

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Práctica 2: Seguridad Perfecta y Criptografía Simétrica

Antes de todo, se incluyen en la misma carpeta **g05/** del comprimido todos los programas pedidos en esta práctica, con sus .c y .h correspondientes. Además, se incluye esta memoria y un **Makefile** que habrá que ejecutar antes de nada para obtener los ejecutables de cada apartado con: > **make**

1. Seguridad Perfecta

A. Comprobación empírica de la Seguridad Perfecta del cifrado por desplazamiento

En este primer apartado tenemos que estudiar si el Cifrado por Desplazamiento posee Seguridad Perfecta si las claves son elegidas con igual probabilidad, es decir, que sean equiprobables y, al contrario, que no consiga Seguridad Perfecta si no se crean las claves de forma equiprobable.

Por lo tanto, hemos creado **seg_perf.c**, que corresponde al main de este apartado, y su librería **seg_perf.h**. El programa ejecutable posee la siguiente entrada por argumentos:

> **./seg_perf {-P|-I} [-i fichero_entrada.txt] [-o fichero_salida.txt]**

donde corresponde a cifrar el fichero de entrada y volcarlo en el fichero de salida, en el caso de que se indiquen al ser opcionales, o cifrar la cadena introducida por teclado, si no se han indicado los ficheros. Y, el argumento obligatorio, -P indica que se generan las claves de manera equiprobable o -I indica que se generan de forma no equiprobable.

Antes de nada, de forma teórica que un criptosistema posea **Seguridad Perfecta** quiere decir que un atacante no puede obtener información clave del mensaje plano con el texto cifrado en su posesión. Por lo tanto, la probabilidad de un carácter plano del alfabeto, $P_p(x)$, tiene que ser igual a la probabilidad condicionada del carácter plano del alfabeto en conocimiento del carácter cifrado del alfabeto, $P_p(x|y)$, es decir, $P_p(x) = P_p(x|y)$ para todo x y y pertenecientes a Z_{26} (en nuestro caso el alfabeto ascii inglés).

Entonces, sabiendo esto, tenemos que calcular en ejecución, mientras se cifra el texto plano introducido en el programa, todas las probabilidades necesarias para hallar la segunda probabilidad mencionada, además de la primera que es trivial.

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Para esto, cuando el usuario elija una de las opciones, si quiere claves equiprobables se utilizará el método “cifrar_equiprobable()”, y si no, se utilizará el método “cifrar_no_equiprobable()”, y en cada una de ellas, donde se cifra el mensaje plano con el cifrado por desplazamiento, se realizarán los conteos de algunas de las frecuencias necesarias para el cálculo de todas las probabilidades.

La primera probabilidad ya la conocemos y corresponde a la frecuencia de un carácter plano en el alfabeto inglés. Sin embargo, esta probabilidad **P_{p(x)}** la vamos a obtener contando frecuencias del texto plano introducido y dividiendo por la longitud total del texto, obteniendo sus probabilidades (en el lenguaje inglés las probabilidades de los caracteres no son iguales, pero no es un requisito para que se cumpla la Seguridad Perfecta). Estas probabilidades se almacenan en el array “p_px” tras calcularlas en la función “calcula_p_caracter_plano_y_cifrado()” de la manera ya indicada.

Tras esto, necesitamos la probabilidad del carácter cifrado **P_{c(y)}** que sabemos que se obtiene mediante el sumatorio de las probabilidades conjuntas de P_{k(k)} y P_{p(x)} que llevan al mismo carácter cifrado y, es decir, **P_{c(y)} = Σ P_{k(k)}*P_{p(x)}**. La probabilidad del carácter plano ya la tenemos, además de la probabilidad de la clave k empleada. Esta segunda se obtiene en “cifrar_equiprobable()” o en “cifrar_no_equiprobable()” contando frecuencias cada vez que se aplica una clave generada de la forma indicada, y dividiendo después por el tamaño del mensaje para hallar la probabilidad correspondiente y almacenada en el array “p_k”. Sin embargo, este cálculo equivale a contar en el texto cifrado resultante las frecuencias de los caracteres del alfabeto, por lo que realizamos este mismo conteo en la misma función “calcula_p_caracter_plano_y_cifrado()” que para P_{p(x)} pero sobre el texto cifrado, y almacenamos las probabilidades obtenidas en el array “p_cy”.

Ahora, hay que hallar las probabilidades condicionadas de los caracteres cifrados en función del carácter plano conocido, **P_{c(y|x)}**, que es lo mismo que el sumatorio de las probabilidades de las claves utilizadas para obtener el carácter cifrado mediante ese carácter plano cumpliendo y = x + k (en teoría solo una clave puede cumplir esto en el cifrado por desplazamiento), es decir, **P_{c(y|x)} = Σ P_{k(k)}**. Para lograr esto, en la función “calcula_p_caracter_cifrado_condicionado()” se recorren ambos mensaje cifrado y plano de manera paralela, obteniendo la clave empleada al cifrar y almacenando la probabilidad de esa clave en la matriz “p_cyx” (una matriz de 26*26 con las P_{c(y|x)}).

Y, con todo esto, ya podemos obtener la probabilidad buscada desde el principio, **P_{p(x|y)}**, que cumple la siguiente relación mediante el Teorema de Bayes:

P_{p(x|y)} = P_{p(x)*P_{c(y|x)} / P_{c(y)}}. Por lo que ahora solo hay que sustituir cada probabilidad con la que le corresponda para cada carácter plano x y para cada carácter cifrado y. Estas probabilidades condicionadas se almacenan en el array

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

“ p_{pxy} ” y se calculan en un simple bucle final para cada valor de x plano y de y cifrado, obteniendo las probabilidades de las demás arrays anteriores. Entonces, ya solo queda comprobar si la probabilidad $P(p(x|y))$ obtenida es igual (aproximado dentro de un rango de 0.05 al ser frecuencias aleatorias). Se notifica por pantalla con un mensaje para cada probabilidad si cumple la relación de Seguridad Perfecta o no.

Ahora, en cuanto a la **generación de las claves**, cuando se piden claves equiprobables el método “cifrar_equiprobable()” genera una clave mediante un rand() (que genera un número aleatorio de manera equiprobable al ser totalmente aleatorio supuestamente) dentro de Z26. Por el contrario, en “cifrar_no_equiprobable()” se generan las claves de igual forma aleatoria con rand() pero creamos una serie de condiciones para que las claves 0, 1 y 2 tengan una probabilidad no equiprobable con las demás (de 3 a 25). Y simplemente cada clave generada se aplica de igual manera en cada iteración del cifrado por desplazamiento. Se pueden ver estas generaciones de claves en las siguientes imágenes:

```
326
327 void cifrar_equiprobable(char *texto, double *p_k){
328     int cont = 0, valor_caracter, valor_clave, aux;
329     int valor_nuevo_caracter, long_texto, i, cont_caracteres = 0;
330
331     long_texto = strlen(texto);
332
333     while (long_texto != 0){
334
335         if (texto[cont] >= 65 && texto[cont] <= 90){
336             /* 65 - 90 */
337             cont_caracteres++;
338             valor_clave = rand() % 26;
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
366 void cifrar_no_equiprobable(char *texto, double *p_k){  
367     int cont = 0, valor_caracter, valor_clave, aux;  
368     int valor_nuevo_caracter, long_texto, i, cont_caracteres = 0;  
369  
370     long_texto = strlen(texto);  
371  
372     while (long_texto != 0){  
373  
374         if (texto[cont] >= 65 && texto[cont] <= 90){  
375             /* 65 - 90 */  
376             cont_caracteres++;  
377             valor_clave = rand() % 100;  
378  
379             if (valor_clave >= 0 && valor_clave < 50) {  
380                 valor_clave = 0;  
381             }  
382             else if (valor_clave >= 50 && valor_clave < 80) {  
383                 valor_clave = 1;  
384             }  
385             else if (valor_clave >= 80 && valor_clave < 90) {  
386                 valor_clave = 2;  
387             }  
388             else{  
389                 valor_clave = rand() % (25-3+1) + 3; /*Generamos un valor entre 3 y 25 */  
390             }  
391         }
```

Con todo explicado, vamos a mostrar una serie de ejecuciones para que vea cómo es el programa y lo que muestra por pantalla, además de su correcto funcionamiento.

