones correspondientes para probarlos para cada una de las 8 S-boxes del DES. En caso de que haya linealidad en algún caso, los datos de este se muestran en decimal por pantalla para comprobarlo. Para ejecutar el programa que prueba la no linealidad de las S-boxes del DES basta con poner “make apartado_3a” para ejecutar con el makefile o “./no_linealidad_SBoxes_DES”. El resultado de la ejecución es el siguiente:


```
#####
#####
CASO LINEAL:
num_caja = 7
x = 60
y = 62
f(x) = 9
f(y) = 2
f(x + y) = 11
f(x) + f(y) = 11
#####
#####
CASO LINEAL:
num_caja = 5
x = 61
y = 62
f(x) = 5
f(y) = 14
f(x + y) = 11
f(x) + f(y) = 11
#####

#####
CASOS NO LINEALES: 15869
CASOS LINEALES: 771
CASOS TOTALES: 16640
PORCENTAJE NO LINEALIDAD: 95.366587
#####
aitor@mele27:~/Escritorio/Examen_Cripto/F-CSTI_p2_g03,
```

Como puede verse, los casos lineales se muestran por pantalla (nos sirvió para debugear), pero lo que nos interesa son los datos finales.

De 16640 casos, en 771 se cumple la linealidad y en 15869 no se cumple, lo que muestra que las S-boxes del DES no tienen linealidad en un 95,36 % de los casos, una linealidad bastante alta y que demuestra que el DES es un algoritmo de cifrado bastante seguro.

Apartado B. Estudio del Efecto de Avalancha.

En este apartado hay que probar el cambio que puede producir un solo bit de diferencia tanto en el bloque a cifrar como en la clave a la hora de usar el algoritmo de cifrado del DES. Para ello, nuestro programa ("avalancha_DES.c"), genera un bloque y una clave de 64 bits aleatoria que serán sometidos al cambio de un bit en una posición aleatoria (en el caso de la clave se cambian 2 bits por la paridad, pero no influye porque los bits de paridad no se usan por PC1). Después, se cifran el texto original y el modificado con la misma clave y se van observando los cambios en las 16 rondas correspondientes de la función F por el cambio del bit del bloque a cifrar y, finalmente, se cifra el bloque original con la clave original y la modificada y se van observando los cambios de los resultados para las 16 rondas de la

función F.