Primero, cifrar un texto medianamente largo (Don Quijote) con claves equiprobables:

```
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/P2  
$ ./seg_perf -P -i don_quixote.txt -o uwu.txt  
# Ha elegido equiprobabilidad en las claves #  
  
Cifrando con el metodo de desplazamiento el fichero de entrada: don_quixote.txt  
Guardamos el texto cifrado en el fichero de salida: uwu.txt  
##### P_p(x) #####  
P_p(A) = 0.088780  
P_p(B) = 0.012296  
P_p(C) = 0.024738  
P_p(D) = 0.044719  
P_p(E) = 0.114470  
P_p(F) = 0.023640  
P_p(G) = 0.020420  
P_p(H) = 0.072751  
P_p(I) = 0.069604  
P_p(J) = 0.000586  
P_p(K) = 0.008124  
P_p(L) = 0.037547  
P_p(M) = 0.024446  
P_p(N) = 0.068287  
P_p(O) = 0.082778  
P_p(P) = 0.013540  
P_p(Q) = 0.002196  
P_p(R) = 0.054454  
P_p(S) = 0.067335  
P_p(T) = 0.094123  
P_p(U) = 0.025397  
P_p(V) = 0.009588  
P_p(W) = 0.022982  
P_p(X) = 0.002269  
P_p(Y) = 0.014199  
P_p(Z) = 0.000732
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
##### P_k(k) #####
P_k(A) = 0.036376
P_k(B) = 0.035936
P_k(C) = 0.038718
P_k(D) = 0.038571
P_k(E) = 0.039157
P_k(F) = 0.038937
P_k(G) = 0.036522
P_k(H) = 0.041499
P_k(I) = 0.037620
P_k(J) = 0.038791
P_k(K) = 0.039010
P_k(L) = 0.037913
P_k(M) = 0.038791
P_k(N) = 0.035205
P_k(O) = 0.038718
P_k(P) = 0.040108
P_k(Q) = 0.040328
P_k(R) = 0.038937
P_k(S) = 0.037986
P_k(T) = 0.040328
P_k(U) = 0.037108
P_k(V) = 0.038205
P_k(W) = 0.040621
P_k(X) = 0.036742
P_k(Y) = 0.036595
P_k(Z) = 0.041279
```

```
##### P_c(y) #####
P_c(A) = 0.039157
P_c(B) = 0.037473
P_c(C) = 0.039962
P_c(D) = 0.038937
P_c(E) = 0.038645
P_c(F) = 0.036742
P_c(G) = 0.035131
P_c(H) = 0.037766
P_c(I) = 0.038132
P_c(J) = 0.039230
P_c(K) = 0.038571
P_c(L) = 0.039523
P_c(M) = 0.039669
P_c(N) = 0.039669
P_c(O) = 0.035790
P_c(P) = 0.036449
P_c(Q) = 0.036888
P_c(R) = 0.039010
P_c(S) = 0.037986
P_c(T) = 0.041279
P_c(U) = 0.039742
P_c(V) = 0.039523
P_c(W) = 0.039889
P_c(X) = 0.039450
P_c(Y) = 0.037181
P_c(Z) = 0.038205
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
##### P_c(y|x) #####
P_c(A|A) = 0.036376
P_c(A|B) = 0.041279
P_c(A|C) = 0.036595
P_c(A|D) = 0.036742
P_c(A|E) = 0.040621
P_c(A|F) = 0.038205
P_c(A|G) = 0.037108
P_c(A|H) = 0.040328
P_c(A|I) = 0.037986
P_c(A|J) = 0.038937
P_c(A|K) = 0.040328
P_c(A|L) = 0.040108
P_c(A|M) = 0.038718
P_c(A|N) = 0.035205
P_c(A|O) = 0.038791
P_c(A|P) = 0.037913
P_c(A|Q) = 0.000000
P_c(A|R) = 0.038791
P_c(A|S) = 0.037620
P_c(A|T) = 0.041499
P_c(A|U) = 0.036522
P_c(A|V) = 0.038937
P_c(A|W) = 0.039157
```

```
P_c(Z|S) = 0.041499
P_c(Z|T) = 0.036522
P_c(Z|U) = 0.038937
P_c(Z|V) = 0.039157
P_c(Z|W) = 0.038571
P_c(Z|X) = 0.000000
P_c(Z|Y) = 0.035936
P_c(Z|Z) = 0.000000
```

```
##### P_p(x|y) #####
P_p(A|A) = P_c(A|A) * P_p(A) / P_c(A) = 0.082474
Cumple seguridad perfecta P_p(A|A) ~= P_p(A)
P_p(B|A) = P_c(A|B) * P_p(B) / P_c(A) = 0.012962
Cumple seguridad perfecta P_p(B|A) ~= P_p(B)
P_p(C|A) = P_c(A|C) * P_p(C) / P_c(A) = 0.023120
Cumple seguridad perfecta P_p(C|A) ~= P_p(C)
P_p(D|A) = P_c(A|D) * P_p(D) / P_c(A) = 0.041961
Cumple seguridad perfecta P_p(D|A) ~= P_p(D)
P_p(E|A) = P_c(A|E) * P_p(E) / P_c(A) = 0.118749
Cumple seguridad perfecta P_p(E|A) ~= P_p(E)
P_p(F|A) = P_c(A|F) * P_p(F) / P_c(A) = 0.023066
Cumple seguridad perfecta P_p(F|A) ~= P_p(F)
P_p(G|A) = P_c(A|G) * P_p(G) / P_c(A) = 0.019351
Cumple seguridad perfecta P_p(G|A) ~= P_p(G)
P_p(H|A) = P_c(A|H) * P_p(H) / P_c(A) = 0.074927
Cumple seguridad perfecta P_p(H|A) ~= P_p(H)
P_p(I|A) = P_c(A|I) * P_p(I) / P_c(A) = 0.067522
Cumple seguridad perfecta P_p(I|A) ~= P_p(I)
P_p(J|A) = P_c(A|J) * P_p(J) / P_c(A) = 0.000582
Cumple seguridad perfecta P_p(J|A) ~= P_p(J)
P_p(K|A) = P_c(A|K) * P_p(K) / P_c(A) = 0.008367
Cumple seguridad perfecta P_p(K|A) ~= P_p(K)
P_p(L|A) = P_c(A|L) * P_p(L) / P_c(A) = 0.038459
Cumple seguridad perfecta P_p(L|A) ~= P_p(L)
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
P_p(M|Z) = P_c(Z|M) * P_p(M) / P_c(Z) = 0.022526
Cumple seguridad perfecta P_p(M|Z) ~= P_p(M)
P_p(N|Z) = P_c(Z|N) * P_p(N) / P_c(Z) = 0.069333
Cumple seguridad perfecta P_p(N|Z) ~= P_p(N)
P_p(O|Z) = P_c(Z|O) * P_p(O) / P_c(Z) = 0.082144
Cumple seguridad perfecta P_p(O|Z) ~= P_p(O)
P_p(P|Z) = P_c(Z|P) * P_p(P) / P_c(Z) = 0.013826
Cumple seguridad perfecta P_p(P|Z) ~= P_p(P)
P_p(Q|Z) = P_c(Z|Q) * P_p(Q) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(Q|Z) ~= P_p(Q)
P_p(R|Z) = P_c(Z|R) * P_p(R) / P_c(Z) = 0.053619
Cumple seguridad perfecta P_p(R|Z) ~= P_p(R)
P_p(S|Z) = P_c(Z|S) * P_p(S) / P_c(Z) = 0.073140
Cumple seguridad perfecta P_p(S|Z) ~= P_p(S)
P_p(T|Z) = P_c(Z|T) * P_p(T) / P_c(Z) = 0.089976
Cumple seguridad perfecta P_p(T|Z) ~= P_p(T)
P_p(U|Z) = P_c(Z|U) * P_p(U) / P_c(Z) = 0.025884
Cumple seguridad perfecta P_p(U|Z) ~= P_p(U)
P_p(V|Z) = P_c(Z|V) * P_p(V) / P_c(Z) = 0.009827
Cumple seguridad perfecta P_p(V|Z) ~= P_p(V)
P_p(W|Z) = P_c(Z|W) * P_p(W) / P_c(Z) = 0.023202
Cumple seguridad perfecta P_p(W|Z) ~= P_p(W)
P_p(X|Z) = P_c(Z|X) * P_p(X) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(X|Z) ~= P_p(X)
P_p(Y|Z) = P_c(Z|Y) * P_p(Y) / P_c(Z) = 0.013356
Cumple seguridad perfecta P_p(Y|Z) ~= P_p(Y)
P_p(Z|Z) = P_c(Z|Z) * P_p(Z) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(Z|Z) ~= P_p(Z)
```

El criptosistema elegido si tiene seguridad perfecta!

X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/P2

Podemos observar que sí cumple Seguridad Perfecta un texto largo como Don Quijote cuando las claves han sido generadas de manera equiprobable. En la captura de $P_k(k)$ puede ver claramente cómo las probabilidades de las claves generadas con aproximadamente las mismas.

A continuación, el mismo texto plano pero con las claves generadas de manera no equiprobable:

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/P2
$ ./seg_perf -I -i don_quixote.txt -o uwu.txt
# Ha elegido no equiprobabilidad en las claves #

Cifrando con el metodo de desplazamiento el fichero de entrada: don_quixote.txt
Guardamos el texto cifrado en el fichero de salida: uwu.txt
##### P_p(x) #####
P_p(A) = 0.088780
P_p(B) = 0.012296
P_p(C) = 0.024738
P_p(D) = 0.044719
P_p(E) = 0.114470
P_p(F) = 0.023640
P_p(G) = 0.020420
P_p(H) = 0.072751
P_p(I) = 0.069604
P_p(J) = 0.000586
P_p(K) = 0.008124
P_p(L) = 0.037547
P_p(M) = 0.024446
P_p(N) = 0.068287
P_p(O) = 0.082778
P_p(P) = 0.013540
P_p(Q) = 0.002196
P_p(R) = 0.054454
P_p(S) = 0.067335
P_p(T) = 0.094123
P_p(U) = 0.025397
P_p(V) = 0.009588
P_p(W) = 0.022982
P_p(X) = 0.002269
P_p(Y) = 0.014199
P_p(Z) = 0.000732
```

```
##### P_k(k) #####
P_k(A) = 0.497621
P_k(B) = 0.298836
P_k(C) = 0.101076
P_k(D) = 0.005123
P_k(E) = 0.004904
P_k(F) = 0.004465
P_k(G) = 0.003952
P_k(H) = 0.005050
P_k(I) = 0.003806
P_k(J) = 0.004391
P_k(K) = 0.004391
P_k(L) = 0.004611
P_k(M) = 0.004465
P_k(N) = 0.004391
P_k(O) = 0.004684
P_k(P) = 0.004538
P_k(Q) = 0.004025
P_k(R) = 0.004391
P_k(S) = 0.004318
P_k(T) = 0.003367
P_k(U) = 0.005123
P_k(V) = 0.005270
P_k(W) = 0.003733
P_k(X) = 0.004465
P_k(Y) = 0.004465
P_k(Z) = 0.004538
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
#####
# P_c(y) #####
P_c(A) = 0.051014
P_c(B) = 0.036010
P_c(C) = 0.029715
P_c(D) = 0.037034
P_c(E) = 0.077728
P_c(F) = 0.051014
P_c(G) = 0.032570
P_c(H) = 0.050282
P_c(I) = 0.059943
P_c(J) = 0.034253
P_c(K) = 0.015004
P_c(L) = 0.025031
P_c(M) = 0.027007
P_c(N) = 0.050501
P_c(O) = 0.068799
P_c(P) = 0.040987
P_c(Q) = 0.018737
P_c(R) = 0.030520
P_c(S) = 0.055332
P_c(T) = 0.075240
P_c(U) = 0.051087
P_c(V) = 0.026129
P_c(W) = 0.020713
P_c(X) = 0.012662
P_c(Y) = 0.013540
P_c(Z) = 0.009149
```

```
#####
# P_c(y|x) #####
P_c(A|A) = 0.497621
P_c(A|B) = 0.000000
P_c(A|C) = 0.004465
P_c(A|D) = 0.004465
P_c(A|E) = 0.003733
P_c(A|F) = 0.005270
P_c(A|G) = 0.000000
P_c(A|H) = 0.003367
P_c(A|I) = 0.004318
P_c(A|J) = 0.000000
P_c(A|K) = 0.000000
P_c(A|L) = 0.004538
P_c(A|M) = 0.004684
P_c(A|N) = 0.004391
P_c(A|O) = 0.004465
P_c(A|P) = 0.004611
P_c(A|Q) = 0.000000
P_c(A|R) = 0.004391
P_c(A|S) = 0.003806
P_c(A|T) = 0.005050

P_c(Z|Q) = 0.004391
P_c(Z|R) = 0.003806
P_c(Z|S) = 0.005050
P_c(Z|T) = 0.003952
P_c(Z|U) = 0.000000
P_c(Z|V) = 0.000000
P_c(Z|W) = 0.005123
P_c(Z|X) = 0.101076
P_c(Z|Y) = 0.298836
P_c(Z|Z) = 0.497621
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
#####
P_p(A|A) = P_c(A|A) * P_p(A) / P_c(A) = 0.866018
No cumple la seguridad perfecta P_p(A|A) != P_p(A)
P_p(B|A) = P_c(A|B) * P_p(B) / P_c(A) = 0.000000
Cumple seguridad perfecta P_p(B|A) ~= P_p(B)
P_p(C|A) = P_c(A|C) * P_p(C) / P_c(A) = 0.002165
Cumple seguridad perfecta P_p(C|A) ~= P_p(C)
P_p(D|A) = P_c(A|D) * P_p(D) / P_c(A) = 0.003914
Cumple seguridad perfecta P_p(D|A) ~= P_p(D)
P_p(E|A) = P_c(A|E) * P_p(E) / P_c(A) = 0.008376
No cumple la seguridad perfecta P_p(E|A) != P_p(E)
P_p(F|A) = P_c(A|F) * P_p(F) / P_c(A) = 0.002442
Cumple seguridad perfecta P_p(F|A) ~= P_p(F)
P_p(G|A) = P_c(A|G) * P_p(G) / P_c(A) = 0.000000
Cumple seguridad perfecta P_p(G|A) ~= P_p(G)
P_p(H|A) = P_c(A|H) * P_p(H) / P_c(A) = 0.004801
No cumple la seguridad perfecta P_p(H|A) != P_p(H)
P_p(I|A) = P_c(A|I) * P_p(I) / P_c(A) = 0.005892
No cumple la seguridad perfecta P_p(I|A) != P_p(I)
P_p(J|A) = P_c(A|J) * P_p(J) / P_c(A) = 0.000000
Cumple seguridad perfecta P_p(J|A) ~= P_p(J)
P_p(K|A) = P_c(A|K) * P_p(K) / P_c(A) = 0.000000
Cumple seguridad perfecta P_p(K|A) ~= P_p(K)
P_p(L|A) = P_c(A|L) * P_p(L) / P_c(A) = 0.003340
Cumple seguridad perfecta P_p(L|A) ~= P_p(L)
P_p(M|A) = P_c(A|M) * P_p(M) / P_c(A) = 0.002245
Cumple seguridad perfecta P_p(M|A) ~= P_p(M)
P_p(N|A) = P_c(A|N) * P_p(N) / P_c(A) = 0.005878
No cumple la seguridad perfecta P_p(N|A) != P_p(N)
P_p(O|A) = P_c(A|O) * P_p(O) / P_c(A) = 0.007245
No cumple la seguridad perfecta P_p(O|A) != P_p(O)
P_p(P|A) = P_c(A|P) * P_p(P) / P_c(A) = 0.001224
Cumple seguridad perfecta P_p(P|A) ~= P_p(P)
P_p(Q|A) = P_c(A|Q) * P_p(Q) / P_c(A) = 0.000000
Cumple seguridad perfecta P_p(Q|A) ~= P_p(Q)
P_p(R|A) = P_c(A|R) * P_p(R) / P_c(A) = 0.004688
Cumple seguridad perfecta P_p(R|A) ~= P_p(R)
```

```
P_p(S|Z) = P_c(Z|S) * P_p(S) / P_c(Z) = 0.037169
Cumple seguridad perfecta P_p(S|Z) ~= P_p(S)
P_p(T|Z) = P_c(Z|T) * P_p(T) / P_c(Z) = 0.040661
No cumple la seguridad perfecta P_p(T|Z) != P_p(T)
P_p(U|Z) = P_c(Z|U) * P_p(U) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(U|Z) ~= P_p(U)
P_p(V|Z) = P_c(Z|V) * P_p(V) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(V|Z) ~= P_p(V)
P_p(W|Z) = P_c(Z|W) * P_p(W) / P_c(Z) = 0.012870
Cumple seguridad perfecta P_p(W|Z) ~= P_p(W)
P_p(X|Z) = P_c(Z|X) * P_p(X) / P_c(Z) = 0.025067
Cumple seguridad perfecta P_p(X|Z) ~= P_p(X)
P_p(Y|Z) = P_c(Z|Y) * P_p(Y) / P_c(Z) = 0.463794
No cumple la seguridad perfecta P_p(Y|Z) != P_p(Y)
P_p(Z|Z) = P_c(Z|Z) * P_p(Z) / P_c(Z) = 0.039810
Cumple seguridad perfecta P_p(Z|Z) ~= P_p(Z)
```

El criptosistema elegido no tiene seguridad perfecta!

```
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/P2
$
```

Se puede ver claramente que hay probabilidades condicionadas $P_c(x|y)$ que no coinciden con la probabilidad del carácter plano $P_p(x)$, como no ocurría en el ejemplo anterior, y es debido a generar claves no equiprobables. Puede ver en la

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

captura de $P_k(k)$ que las tres primeras claves (A, B, C) poseen un una probabilidad mayor a todas las demás (D a Z son realmente equiprobables entre sí por cómo hemos diseñado la generación de claves) y distintas entre sí. El mensaje final indica que hay uno o más casos donde $P_p(x)$ no coincide con $P_c(x|y)$, indicando que este criptosistema con generación de claves no equiprobables no posee Seguridad Perfecta, a diferencia del criptosistema que generaba claves equiprobables.

Y, para terminar, una ejecución de una cadena de texto pequeña introducida a cifrar con claves equiprobables. Solo mostraremos $P_p(x)$, $P_k(k)$ y el resultado final:

```
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/P2
$ ./seg_perf -P
# Ha elegido equiprobabilidad en las claves #

Introduzca lo que quiere cifrar:
HOLA ESTO ES UNA PRUEBA PARA CIFRAR CON CLAVES EQUIPROBABLES PERO UN TEXTO PLANO DE CORTA LONGITUD

Cifrando con el metodo de desplazamiento HOLA ESTO ES UNA PRUEBA PARA CIFRAR CON CLAVES EQUIPROBABLES PERO UN TEXTO PLANO DE CORTA LONGITUD
#####
P_p(x) #####
P_p(A) = 0.121951
P_p(B) = 0.036585
P_p(C) = 0.048780
P_p(D) = 0.024390
P_p(E) = 0.109756
P_p(F) = 0.012195
P_p(G) = 0.012195
P_p(H) = 0.012195
P_p(I) = 0.036585
P_p(J) = 0.000000
P_p(K) = 0.000000
P_p(L) = 0.060976
P_p(M) = 0.000000
P_p(N) = 0.060976
P_p(O) = 0.109756
P_p(P) = 0.060976
P_p(Q) = 0.012195
P_p(R) = 0.085366
P_p(S) = 0.048780
P_p(T) = 0.060976
P_p(U) = 0.060976
P_p(V) = 0.012195
P_p(W) = 0.000000
P_p(X) = 0.012195
P_p(Y) = 0.000000
P_p(Z) = 0.000000
```

```
#####
P_k(k) #####
P_k(A) = 0.012195
P_k(B) = 0.012195
P_k(C) = 0.048780
P_k(D) = 0.024390
P_k(E) = 0.048780
P_k(F) = 0.085366
P_k(G) = 0.060976
P_k(H) = 0.024390
P_k(I) = 0.024390
P_k(J) = 0.012195
P_k(K) = 0.036585
P_k(L) = 0.024390
P_k(M) = 0.036585
P_k(N) = 0.060976
P_k(O) = 0.012195
P_k(P) = 0.060976
P_k(Q) = 0.036585
P_k(R) = 0.060976
P_k(S) = 0.048780
P_k(T) = 0.024390
P_k(U) = 0.024390
P_k(V) = 0.048780
P_k(W) = 0.036585
P_k(X) = 0.048780
P_k(Y) = 0.060976
P_k(Z) = 0.024390
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
P_p(T|Z) = P_c(Z|T) * P_p(T) / P_c(Z) = 0.000000
No cumple la seguridad perfecta P_p(T|Z) != P_p(T)
P_p(U|Z) = P_c(Z|U) * P_p(U) / P_c(Z) = 0.000000
No cumple la seguridad perfecta P_p(U|Z) != P_p(U)
P_p(V|Z) = P_c(Z|V) * P_p(V) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(V|Z) ~= P_p(V)
P_p(W|Z) = P_c(Z|W) * P_p(W) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(W|Z) ~= P_p(W)
P_p(X|Z) = P_c(Z|X) * P_p(X) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(X|Z) ~= P_p(X)
P_p(Y|Z) = P_c(Z|Y) * P_p(Y) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(Y|Z) ~= P_p(Y)
P_p(Z|Z) = P_c(Z|Z) * P_p(Z) / P_c(Z) = 0.000000
Cumple seguridad perfecta P_p(Z|Z) ~= P_p(Z)

El criptosistema elegido no tiene seguridad perfecta!