Para ejecutar este programa basta con poner "make apartado_3b" para ejecutar con el makefile o "./avalancha_DES". El resultado de la ejecución es el siguiente:

```
aitor@mele27:~/Escritorio/Examen_Cripto/F-CESI_p2_g03/g03$ make apartado_3b
./avalancha_DES
#####
##### ESTUDIO AVALANCHA #####
#####
TEXTO: 01101100 01100100 10111000 01110011 10011111 10000110 01001011 11010011
CLAVE: 00101100 10000101 00101111 00011100 00110111 11011001 11001000 01110000

TEXTO CAMBIADO: 01101100 01100100 10111000 01110011 10011111 10000110 01001011 11010010
CLAVE CAMBIADA: 00101100 10000101 00101111 00011100 00110111 11001000 11001000 01110000

AVALANCHA PRODUCIDA POR EL BLOQUE DE TEXTO

#####
##### RONDA 1 #####
TEXTO ORIGINAL: 10110100 00001111 01010101 11111000 00101111 11011000 00111001 10010011
TEXTO CAMBIADO: 10110100 00001111 01010101 11111000 00101111 11011000 00111001 00010011
CAMBIAN 1 BITS.
#####

##### RONDA 2 #####
TEXTO ORIGINAL: 00101111 11011000 00111001 10010011 01101001 10001001 00011000 10101101
TEXTO CAMBIADO: 00101111 11011000 00111001 00010011 01111001 10001001 00111100 10100100
CAMBIAN 6 BITS.
#####

##### RONDA 3 #####
TEXTO ORIGINAL: 01101001 10001001 00011000 10101101 11011111 00101110 11101000 00000001
TEXTO CAMBIADO: 01111001 10001001 00111100 10100100 10101100 00101000 11100110 00111000
CAMBIAN 19 BITS.
#####

##### RONDA 4 #####
TEXTO ORIGINAL: 11011111 00101110 11101000 00000001 01111011 11011110 11010010 10010100
TEXTO CAMBIADO: 10101100 00101000 11100110 00111000 10011001 00011011 01011010 01001000
CAMBIAN 29 BITS.
#####
```

```
#####
##### RONDA 5 #####
TEXTO ORIGINAL: 01111011 11011110 11010010 10010100 00111010 00000001 01001010 00100101
TEXTO CAMBIADO: 10011001 00011011 01011010 01001000 11000011 11101000 00100001 01101111
CAMBIAN 34 BITS.
#####

##### RONDA 6 #####
TEXTO ORIGINAL: 00111010 00000001 01001010 00100101 10110001 00001101 01101010 11011001
TEXTO CAMBIADO: 11000011 11101000 00100001 01101111 00111000 10111100 10111001 00010111
CAMBIAN 36 BITS.
#####

##### RONDA 7 #####
TEXTO ORIGINAL: 10110001 00001101 01101010 11011001 11100011 11101110 10110000 11011100
TEXTO CAMBIADO: 00111000 10111100 1011001 00010111 01011101 00000110 01010101 01000010
CAMBIAN 37 BITS.
#####

##### RONDA 8 #####
TEXTO ORIGINAL: 11100011 11101110 10110000 11011100 10110000 01011111 11010010 00111100
TEXTO CAMBIADO: 01011101 00000110 01010101 01000010 11011110 00010001 01000110 11011110
CAMBIAN 36 BITS.
#####

##### RONDA 9 #####
TEXTO ORIGINAL: 10110000 01011111 11010010 00111100 10110011 11111100 01011011 11011000
TEXTO CAMBIADO: 11011110 00010001 01000110 11011110 00110001 01110011 10110010 11011100
CAMBIAN 29 BITS.
#####

##### RONDA 10 #####
TEXTO ORIGINAL: 10110011 11111100 01011011 11011000 01111110 11001110 01001010 01011010
TEXTO CAMBIADO: 00110001 01110011 10110010 11011100 01100010 00011110 11001000 00000001
CAMBIAN 26 BITS.
#####
```



```
#####
##### Ronda 11 #####
TEXTO ORIGINAL: 01111110 11001110 01001010 01011010 01010010 01001000 11000111 01100010
TEXTO CAMBIADO: 01100010 00011110 11001000 00000001 10101101 01100101 10011001 11110001
CAMBIAN 34 BITS.
#####

##### Ronda 12 #####
TEXTO ORIGINAL: 01010010 01001000 11000111 01100010 00011101 10101111 10001010 00111101
TEXTO CAMBIADO: 10101101 01100101 10011001 11110001 01011000 10010110 00110111 10001010
CAMBIAN 40 BITS.
#####

##### Ronda 13 #####
TEXTO ORIGINAL: 00011101 10101111 10001010 00111101 00110111 11111001 01001001 11011010
TEXTO CAMBIADO: 01011000 10010110 00110111 10001010 10000011 10100001 00110001 11010011
CAMBIAN 32 BITS.
#####

##### Ronda 14 #####
TEXTO ORIGINAL: 00110111 11111001 01001001 11011010 00000110 11011010 10001001 10010000
TEXTO CAMBIADO: 10000011 10100001 00110001 11010011 01001000 00011010 00110001 00100111
CAMBIAN 29 BITS.
#####

##### Ronda 15 #####
TEXTO ORIGINAL: 00000110 11011010 10001001 10010000 11011100 01110111 10110111 01100100
TEXTO CAMBIADO: 01001000 00011010 00110001 00100111 00111101 01001011 10010101 11111110
CAMBIAN 30 BITS.
#####

##### Ronda 16 #####
TEXTO ORIGINAL: 11011100 01110111 10110111 01100100 11000010 11100111 01001101 10001000
TEXTO CAMBIADO: 00111101 01001011 10010101 11111110 00000000 01011011 10101001 10000000
CAMBIAN 27 BITS.
#####
```