X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/P2
$ -
```

Se puede ver que con un texto de longitud corta, a diferencia de Don Quijote, la probabilidad de las claves pueden llegar a diferir un poco y convertirse en no equiprobables por el simple hecho de haberse generado algunas de ellas más veces que otras. También, no aparecen todos los caracteres del alfabeto, pero esto no afecta en el cumplimiento de Seguridad Perfecta. Por lo tanto, no se llega a cumplir que $P_p(x) = P_p(x|y)$ al poseer claves equiprobables que por tener pocas y diferentes frecuencias no son al final realmente equiprobables en la práctica.

2. Implementación del DES

A. Programación del DES

En este apartado nos pide implementar el método DES de 16 rondas en el modo CBC (Cipher-block chaining). Por lo tanto, hemos creado **desCBC.c**, que corresponde al main de este apartado. Las funciones que utiliza están implementadas en el fichero **funciones_DES_CBC.c** con las cabeceras de las funciones en la librería **funciones_DES_CBC.h**, además también incluye la librería **des_tables.h** que contiene las S-boxes y las permutaciones necesarias para el DES. El programa ejecutable posee la siguiente entrada por argumentos:

```
> ./desCBC {-C|-D -k clave -iv vectorinicializacion} [-i fichero_entrada] [-o fichero_salida]
```

En el caso de introducir fichero de entrada (-i) es obligatorio introducir fichero de salida (-o). El fichero de entrada puede ser de cualquier tipo (png, gif, txt, exe, sin tipo...etc).

Tanto en la opción de cifrado -C como en la de descifrado -D, comprobamos que la clave y el vector de inicialización sean de al menos 8 caracteres, es decir, de 8 bytes que es lo mismo que 64 bits.

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

En el caso de indicar la opción -C se realiza el cifrado DES. Hay dos opciones:

Tenemos el caso en el que no se indique el fichero de entrada ni el de salida, en este caso se ejecutará el cifrado DES del bloque de texto plano del ejemplo propuesto en el enunciado, imprimiendo por pantalla cada uno de los pasos con sus resultados, imprimimos las claves generadas, la salida de cada permutación y la salida de cada ronda. Con esta opción pudimos comprobar el correcto funcionamiento de nuestro algoritmo DES comparando nuestros resultados con los de la página dada en el enunciado. Lo vemos a continuación:

Texto plano:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

Clave:

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

Cuando se elige este modo la clave y el texto plano corresponden al del ejemplo como podemos ver en la imagen.

```
#####CIFRADO/DESCIFRADO DE EJEMPLO#####
Texto plano = 00000001 00100011 01000101 01100111 10001001 10101011 11001101 11101111
Clave = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001
```

Primero comprobamos si es correcta la generación de claves, para ello primero permutamos y comprimimos la clave inicial eliminando los bits de paridad (**PC-1**) y obtenemos K +. Vemos que ambas son iguales:

K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

```
K+ = 11110000 11001100 10101010 11110101 01010110 01100111 10001111
```

A continuación, generamos las 16 claves que se utilizarán en cada ronda Feistel (más adelante explicaremos cómo se generan estas claves):

C_I = 1110000110011001010101011111

D_I = 1010101011001100111100011110

```
k1 = 11100001 10011001 01010101 11111010 10101100 11001111 00011110
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

$$C_2 = 110000110011001010101010111111$$

$$D_2 = 0101010110011001111000111101$$

$$k2 = 11000011 00110010 10101011 11110101 01011001 10011110 00111101$$

$$C_3 = 0000110011001010101011111111$$

$$D_3 = 0101011001100111100011110101$$

$$k3 = 00001100 11001010 10101111 11110101 01100110 01111000 11110101$$

$$C_4 = 0011001100101010101111111100$$

$$D_4 = 0101100110011110001111010101$$

$$k4 = 00110011 00101010 10111111 11000101 10011001 11100011 11010101$$

$$C_5 = 1100110010101010111111110000$$

$$D_5 = 0110011001111000111101010101$$

$$k5 = 11001100 10101010 11111111 00000110 01100111 10001111 01010101$$

$$C_6 = 0011001010101011111111000011$$

$$D_6 = 1001100111100011110101010101$$

$$k6 = 00110010 10101011 11111100 00111001 10011110 00111101 01010101$$

$$C_7 = 1100101010101111111100001100$$

$$D_7 = 0110011110001111010101010110$$

$$k7 = 11001010 10101111 11110000 11000110 01111000 11110101 01010110$$

$$C_8 = 001010101011111110000110011$$

$$D_8 = 1001111000111101010101011001$$

$$k8 = 00101010 10111111 11000011 00111001 11100011 11010101 01011001$$

$$C_9 = 010101010111111100001100110$$

$$D_9 = 0011110001111010101010110011$$

$$k9 = 01010101 01111111 10000110 01100011 11000111 10101010 10110011$$

$$C_{10} = 010101011111110000110011001$$

$$D_{10} = 1111000111101010101011001100$$

$$k10 = 01010101 11111110 00011001 10011111 00011110 10101010 11001100$$

$$C_{11} = 010101111111000011001100101$$

$$D_{11} = 1100011110101010101100110011$$

$$k11 = 01010111 11111000 01100110 01011100 01111010 10101011 00110011$$

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

$C_{I2} = 01011111100001100110010101$

$D_{I2} = 000111101010101010110011001111$

k₁₂ = 01011111 11100001 10011001 01010001 11101010 10101100 11001111

$C_{I3} = 011111110000110011001010101$

$D_{I3} = 011101010101011001100111100$

k₁₃ = 01111111 10000110 01100101 01010111 10101010 10110011 00111100

$C_{I4} = 1111111000011001100101010101$

$D_{I4} = 11101010101011001100111100001$

k₁₄ = 11111110 00011001 10010101 01011110 10101010 11001100 11110001

$C_{I5} = 1111100001100110010101010111$

$D_{I5} = 10101010101100110011110000111$

k₁₅ = 11111000 01100110 01010101 01111010 10101011 00110011 11000111

$C_{I6} = 1111000011001100101010101111$

$D_{I6} = 0101010101100110011110001111$

k₁₆ = 11110000 11001100 10101010 11110101 01010110 01100111 10001111

Las 16 claves generadas son correctas. Tras esto realizamos la permutación inicial IP al texto plano:

IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

IP Bloque 1 = 11001100 00000000 11001100 11111111 11110000 10101010 11110000 10101010

Obtenemos el mismo resultado. Ahora empiezan las 16 rondas Feistel, en cada ronda comprobamos que la clave sea correcta (tras hacer la PC-2) y podremos comprobar que el resultado tras las 16 rondas es correcto:

K₁ = 000110 110000 001011 101111 111111 000111 000001 110010

**Bloque ronda 1 = 11001100 00000000 11001100 11111111 11110000 10101010 11110000 10101010
Final k₁ = 00011011 00000010 11011111 11111100 01110000 01110010
L₁ = 11110000 10101010 11110000 10101010
R₁ = 11011111 01001010 01100101 01000100**

K₂ = 011110 011010 111011 011001 110110 111100 100111 100101

**Bloque ronda 2 = 11110000 10101010 11110000 10101010 11011111 01001010 01100101 01000100
Final k₂ = 01111001 10101110 11011001 11011011 11001001 11100101
L₂ = 11101111 01001010 01100101 01000100
R₂ = 11001100 00000001 01110111 00001001**

K₃ = 010101 011111 110010 001010 010000 101100 111110 011001

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
Bloque ronda 3 = 11101111 01001010 01100101 01000100 11001100 00000001 01110111 00001001
Final k3 = 01010101 11111100 10001010 01000010 11001111 10011001
L3 = 11001100 00000001 01110111 00001001
R3 = 10100010 01011100 00001011 11110100
```

$K_4 = 011100 101010 110111 010110 110110 110011 010100 011101$

```
Bloque ronda 4 = 11001100 00000001 01110111 00001001 10100010 01011100 00001011 11110100
Final k4 = 01110010 10101101 11010110 11011011 00110101 00011101
L4 = 10100010 01011100 00001011 11110100
R4 = 01110111 00100010 00000000 01000101
```

$K_5 = 011111 001110 110000 000111 110101 110101 001110 101000$

```
Bloque ronda 5 = 10100010 01011100 00001011 11110100 01110111 00100010 00000000 01000101
Final k5 = 01111100 11101100 00000111 11101011 01010011 10101000
L5 = 01110111 00100010 00000000 01000101
R5 = 10001010 01001111 10100110 00110111
```

$K_6 = 011000 111010 010100 111110 010100 000111 101100 101111$

```
Bloque ronda 6 = 01110111 00100010 00000000 01000101 10001010 01001111 10100110 00110111
Final k6 = 01100011 10100101 00111110 01010000 01111011 00101111
L6 = 10001010 01001111 10100110 00110111
R6 = 11101001 01100111 11001101 01101001
```

$K_7 = 111011 001000 010010 110111 111101 100001 100010 111100$

```
Bloque ronda 7 = 10001010 01001111 10100110 00110111 11101001 01100111 11001101 01101001
Final k7 = 11101100 10000100 10110111 11110110 00011000 10111100
L7 = 11101001 01100111 11001101 01101001
R7 = 00000110 01001010 10111010 00010000
```

$K_8 = 111101 111000 101000 111010 110000 010011 101111 111011$

```
Bloque ronda 8 = 11101001 01100111 11001101 01101001 00000110 01001010 10111010 00010000
Final k8 = 11110111 10001010 00111010 11000001 00111011 11111011
L8 = 00000110 01001010 10111010 00010000
R8 = 11010101 01101001 01001011 10010000
```

$K_9 = 111000 001101 101111 101011 111011 011110 011110 000001$

```
Bloque ronda 9 = 00000110 01001010 10111010 00010000 11010101 01101001 01001011 10010000
Final k9 = 11100000 11011011 11101011 11101101 11100111 10000001
L9 = 11010101 01101001 01001011 10010000
R9 = 00100100 01111100 11000110 01111010
```

$K_{10} = 101100 011111 001101 000111 101110 100100 011001 001111$

```
Bloque ronda 10 = 11010101 01101001 01001011 10010000 00100100 01111100 11000110 01111010
Final k10 = 10110001 11110011 01000111 10111010 01000110 01001111
L10 = 00100100 01111100 11000110 01111010
R10 = 10110111 11010101 11010111 10110010
```

$K_{11} = 001000 010101 111111 010011 110111 101101 001110 000110$

```
Bloque ronda 11 = 00100100 01111100 11000110 01111010 10110111 11010101 11010111 10110010
Final k11 = 00100001 01011111 11010011 11011110 11010011 10000110
L11 = 10110111 11010101 11010111 10110010
R11 = 11000101 01110000 00111100 01111000
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

$K_{I2} = 011101 010111 000111 110101 100101 000110 011111 101001$

```
Bloque ronda 12 = 10110111 11010101 11010111 10110010 11000101 01111000 00111100 01111000
Final k12 = 01110101 01110001 11110101 10010100 01100111 11101001
L12 = 11000101 01111000 00111100 01111000
R12 = 01110101 10111101 00011000 01011000
```

$K_{I3} = 100101 111100 010111 010001 111110 101011 101001 000001$

```
Bloque ronda 13 = 11000101 01111000 00111100 01111000 01110101 10111101 00011000 01011000
Final k13 = 10010111 11000101 11010001 11111010 10111010 01000001
L13 = 01110101 10111101 00011000 01011000
R13 = 00011000 11000011 00010101 01011010
```

$K_{I4} = 010111 110100 001110 110111 111100 101110 011100 111010$

```
Bloque ronda 14 = 01110101 10111101 00011000 01011000 00011000 11000011 00010101 01011010
Final k14 = 01011111 01000011 10110111 11110010 11100111 00111010
L14 = 00011000 11000011 00010101 01011010
R14 = 11000010 10001100 10010110 00001101
```

$K_{I5} = 101111 111001 000110 001101 001111 010011 111100 001010$

```
Bloque ronda 15 = 00011000 11000011 00010101 01011010 11000010 10001100 10010110 00001101
Final k15 = 10111111 10010001 10001101 00111101 00111111 00001010
L15 = 11000010 10001100 10010110 00001101
R15 = 01000011 01000010 00110010 00110100
```

$K_{I6} = 110010 110011 110110 001011 000011 100001 011111 110101$

$L_{I6} = 0100 0011 0100 0010 0011 0010 0011 0100$

$R_{I6} = 0000 1010 0100 1100 1101 1001 1001 0101$

```
Bloque ronda 16 = 11000010 10001100 10010110 00001101 01000011 01000010 00110010 00110100
Final k16 = 11001011 00111101 10001011 00001110 00010111 11110101
L16 = 01000011 01000010 00110010 00110100
R16 = 00001010 01001100 11011001 10010101
```

Todas las subclaves y el resultado final de las 16 rondas es correcto. Y para finalizar el SWAP final y la inversa de la permutación inicial (IP^{-1}):

$IP^I = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101$

```
IP-1 Bloque 1 = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101
```

Ambos resultados coinciden, tras estas comprobaciones podemos concluir que nuestro algoritmo DES cifra correctamente un bloque.

El segundo caso es cuando sí que se indica fichero de entrada (-i) y fichero de salida (-o), en este caso se realiza un cifrado DES del fichero de entrada con el modo CBC y el resultado del cifrado se escribe en el fichero de salida.

Si la opción escogida es -D se realiza el descifrado del DES, cuya única diferencia con el cifrado es aplicar las claves en orden inverso y el modo de aplicar el CBC. De nuevo hay dos opciones:

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Sin indicar fichero de entrada ni de salida, en este caso imprimimos por pantalla cada paso del descifrado de un bloque comprobando su correcto funcionamiento. La clave y el texto cifrado corresponden al del ejemplo como podemos ver en la imagen. El texto cifrado se trata del obtenido en la anterior prueba:

```
#####CIFRADO/DESCIFRADO DE EJEMPLO#####
Texto cifrado = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101
Clave = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

K+ = 11110000 11001100 10101010 11110101 01010110 01100111 10001111
k1 = 11100001 10011001 01010101 11110100 10101100 11001111 00011110
k2 = 11000011 00110010 10101011 11110101 01011001 10011110 00111101
k3 = 00001100 11001010 10101111 11110101 01100110 01111000 11110101
k4 = 00110011 00101010 10111111 11000101 10011001 11100011 11010101
k5 = 11001100 10101010 11111111 00000110 01100111 10001111 01010101
k6 = 00110010 10101011 11111100 00111001 10011110 00111101 01010101
k7 = 11001010 10101111 11110000 11000110 01111000 11110101 01010110
k8 = 00101010 10111111 11000011 00111001 11100011 11010101 01011001
k9 = 01010101 01111111 10000110 01100011 11000011 10101010 10110011
k10 = 01010101 11111110 00011001 10011111 00011110 10101010 11001100
k11 = 01010111 11111000 01100110 01011100 01110101 10101011 00110011
k12 = 01011111 11100001 10011001 01010001 11101010 10101100 11001111
k13 = 01111111 10000110 01100101 01010111 10101010 10110011 00111100
k14 = 11111110 00011001 10010101 01011110 10101010 11001100 11110001
k15 = 11111000 01100110 01010101 01111010 10101011 00110011 11000111
k16 = 11110000 11001100 10101010 11110101 01010110 01100111 10001111

IP Bloque 1 = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100
Bloque ronda 1 = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100
Final k1 = 11001011 00111101 10001011 000001110 00010111 11110101
L1 = 01000011 01000010 00110010 00110100
R1 = 11000010 10001100 10010110 000001101

Bloque ronda 2 = 01000011 01000010 00110010 00110100 11000010 10001100 10010110 000001101
Final k2 = 10111111 10010001 10001101 00111101 00111111 000001010
L2 = 11000010 10001100 10010110 000001101
R2 = 00011000 11000011 00010101 01011010

Bloque ronda 3 = 11000010 10001100 10010110 00001101 00011000 11000011 00010101 01011010
Final k3 = 01011111 01000011 10110111 11110010 11100111 00111010
L3 = 00011000 11000011 00010101 01011010
R3 = 01110101 10111101 00011000 01011000

Bloque ronda 4 = 00011000 11000011 00010101 01011010 01110101 10111101 00011000 01011000
Final k4 = 10010111 11000101 11010001 11111010 10111010 01000001
L4 = 01110101 10111101 00011000 01011000
R4 = 11000101 01110000 00111100 01111000

Bloque ronda 5 = 01110101 10111101 00011000 01011000 11000101 01111000 00111100 01111000
Final k5 = 01110101 01110001 11110101 10010100 01100111 11101001
L5 = 11000101 01110000 00111100 01111000
R5 = 10110111 11010101 11010111 10110010

Bloque ronda 6 = 11000101 01110000 00111100 01111000 10110111 11010101 11010111 10110010
Final k6 = 00100001 01011111 11010011 11011110 11010011 10000110
L6 = 10110111 11010101 11010111 10110010
R6 = 00100100 01111100 11000110 01111010
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
Bloque ronda 7 = 10110111 11010101 11010111 10110010 00100100 01111100 11000110 01111010
Final k7 = 10110001 11110011 01000111 10111010 01000110 01001111
L7 = 00100100 01111100 11000110 01111010
R7 = 11010101 01101001 01001011 10010000

Bloque ronda 8 = 00100100 01111100 11000110 01111010 11010101 01101001 01001011 10010000
Final k8 = 11100000 11011011 11101011 11011011 11100111 10000001
L8 = 11010101 01101001 01001011 10010000
R8 = 00000010 01001010 10111010 00010000

Bloque ronda 9 = 11010101 01101001 01001011 10010000 00000110 01001010 10111010 00010000
Final k9 = 11110111 10001010 00111010 11000001 00111011 11111011
L9 = 00000010 01001010 10111010 00010000
R9 = 11101001 01100111 11001101 01101001

Bloque ronda 10 = 00000010 01001010 10111010 00010000 11101001 01100111 11001101 01101001
Final k10 = 11101100 10000100 10110111 11110110 00011000 10111100
L10 = 11101001 01100111 11001101 01101001
R10 = 10001010 01001111 10100110 00110111

Bloque ronda 11 = 11101001 01100111 11001101 01101001 10001010 01001111 10100110 00110111
Final k11 = 01100011 10100101 00111110 01010000 01111011 00101111
L11 = 10001010 01001111 10100110 00110111
R11 = 01110111 00100010 00000000 01000101

Bloque ronda 12 = 10001010 01001111 10100110 00110111 01110111 00100010 00000000 01000101
Final k12 = 01111000 11101100 00000111 11101011 01010011 10101000
L12 = 01110111 00100010 00000000 01000101
R12 = 10100010 01011100 00001011 11110100

Bloque ronda 13 = 01110111 00100010 00000000 01000101 10100010 01011100 00001011 11110100
Final k13 = 01110010 10101101 11010110 11011011 00110101 00011101
L13 = 10100010 01011100 00001011 11110100
R13 = 11001100 00000001 01101111 00001001

Bloque ronda 14 = 10100010 01011100 00001011 11110100 11001100 00000001 01110111 00001001
Final k14 = 01010101 11111100 10001010 01000010 11001111 10011001
L14 = 11001100 00000001 01101011 00001001
R14 = 11101111 01001010 01100101 01000100

Bloque ronda 15 = 11001100 00000001 01110111 00001001 11101111 01001010 01100101 01000100
Final k15 = 01111001 10101110 11011001 11011011 11001001 11100101
L15 = 11101111 01001010 01100101 01000100
R15 = 11110000 10101010 11110000 10101010

Bloque ronda 16 = 11101111 01001010 01100101 01000100 11110000 10101010 11110000 10101010
Final k16 = 00011011 00000010 11101111 11111100 01110000 01110010
L16 = 11110000 10101010 11110000 10101010
R16 = 11001100 00000000 11001100 11111111

IP-1 Bloque 1 = 00000001 00100011 01000101 01100111 10001001 10101011 11001101 11101111

Memoria liberada correctamente. Cerrando programa...
```

El resultado del descifrado corresponde con el texto plano utilizado en el cifrado, por lo tanto, el descifrado también es correcto. Tras este resultado podemos concluir que nuestro algoritmo DES cifra y descifra correctamente.

La otra opción, cuando se indica fichero de entrada y de salida, consiste en descifrar el contenido del fichero de entrada y guardar el resultado en el fichero de salida. Aplicando el modo CBC a cada bloque cifrado.

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Ahora explicaremos cómo lo hemos implementado. Lo primero que hacemos es leer el fichero de entrada que vamos a cifrar o descifrar, y abrir en modo escritura el fichero de salida donde vamos a escribir el resultado del cifrado o descifrado.

Tras esto, generamos las 16 subclaves. Para ello primero tenemos que conseguir la clave inicial, permutando y comprimiendo la clave inicial de 64 bits, eliminando los bits de paridad, esto lo conseguimos aplicando la permutación **PC-1** que se encuentra en el fichero **des_tables.h**. La clave pasa a tener 56 bits. Una vez hemos obtenido la clave inicial pasamos a generar las 16 subclaves, cada subclave es generada a partir de la anterior. Dividimos la clave anterior en dos, y a cada mitad le aplicamos una rotación hacia la izquierda de un bit o dos dependiendo de lo que indique **ROUND_SHIFTS** (definida en **des_tables.h**) para esa ronda (**LCSi**), unimos ambas mitades y obtenemos la subclave para esa ronda. Guardamos las subclaves en un array hasta que tengamos que utilizarlas en las rondas Feistel.