AVALANCHA PRODUCIDA POR LA CLAVE

```
#####
##### Ronda 1 #####
TEXTO ORIGINAL: 10110100 00001111 01010101 11111000 00101111 11011000 00111001 10010011
TEXTO CAMBIADO: 10110100 00001111 01010101 11111000 00101111 11011000 00111000 10010111
CAMBIAN 2 BITS.
#####

##### Ronda 2 #####
TEXTO ORIGINAL: 00101111 11011000 00111001 10010011 01101001 10001001 00011000 10101101
TEXTO CAMBIADO: 00101111 11011000 00111000 10010111 01110011 10111011 00010100 10100100
CAMBIAN 12 BITS.
#####

##### Ronda 3 #####
TEXTO ORIGINAL: 01101001 10001001 00011000 10101101 11011111 00101110 11101000 00000001
TEXTO CAMBIADO: 01110011 10111011 00010100 10100100 01001100 11001001 01010111 00111010
CAMBIAN 32 BITS.
#####

##### Ronda 4 #####
TEXTO ORIGINAL: 11011111 00101110 11101000 00000001 01111011 11011110 11010010 10010100
TEXTO CAMBIADO: 01001100 11001001 01010111 00111010 11011000 00001000 00010100 00101001
CAMBIAN 41 BITS.
#####

##### Ronda 5 #####
TEXTO ORIGINAL: 01111011 11011110 11010010 10010100 00111010 00000001 01001010 00100101
TEXTO CAMBIADO: 11011000 00001000 00010100 00101001 01101111 00110000 10110001 00011110
CAMBIAN 38 BITS.
#####

##### Ronda 6 #####
TEXTO ORIGINAL: 00111010 00000001 01001010 00100101 10110001 00001101 01101010 11011001
TEXTO CAMBIADO: 01101111 00110000 10110001 00011110 11011011 01001010 10010110 10111011
CAMBIAN 36 BITS.
#####
```

```
#####
##### RONDA 7 #####
TEXTO ORIGINAL: 10110001 00001101 01101010 11011001 11100011 11101110 10110000 11011100
TEXTO CAMBIADO: 11011011 01001010 10010110 10111011 01101010 11101101 10001110 11100111
CAMBIAN 32 BITS.
#####

##### RONDA 8 #####
TEXTO ORIGINAL: 11100011 11101110 10110000 11011100 10110000 01011111 11010010 00111100
TEXTO CAMBIADO: 01101010 11101101 10001110 11100111 01111000 10100111 10000001 00010111
CAMBIAN 31 BITS.
#####

##### RONDA 9 #####
TEXTO ORIGINAL: 10110000 01011111 11010010 00111100 10110011 11111100 01011011 11011000
TEXTO CAMBIADO: 01111000 10100111 10000001 00010111 11010000 11010101 01111100 01101010
CAMBIAN 31 BITS.
#####

##### RONDA 10 #####
TEXTO ORIGINAL: 10110011 11111100 01011011 11011000 01111110 11001110 01001010 01011010
TEXTO CAMBIADO: 11010000 11010101 01111100 01101010 01110101 00100011 10001010 11011000
CAMBIAN 28 BITS.
#####

##### RONDA 11 #####
TEXTO ORIGINAL: 01111110 11001110 01001010 01011010 01010010 01001000 11000111 01100010
TEXTO CAMBIADO: 01110101 00100011 10001010 11011000 01101100 10010010 11000110 10010101
CAMBIAN 31 BITS.
#####

##### RONDA 12 #####
TEXTO ORIGINAL: 01010010 01001000 11000111 01100010 00011101 10101111 10001010 00111101
TEXTO CAMBIADO: 01101100 10010010 11000110 11000101 01000000 01001000 01011110 10011111
CAMBIAN 36 BITS.
#####
```

```
#####
##### RONDA 13 #####
TEXTO ORIGINAL: 00011101 10101111 10001010 00111101 00110111 11111001 01001001 11011010
TEXTO CAMBIADO: 01000000 01001000 01011110 10011111 00100111 11010110 11000101 01100000
CAMBIAN 32 BITS.
#####

##### RONDA 14 #####
TEXTO ORIGINAL: 00110111 11111001 01001001 11011010 00000110 11011010 10001001 10010000
TEXTO CAMBIADO: 00100111 11010110 11000101 01100000 10111100 10001001 00000011 11011111
CAMBIAN 31 BITS.
#####

##### RONDA 15 #####
TEXTO ORIGINAL: 00000110 11011010 10001001 10010000 11011100 01110111 10110111 01100100
TEXTO CAMBIADO: 10111100 10001001 00000011 11011111 00110010 10010001 01101011 01000101
CAMBIAN 35 BITS.
#####

##### RONDA 16 #####
TEXTO ORIGINAL: 11011100 01110111 10110111 01100100 11000010 11100111 01001101 10001000
TEXTO CAMBIADO: 00110010 10010001 01101011 01000101 01100010 01111100 00100100 00110110
CAMBIAN 35 BITS.
#####
```

Como puede verse en las imágenes de la ejecución, el cambio de un solo bit del bloque original y de la clave, apenas cambia uno o dos bits en una sola ronda, pero el hecho de existan varias rondas (16) genera más cambios sobre los cambios anteriores, en avalancha como se le conoce a este efecto. Se puede observar también que llega un momento que el número de bits diferentes se estabiliza entre 25-40 bits más o menos. Esto se debe a que si el tamaño del bloque con el que trabajar en DES es de 64 bits, se quedará en torno a la mitad de bits cambiados por las coincidencias que pueden existir por los cambios del bloque anterior.