Una vez obtenidas las 16 subclaves ya podemos realizar el cifrado DES de un bloque de texto plano. Por lo tanto, vamos a ir cifrando el texto plano bloque a bloque de tamaños 64 bits, que es lo mismo que 8 caracteres. Es posible que para los últimos caracteres del texto plano no sea posible formar un bloque, en ese caso lo que hacemos es añadir 0 hasta conseguir que sea de tamaño 64 bits.

Como el **modo de operación es CBC**, en el caso de estar cifrando, al bloque le tenemos que aplicar una XOR con el anterior bloque cifrado, y si se trata del primer bloque una XOR con el vector de inicialización.

Ya estemos cifrando o descifrando al bloque hay que aplicarle la permutación inicial **IP** (definida en **des_tables.h**). Y a continuación vienen las 16 rondas Feistel.

La diferencia entre el cifrado y el descifrado, como hemos dicho anteriormente, es que en el descifrado las subclaves se emplean de manera inversa, es decir, primero la subclave 16 en la ronda 1 y en la última ronda la subclave 1. Tras esta aclaración las **rondas de Feistel** para el cifrado y descifrado son iguales.

Primero aplicamos la permutación **PC-2** (definida en **des_tables.h**) a la subclave correspondiente a esa ronda, la cual comprime la clave haciendo que pase de 56 bits a 48 bits, y el resultado de esta permutación es la clave que emplearemos finalmente en esa ronda.

La entrada a cada ronda Feistel es el resultado de la ronda anterior o la IP en caso de ser la primera ronda, tienen tamaño 64 bits. Dividimos estos 64 bits en dos mitades, los primeros 32 bits es **Li-1** y los otros 32 son **Ri-1**, para obtener el resultado de la ronda hay que hacer lo siguiente:

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

$L_i = R_{i-1}$

$R_i = L_{i-1} \text{ XOR } F(R_{i-1}, K_i)$

Los primeros 32 bits del resultado de la ronda, son muy fáciles de obtener ya que son la segunda mitad de los 64 bits. Pero para los otros 32 bits del resultado hay que realizar muchos más pasos.

Primero hay que obtener el resultado de $F(R_{i-1}, K_i)$ para ello primero aplicamos la permutación **E** (definida en **des_tables.h**) que se encarga de permutar y expandir R_{i-1} el cual pasa a tener 48 bits como la subclave K_i . A continuación, hacemos la **XOR entre el resultado de la permutación E y la subclave** para esa ronda, el resultado de la XOR pasará por las S-boxes del DES.

Los 48 bits los vamos a ir sustituyendo en grupos de 6 bits, salen en total 8 grupos. El primer grupo se sustituye con la primera caja de sustitución y así sucesivamente. Para cada grupo de 6 bits obtenemos la fila y la columna para acceder a la caja de sustitución, la fila es el bit 0 y el bit 5, y la columna se obtiene a partir de los bits 1, 2, 3 y 4. Una vez obtenidas la fila y columna accedemos a esa posición de la **S_BOXES[]** (definida en **des_tables.h**) y obtenemos los 4 bits por los que vamos a sustituir los 6 bits. Tras aplicar a los 8 grupos la sustitución obtenemos 32 bits.

A la salida de las cajas de sustitución le aplicamos la permutación **P**, la cual es el resultado de $F(R_{i-1}, K_i)$.

Por lo tanto, ya podemos obtener los siguientes 32 bits haciendo la **XOR entre L_{i-1} y $F(R_{i-1}, K_i)$** . Ya tenemos ambas mitades, las unimos y obtenemos el resultado de la ronda Feistel.

Una vez realizadas las 16 rondas Feistel, realizamos el **SWAP final** del DES, para ello intercambiamos L_{16} por R_{16} y R_{16} por L_{16} . Y para acabar aplicamos la permutación inversa IP (**IP_INV**) y este es el resultado final del bloque cifrado final.

En el caso del descifrado hay que aplicar el **modo de operación CBC**, para ello hacemos una **XOR entre el resultado de la permutación IP_INV y el bloque cifrado anterior**. Y este es el bloque descifrado.

Cada bloque cifrado/descifrado los vamos escribiendo en el fichero de salida (-o).

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Algunos ejemplos de ejecución:

- Ciframos y desciframos un gif

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./descBC -C -k 123456876 -iv CHONCHON -i cascada.gif -o salida
Clave:
00110001 00110010 00110011 00110100 00110101 00110110 00111000 00110111
Vector de inicializacion:
01000011 01001000 01001111 01001110 01000011 01001000 01001111 01001110
Ha elegido la opcion de cifrado...

Memoria liberada correctamente. Cerrando programa...
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./descBC -D -k 123456876 -iv CHONCHON -i salida -o resultado.gif
Clave:
00110001 00110010 00110011 00110100 00110101 00110110 00111000 00110111
Vector de inicializacion:
01000011 01001000 01001111 01001110 01000011 01001000 01001111 01001110
Ha elegido la opcion de descifrado...

Memoria liberada correctamente. Cerrando programa...
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```



- Ciframos y desciframos un ejecutable

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./desCBC -C -k 123456876 -iv CHONCHON -i seg_perf -o salida
Clave:
00110001 00110010 00110011 00110100 00110101 00110110 00111000 00110111
Vector de inicializacion:
01000011 01001000 01001111 01001110 01000011 01001000 01001111 01001110
Ha elegido la opcion de cifrado...

Memoria liberada correctamente. Cerrando programa...
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./desCBC -D -k 123456876 -iv CHONCHON -i salida -o ejecutable
Clave:
00110001 00110010 00110011 00110100 00110101 00110110 00111000 00110111
Vector de inicializacion:
01000011 01001000 01001111 01001110 01000011 01001000 01001111 01001110
Ha elegido la opcion de descifrado...

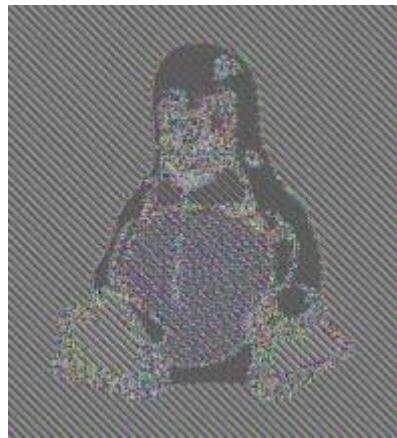
Memoria liberada correctamente. Cerrando programa...
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./ejecutable
Error en los argumentos. Introduce:
./seg_perf [-P|-I] [-i filein] [-o fileout]
[-i] [-o] son opcionales.
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

En el modo de operación **CBC** cada bloque de texto cifrado depende de todo el texto plano procesado hasta ese punto. Al emplearse un vector de inicialización hace que el mensaje cifrado sea único. En cambio, el modo de operación **ECB** cifra los bloques de forma independiente, es idóneo para mensajes cortos, pero en mensajes largos puede dejar pistas ya que los bloques iguales se cifran igual. Por ejemplo si cifras una imagen con ECB es posible que a simple vista se pueda ver la imagen (ejemplo sacado de internet):



B. Programación del Triple DES

En este apartado nos pide implementar el método triple DES en el modo CBC (Cipher-block chaining). Por lo tanto, hemos creado **TDEA_CBC.c**, que corresponde al main de este apartado. Las funciones que utiliza están implementadas en el fichero **funciones_DES_CBC.c** con las cabeceras de las funciones en la librería **funciones_DES_CBC.h**, además también incluye la librería **des_tables.h** que contiene las S-boxes y las permutaciones necesarias para el DES. El programa ejecutable posee la siguiente entrada por argumentos:

> ./TDEA_CBC {-C|-D -k clave -iv vectorinicializacion} [-i fichero_entrada] [-o fichero_salida]

La única diferencia respecto a los argumentos con el DES simple es que la clave tiene que tener 192 bits, es decir, 24 caracteres. Además el fichero de entrada y de salida son obligatorios.

Al tratarse de un triple DES con tres claves diferentes se hace de la siguiente manera:

$$C_n = DES_k3(DES^{^-1}_k2(DES_k1(P_n)))$$

$$P_n = DES^{^-1}_k1(DES_k2(DES^{^-1}_k3(C_n)))$$

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Explicaremos a continuación como lo hemos implementado, primero leemos el fichero de entrada y abrimos en modo escritura el fichero de salida. La clave que nos han pasado por argumento es de 192 bits o 24 caracteres, la dividimos en tres claves cada una de 8 caracteres o lo que es lo mismo 64 bits. Por lo tanto, la **k1[0:7], k2[8,15], y k3[16:23]** tratando la clave como si fueran 24 caracteres.

Igual que en el DES simple para k1, k2, y k3 generamos las 16 subclaves que utilizaremos en las rondas Feistel. Es decir, primero aplicaremos la permutación **PC1** a las claves iniciales que es la encargada de quitar los bits de paridad, pasan a tener 56 bits, y luego generamos las subclaves con **LCSi** como explicamos en el apartado anterior. Tras esto, lo que tenemos son 3 conjuntos de 16 subclaves, el primer conjunto de las subclaves generadas por k1, el segundo por k2, y el tercero por k3.

A continuación empezamos a cifrar bloque a bloque, en el caso de estar cifrando (-C) aplicamos el modo CBC. Y aquí ya llega la gran diferencia con el DES simple, tenemos que aplicar tres veces el DES cada vez con una clave diferente. Hay dos casos diferentes:

1. Cifrado → **Cn = DES_k3(DES^-1_k2(DES_k1(Pn)))**:

Primero haremos un cifrado DES del bloque, utilizando en las rondas Feistel el primer conjunto de subclaves generadas de la clave inicial k1. Después a este bloque resultante le aplicamos otro cifrado DES pero utilizando de forma inversa las subclaves generadas por la clave inicial k2, es decir, como en un descifrado del DES simple. Y por ultimo, al resultado de esta segunda ronda del DES le aplicamos el ultimo cifrado DES utilizando las subclaves generadas por k3. Este es el resultado del bloque cifrado con triple DES.

2. Descifrado → **Pn = DES^-1_k1(DES_k2(DES^-1_k3(Cn)))**:

Primero hacemos un descifrado DES con las subclaves generadas por k3, es decir, las subclaves se aplican en orden inverso. Luego un cifrado DES con las subclaves generadas por k2. Y para acabar un descifrado DES con las subclaves generadas por k1. Al tratarse del descifrado, y estar utilizando el modo de operación CBC, es necesario aplicar el modo CBC a este resultado para obtener el resultado final del bloque descifrado con el triple DES.