En definitiva, un cambio en un solo bit puede generar varios cambios gracias a las rondas, en cada ronda se va arrastrando el cambio de la ronda anterior en avalancha.

APARTADO 4. Principios de diseño del AES.

Apartado A. Estudio de la no linealidad de las S-boxes del AES.

Para este apartado, hemos usado un código bastante parecido al del apartado 3a ya que nos hemos centrado en el mismo tipo de prueba a realizar. Para estas pruebas, al igual que en las del 3a, hemos creado todas las posibles entradas para la S-box directa e inversa del AES y hemos comprobado uno a uno cada caso para ver si se cumple la no linealidad descrita en el apartado 3a.

Aunque hemos usado casi el código idéntico anterior y el mismo principio, si hemos tenido que implementar la funcionalidad adecuada para acceder al contenido de las S-boxes del AES. Esto no ha tenido mucha complicación porque tan solo había que coger los 4 bits de más peso para saber la fila y los 4 bits de menos peso para conocer la columna a la que acceder a la S-box del AES correspondiente. Además, para el caso del AES hemos tenido que probar con muchos más casos que el DES, porque el AES tiene 8 bits de entrada a la S-box, los 4 de la fila y los 4 de la columna que se acaban de explicar. En definitiva, sacamos la fila, sacamos la columna, accedemos al contenido de 8 bits de la S-box correspondiente para dos bloques y comprobamos si se cumple o no la no linealidad. Todo esto está implementado en el fichero “no_linealidad_SBoxes_AES.c”.

Para ejecutar el programa que prueba la no linealidad de las S-boxes del AES se tiene que introducir por teclado “make apartado_4a_1” o “./no_linealidad_SBoxes_AES -D” para las pruebas para la S-box directa o “make apartado_4a_2” o “./no_linealidad_SBoxes_AES -I” para las pruebas de la S-box inversa. A continuación mostramos los resultados para la directa y para la inversa respectivamente:

```
(x) + f(y) = 111
#####
#####
CASO LINEAL:
x = 244
y = 248
f(x) = 191
f(y) = 65
f(x + y) = 254
f(x) + f(y) = 254
#####

#####
CASOS NO LINEALES: 32770
CASOS LINEALES: 126
CASOS TOTALES: 32896
PORCENTAJE NO LINEALIDAD: 99.616975
#####
```

```
#####
CASO LINEAL:
x = 230
y = 245
f(x) = 245
f(y) = 119
f(x + y) = 130
f(x) + f(y) = 130
#####

#####
CASOS NO LINEALES: 32770
CASOS LINEALES: 126
CASOS TOTALES: 32896
PORCENTAJE NO LINEALIDAD: 99.616975
#####
```

De nuevo se muestran los casos lineales para poder ver (debugear) que sí existen dichos casos lineales. Como puede verse en ambos casos, tanto para la directa como para la inversa se obtiene el mismo porcentaje de no linealidad de 99,61 %, es decir, de 32896 casos posibles, 32770 no son lineales y los 126 restantes sí cumplen la linealidad. Este porcentaje es prácticamente del 100 % lo que demuestra que AES es un algoritmo muy seguro, incluso más que el DES que también tiene un alto porcentaje de no linealidad en las s-boxes del 95,36 % como vimos anteriormente. La no linealidad de las s-boxes del AES cumpliendo este porcentaje tan alto de no linealidad son una de las bases para que el AES tenga una alta seguridad superando al DES y sustituyendo al mismo en la actualidad, además de que el AES es más rápido que el DES a la hora de cifrar y descifrar.

Apartado B. Generación de las S-boxes AES.

Para la ejecución de este ejercicio podemos hacer “make apartado_4b” esto ejecuta la creación de la s-box directa de AES, la escribe por defecto en el fichero Tabla_AEs/sbox_aes_directa.txt.

Para ejecutar la s-box inversa podemos escribir en la terminal ./sbox_AES -D que guarda la tabla en Tabla_AEs/sbox_aes_inversa.txt.

Los valores de la s-box directa se obtienen:

$$s = b \text{ xor } (b \ll 1) \text{ xor } (b \ll 2) \text{ xor } (b \ll 3) \text{ xor } (b \ll 4) \text{ xor } 0x63$$

Siendo b el inverso multiplicativo de valores de 0 a 255, y \ll es el desplazamiento circular a la izquierda.

Para hallar el inverso multiplicativo de b tenemos que implementar el algoritmo de euclides adaptado para polinomios, para comprobar si b es coprimo con $m(x)$ de $GF(2^8)$. Y el algoritmo de euclides extendido que nos devuelve el inverso multiplicativo.

También, tenemos que implementar la función división, resto y multiplicación adaptada para polinomios, las podemos encontrar en operaciones_bit.c

El valor $s[0][0]$ lo insertamos directamente.


```

P2 > F-CSI_p2_g03 > g03 > Tablas_AES > sbox_aes_directa.txt
1 63 7C 77 7B F2 6B 6F C5 30 01 67 2B FE D7 AB 76
2 CA 82 C9 7D FA 59 47 F0 AD D4 A2 AF 9C A4 72 C0
3 B7 FD 93 26 36 3F F7 CC 34 A5 E5 F1 71 D8 31 15
4 04 C7 23 C3 18 96 05 9A 07 12 80 E2 EB 27 B2 75
5 09 83 2C 1A 1B 6E 5A A0 52 3B D6 B3 29 E3 2F 84
6 53 D1 00 ED 20 FC B1 5B 6A CB BE 39 4A 4C 58 CF
7 D0 EF AA FB 43 4D 33 85 45 F9 02 7F 50 3C 9F A8
8 51 A3 40 8F 92 9D 38 F5 BC B6 DA 21 10 FF F3 D2
9 CD 0C 13 EC 5F 97 44 17 C4 A7 7E 3D 64 5D 19 73
10 60 81 4F DC 22 2A 90 88 46 EE B8 14 DE 5E 0B DB
11 E0 32 3A 0A 49 06 24 5C C2 D3 AC 62 91 95 E4 79
12 E7 C8 37 6D 8D D5 4E A9 6C 56 F4 EA 65 7A AE 08
13 BA 78 25 2E 1C A6 B4 C6 E8 DD 74 1F 4B BD 8B 8A
14 70 3E B5 66 48 03 F6 0E 61 35 57 B9 86 C1 1D 9E
15 E1 F8 98 11 69 D9 8E 94 9B 1E 87 E9 CE 55 28 DF
16 8C A1 89 0D BF E6 42 68 41 99 2D 0F B0 54 BB 16
17

```

Los valores de la s-box inversa se obtienen:

$$b = s \text{ xor } (s \ll 1) \text{ xor } (s \ll 3) \text{ xor } (s \ll 6) \text{ xor } 0x05$$

Siendo s valores de 0 a 255, y b el inverso multiplicativo del resultado de todas las xor.

```

P2 > F-CSI_p2_g03 > g03 > Tablas_AES > sbox_aes_inversa.txt
1 52 09 6A D5 30 36 A5 38 BF 40 A3 9E 81 F3 D7 FB
2 7C E3 39 82 9B 2F FF 87 34 8E 43 44 C4 DE E9 CB
3 54 7B 94 32 A6 C2 23 3D EE 4C 95 0B 42 FA C3 4E
4 08 2E A1 66 28 D9 24 B2 76 5B A2 49 6D 8B D1 25
5 72 F8 F6 64 86 68 98 16 D4 A4 5C CC 5D 65 B6 92
6 6C 70 48 50 FD ED B9 DA 5E 15 46 57 A7 8D 9D 84
7 90 D8 AB 00 8C BC D3 0A F7 E4 58 05 B8 B3 45 06
8 D0 2C 1E 8F CA 3F 0F 02 C1 AF BD 03 01 13 8A 6B
9 3A 91 11 41 4F 67 DC EA 97 F2 CF CE F0 B4 E6 73
10 96 AC 74 22 E7 AD 35 85 E2 F9 37 E8 1C 75 DF 6E
11 47 F1 1A 71 1D 29 C5 89 6F B7 62 0E AA 18 BE 1B
12 FC 56 3E 4B C6 D2 79 20 9A DB C0 FE 78 CD 5A F4
13 1F DD A8 33 88 07 C7 31 B1 12 10 59 27 80 EC 5F
14 60 51 7F A9 19 B5 4A 0D 2D E5 7A 9F 93 C9 9C EF
15 A0 E0 3B 4D AE 2A F5 B0 C8 EB BB 3C 83 53 99 61
16 17 2B 04 7E BA 77 D6 26 E1 69 14 63 55 21 0C 7D
17

```