Cada vez que obtengamos un bloque cifrado o descifrado lo escribimos en el fichero de salida.

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Algunos ejemplos de ejecución:

- Cifrado y descifrado de una imagen

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./TDEA_CBC -C -k ABCDEFGHIJKLMNOPQRSTUVWXYZ -iv CHONCHON -i paisaje.jpg -o salida
Ha elegido la opcion de cifrado...
Clave 1:
01000001 01000010 01000011 01000100 01000101 01000110 01000111 01001000
Clave 2:
01001001 01001010 01001011 01001100 01001101 01001110 01001111 01010000
Clave 3:
01010001 01010010 01010011 01010100 01010101 01010110 01010111 01011000
Vector de inicializacion:
01000011 01001000 01001111 01001110 01000011 01001000 01001111 01001110

Memoria liberada correctamente. Cerrando programa...
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./TDEA_CBC -D -k ABCDEFGHIJKLMNOPQRSTUVWXYZ -iv CHONCHON -i salida -o result_imagen
Ha elegido la opcion de descifrado...
Clave 1:
01000001 01000010 01000011 01000100 01000101 01000110 01000111 01001000
Clave 2:
01001001 01001010 01001011 01001100 01001101 01001110 01001111 01010000
Clave 3:
01010001 01010010 01010011 01010100 01010101 01010110 01010111 01011000
Vector de inicializacion:
01000011 01001000 01001111 01001110 01000011 01001000 01001111 01001110

Memoria liberada correctamente. Cerrando programa...
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```



- Cifrado y descifrado de un texto largo

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./TDEA_CBC -C -k ABCDEFGHIJKLMNOPQRSTUVWXYZ -iv CHONCHON -i don_quixote.txt -o salida
Ha elegido la opcion de cifrado...
Clave 1:
01000001 01000010 01000011 01000100 01000101 01000110 01000111 01001000
Clave 2:
01001001 01001010 01001011 01001100 01001101 01001110 01001111 01010000
Clave 3:
01010001 01010010 01010011 01010100 01010101 01010110 01010111 01011000
Vector de inicializacion:
01000011 01001000 01001111 01001110 01000011 01001000 01001111 01001110

Memoria liberada correctamente. Cerrando programa...
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./TDEA_CBC -D -k ABCDEFGHIJKLMNOPQRSTUVWXYZ -iv CHONCHON -i salida -o don_quixote_res.txt
Ha elegido la opcion de descifrado...
Clave 1:
01000001 01000010 01000011 01000100 01000101 01000110 01000111 01001000
Clave 2:
01001001 01001010 01001011 01001100 01001101 01001110 01001111 01010000
Clave 3:
01010001 01010010 01010011 01010100 01010101 01010110 01010111 01011000
Vector de inicializacion:
01000011 01001000 01001111 01001110 01000011 01001000 01001111 01001110

Memoria liberada correctamente. Cerrando programa...
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ diff don_quixote.txt don_quixote_res.txt
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

3. Principios de diseño del DES

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

Para demostrar la no linealidad de las S-boxes del DES hicimos el programa **linealidad_des.c**, que corresponde al main de este apartado. Las funciones que utiliza están implementadas en el fichero **funciones_DES_CBC.c** con las cabeceras de las funciones en la librería **funciones_DES_CBC.h**, además también incluye la librería **des_tables.h** que contiene las S-boxes. El programa ejecutable posee la siguiente entrada por argumentos:

> ./linealidad_des

Para que tenga no linealidad es necesario que **S(A XOR B) != S(A) XOR S(B)**.

Para demostrar la no linealidad de las S-boxes del DES lo que hicimos fue probar para cada una de las 8 cajas de sustitución todas las parejas posibles de valores de 6 bits, es decir, del 0 al 63 y comprobar si cumplían o no la condición. En total probamos para cada caja 64×64 casos.

Vemos un ejemplo de ejecución:

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./linealidad_des
Se cumple la linealidad en la S-box 0 para los valores x=4 e y=19
Se cumple la linealidad en la S-box 0 para los valores x=4 e y=23
Se cumple la linealidad en la S-box 0 para los valores x=4 e y=27
Se cumple la linealidad en la S-box 0 para los valores x=4 e y=31
Se cumple la linealidad en la S-box 0 para los valores x=4 e y=42
Se cumple la linealidad en la S-box 0 para los valores x=4 e y=46
Se cumple la linealidad en la S-box 0 para los valores x=4 e y=59
Se cumple la linealidad en la S-box 0 para los valores x=4 e y=63
Se cumple la linealidad en la S-box 0 para los valores x=5 e y=58
Se cumple la linealidad en la S-box 0 para los valores x=5 e y=63
Se cumple la linealidad en la S-box 0 para los valores x=6 e y=17
Se cumple la linealidad en la S-box 0 para los valores x=6 e y=23
Se cumple la linealidad en la S-box 0 para los valores x=6 e y=25
Se cumple la linealidad en la S-box 0 para los valores x=6 e y=31
Se cumple la linealidad en la S-box 0 para los valores x=9 e y=55
Se cumple la linealidad en la S-box 0 para los valores x=9 e y=62
Se cumple la linealidad en la S-box 0 para los valores x=10 e y=17
Se cumple la linealidad en la S-box 0 para los valores x=10 e y=18
Se cumple la linealidad en la S-box 0 para los valores x=10 e y=19
Se cumple la linealidad en la S-box 0 para los valores x=10 e y=24

Se cumple la linealidad en la S-box 7 para los valores x=62 e y=16
Se cumple la linealidad en la S-box 7 para los valores x=62 e y=29
Se cumple la linealidad en la S-box 7 para los valores x=62 e y=35
Se cumple la linealidad en la S-box 7 para los valores x=62 e y=46
Se cumple la linealidad en la S-box 7 para los valores x=62 e y=59
Se cumple la linealidad en la S-box 7 para los valores x=63 e y=19
Se cumple la linealidad en la S-box 7 para los valores x=63 e y=44

La caja 0 tiene frecuencia de no linealidad 0.960449
La caja 1 tiene frecuencia de no linealidad 0.963379
La caja 2 tiene frecuencia de no linealidad 0.939941
La caja 3 tiene frecuencia de no linealidad 0.950195
La caja 4 tiene frecuencia de no linealidad 0.964844
La caja 5 tiene frecuencia de no linealidad 0.937012
La caja 6 tiene frecuencia de no linealidad 0.961914
La caja 7 tiene frecuencia de no linealidad 0.945801
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ □
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Como podemos observar en las imágenes anteriores hay algunos casos donde sí se cumple la linealidad, en estos casos incrementamos un contador que hemos hecho para cada caja que cuenta las veces que se cumple.

Por lo tanto, una vez obtenidas las veces que se cumplen para cada caja, dividimos ese número entre el total de casos que hemos probado que han sido 64×64 y nos sale la probabilidad de no linealidad de cada caja de sustitución. En la segunda imagen vemos estos resultados, van del **93% al 96%** que son probabilidades bastante altas por lo que podemos concluir que las S-boxes del DES sí que cumplen la no linealidad.

B. Estudio del Efecto de Avalanche

En este apartado vamos a estudiar el efecto avalanche del DES, es decir, cuántos bits cambian a la salida del DES cambiando un bit del bloque o un bit de la clave. Para ello hemos creado el programa **avalancha_des.c**, que corresponde al main de este apartado. Algunas de las funciones que utiliza están implementadas en el fichero **funciones_DES_CBC.c** con las cabeceras de las funciones en la librería **funciones_DES_CBC.h**, además también incluye la librería **des_tables.h** que contiene las S-boxes y las permutaciones necesarias para el DES. El programa ejecutable posee la siguiente entrada por argumentos:

> ./avalancha_des {-B|-K}

-B: si lo que queremos es estudiar que ocurre cuando se cambia un bit del bloque de entrada.

-K: si lo que queremos es estudiar que ocurre cuando se cambia un bit de la clave inicial.

Lo que hacemos es generar un bloque y una clave de 64 bits aleatorios, en el caso de haber elegido la opción -B modificamos un bit del bloque y ejecutamos el algoritmo DES de manera paralela para el bloque y para el bloque modificado imprimiendo por pantalla el número de bits diferentes en cada una de las 16 rondas Feistel. Y lo mismo hacemos para la opción -K, pero en este caso lo que modificamos es un bit de la clave y ejecutaremos paralelamente el mismo bloque pero uno con las subclaves generadas de la clave sin modificar y el otro con las subclaves de la clave modificada. A continuación estudiamos los resultados con ambas opciones:

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Ejemplo de ejecución:

- Cambiando un bit del bloque de entrada:

```
laura@Universidad:~/Escritorio/CUARTO/cripto/P2$ ./avalancha_des -B
Bloque generado: 11010001 10101110 01101110 00001110 01100101 10001001 10110000 10001001
Clave generada: 01111110 10010000 11010010 11010011 10010010 10010100 11010100 00011010
Has elegido la opcion de modificar un bit del bloque:
11010001 10101110 01101110 00001110 01100101 10001001 10110000 10001001

Ronda 1
Bloque normal: 11100011 01010110 10101110 00001110 00111011 01000101 10011011 00001010
Bloque modificado: 11100011 01010110 11010110 00001110 00011011 01000001 10011011 00001010
Se diferencian en 3 bits

Ronda 2
Bloque normal: 00111011 01000101 10011011 00001010 10111110 10101100 00011111 00011100
Bloque modificado: 00011011 01000001 10011011 00001010 00111110 01101100 11011101 01011110
Se diferencian en 10 bits

Ronda 3
Bloque normal: 10111110 10101100 00011111 00011100 01011010 10111010 11011100 10101010
Bloque modificado: 00111110 01101100 11011101 01011110 01001000 01100101 10111011 11000011
Se diferencian en 26 bits

Ronda 4
Bloque normal: 01011010 10111010 11011100 10101010 11001111 01110010 11111110 01110010
Bloque modificado: 01001000 01100101 10111011 11000011 00110001 01100000 00100010 00100001
Se diferencian en 36 bits

Ronda 5
Bloque normal: 11001111 01110010 11111110 01110010 01011111 00111111 11010011 00101100
Bloque modificado: 00110001 01100000 00100010 00100001 10001110 11011101 11100001 10001000
Se diferencian en 32 bits

Ronda 6
Bloque normal: 01011111 00111111 11010011 00101100 10010010 10111001 00110110 10100100
Bloque modificado: 10001110 11010101 11000001 10001000 00111100 10111010 00010011 01101100
Se diferencian en 27 bits

Ronda 7
Bloque normal: 10010010 10111001 00110110 10100100 00000100 11111100 01101010 10100011
Bloque modificado: 00111100 10111010 00010011 01101100 01111111 00001000 11011001 00011101
Se diferencian en 34 bits

Ronda 8
Bloque normal: 00000100 11111100 01101011 10100011 10011101 10011101 00000011 11011100
Bloque modificado: 01111111 00001000 11011001 00011101 00001011 10011000 10100100 11110000
Se diferencian en 35 bits

Ronda 9
Bloque normal: 10011101 10011101 00000011 11101100 00101011 01100010 11110111 01100110
Bloque modificado: 00001011 10011000 10100100 11100000 10110011 00010000 01110101 11011010
Se diferencian en 28 bits

Ronda 10
Bloque normal: 00101011 01100010 11110111 01100110 11000111 01010101 01111100 00000100
Bloque modificado: 10110011 00010000 01110101 11011010 00011001 10001110 00011011 10101100
Se diferencian en 35 bits

Ronda 11
Bloque normal: 11100111 01010101 01111100 00000100 10110011 00100010 10011100 00101100
Bloque modificado: 00011001 10001110 00011011 10101100 00111010 00111111 10001111 10010111
Se diferencian en 36 bits

Ronda 12
Bloque normal: 10110011 00100110 10011100 00101100 10100010 00010000 10010010 11101011
Bloque modificado: 00111010 00111111 10001111 10010111 00001101 11101000 10010000 11101110
Se diferencian en 28 bits

Ronda 13
Bloque normal: 10101110 00010000 10010010 11101011 00111110 11000010 01001100 11001111
Bloque modificado: 00001101 11101000 10010000 11101110 10100111 11011000 01110100 00001100
Se diferencian en 27 bits

Ronda 14
Bloque normal: 00111110 11000010 01001100 11001111 11001001 11000010 10100010 00111101
Bloque modificado: 10100111 11010000 01101000 00001100 11101010 10001001 01101001 00100010
Se diferencian en 31 bits

Ronda 15
Bloque normal: 11001001 11000010 10100010 00111101 01011100 10010111 11011001 10001110
Bloque modificado: 11101010 10001001 01101001 00100010 11100010 00101011 00001100 10100000
Se diferencian en 36 bits

Ronda 16
Bloque normal: 01011100 10010111 11011001 10001110 10100010 01000101 11000001 01011111
Bloque modificado: 11100010 00101011 00001100 10100000 01011111 10000100 01111101 11010110
Se diferencian en 37 bits
```

Para la opción -B podemos ver como para las 4 primeras rondas el número de bits diferentes se va incrementando teniendo en la primera solo 3 bits diferentes y en la cuarta 36 bits. A partir de la quinta ronda se mantiene entre los 27 y los 37 bits de diferencia.

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

- Cambiando un bit de la clave inicial:

```
laura@Universidad:~/Escritorio/ CUARTO/cripto/P2S ./avalancha_des -K
Bloque generado: 10111100 01001001 11010000 01010010 10001101 11010000 00100000 10101110
Clave generada: 11010010 10011011 00010010 10000010 00011101 11001000 10000001 01010100
Has elegido la opcion de modificar un bit de la clave:
11010010 10011011 00010010 10000010 00011101 11001000 10000001 01010100

Ronda 1
Bloque normal: 10110101 11000101 10010111 10001000 01110100 01010101 11110011 10111100
Bloque modificado: 10110101 11000101 10010111 10001000 01110100 00010101 11100011 11111100
Se diferencian en 3 bits

Ronda 2
Bloque normal: 01110100 01010101 11110011 10111100 00110111 10100101 01000000 00011001
Bloque modificado: 01110100 00010101 11000011 11111100 00110111 10010000 01000001 10010100
Se diferencian en 12 bits

Ronda 3
Bloque normal: 00110111 10100101 01000000 00011001 01000111 10010111 01110100 10001011
Bloque modificado: 00110111 10010000 01000001 10010100 11000100 00000010 01011000 11001011
Se diferencian en 20 bits

Ronda 4
Bloque normal: 01000111 10010111 01110100 10001011 00001101 10110010 10001110 10011111
Bloque modificado: 11000100 00000010 01011000 11001011 11110010 01010110 10001101 10111111
Se diferencian en 26 bits

Ronda 5
Bloque normal: 00001101 10110010 10001110 10011111 00001101 11010110 00111001 11010111
Bloque modificado: 11110010 01010110 10001101 10111111 10001010 10110010 11011111 11010101
Se diferencian en 33 bits

Ronda 6
Bloque normal: 00001101 11010110 00111001 11010111 10110011 00100010 01000010 10011110
Bloque modificado: 10010010 10110010 11011111 11010100 11010011 00001010 01010111 11110101
Se diferencian en 31 bits

Ronda 7
Bloque normal: 10110011 00100100 01000010 10011110 11110001 01111000 01011111 01000110
Bloque modificado: 11010011 00001010 01101011 11110011 11010101 11110101 10011011 00111011
Se diferencian en 28 bits

Ronda 8
Bloque normal: 11111001 01111100 01011111 01000110 00100111 10100110 00000110 00100110
Bloque modificado: 11010101 11110101 10011011 00111011 10101110 11111111 10110111 01011011
Se diferencian en 32 bits

Ronda 9
Bloque normal: 00100111 10100110 00000110 00100110 01001100 00110110 11010110 00100011
Bloque modificado: 10101110 11111111 10110111 01011011 11001011 00101011 11011100 01001100
Se diferencian en 33 bits

Ronda 10
Bloque normal: 01001100 00110110 11010110 00100011 10000111 10111101 00010011 11111011
Bloque modificado: 11001011 00101011 11011100 01001100 00100100 01101011 11001001 01101110
Se diferencian en 32 bits

Ronda 11
Bloque normal: 10000111 10111101 00010011 11110101 11110101 01011111 10001001 11001010
Bloque modificado: 00100100 01101010 11001001 01010110 01100011 01001100 00011101 01010011
Se diferencian en 29 bits

Ronda 12
Bloque normal: 11110101 01011111 10001001 11001010 00101101 11100011 00010111 11010010
Bloque modificado: 01100111 00011000 00011101 01010011 11110100 11000011 11111100 01001111
Se diferencian en 30 bits

Ronda 13
Bloque normal: 00101101 11100011 00010111 11010010 01111100 01100110 00010000 01010111
Bloque modificado: 11110101 11100011 11111100 01001111 10110100 00101101 10110001 11000000
Se diferencian en 32 bits

Ronda 14
Bloque normal: 01111100 01100110 00010000 01010111 10110011 11001100 00011011 00111001
Bloque modificado: 10110100 00101101 10110001 11000000 10000010 00001001 01010000 10011000
Se diferencian en 29 bits

Ronda 15
Bloque normal: 10110011 11001100 00011011 00111001 01010110 11000011 11101011 01011110
Bloque modificado: 00000010 00001001 01010000 10011000 00100010 10000000 01000111 01010001
Se diferencian en 30 bits

Ronda 16
Bloque normal: 01010110 11100011 11010111 01011110 11010001 00010011 11100001 11011110
Bloque modificado: 00100010 10000000 01000111 01010001 01110101 00000100 01111101 11010011
Se diferencian en 31 bits
```

Y para esta opción -K pasa algo parecido, en este caso los bits se incrementan hasta la ronda 5, teniendo 3 bits diferentes en la primera ronda y 33 bits es la quinta. A partir de la sexta ronda se mantienen los bits diferentes rondando los valores entre 28 y 32 bits.

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

Podemos concluir que el efecto avalancha si se produce y que se incrementa sobretodo en las primeras 5 rondas y luego se mantiene.

4. Principios de diseño del AES

En esta parte de la práctica nos centramos en el algoritmo de cifrado del AES.

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

Concretamente, en este apartado se nos pide estudiar la no linealidad de las S-boxes que posee el AES. El programa que hemos creado es muy parecido al que hemos hecho en el apartado A del punto anterior. Pero antes, el archivo que posee el código correspondiente a esta parte es **linealidad_aes.c** y hace uso de la librería **aes_tables.h** que posee las S-boxes ya construidas. La entrada por argumentos y ejecución del ejecutable correspondiente es:

> ./linealidad_aes

donde no hay argumentos que introducir.

Primero, AES posee dos cajas de sustitución afín. Una directa para el cifrado en ByteSub y otra inversa para el descifrado en InvByteSub. Ambas son dos matrices de 16*16 bytes que corresponden a la **transformación afín**, directa o inversa, del byte de entrada. Esta transformación afín corresponde a la multiplicación del byte de entrada por una matriz X (o su inversa Y) y una posterior XOR con una constante C (o D en el caso inverso), y ambas son un estándar elegido, de manera que la S-box equivalente no posea puntos fijos, es decir, que $S\text{-box}(b) = b$, ni puntos fijos opuestos, donde $S\text{-box}(b) = \text{complemento}(b)$. Por lo tanto, ambas fórmulas quedarían, para la **directa** $y = X^*(b)^{-1} + C$, y para la **inversa** $y = Y^*b + D$ y luego **se halla (y)⁻¹** en el caso inverso.

Por lo tanto, estas dos S-boxes almacenarán las transformaciones afines correspondientes a cada uno de los **2^8 posibles valores de entrada** en la sustitución. Y, para obtener su valor correspondiente, basta con separar el byte de entrada en **4 bits más significativos para la fila** y los **4 bits menos significativos para la columna**, de modo que con ambos valores se obtiene la posición concreta en la matriz de sustitución de la S-box el byte correspondiente a su transformación afín (directa o inversa).

Una vez ya se ha explicado en qué consisten ambas S-boxes, vamos a centrarnos en lo que se nos pide en este apartado, que es estudiar la no linealidad de ambas cajas. De manera semejante a como hicimos en el punto 3, lo que hemos hecho es

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

construir un programa que pruebe combinaciones de todas las entradas posibles, que son **$2^8 = 256$ entradas** como hemos dicho antes.

Por lo tanto, como ya dijimos, se cumple linealidad para dos x e y entradas concretas si **S-box(x XOR y) = S-box(x) XOR S-box(y)**, por lo que nuestro programa cada vez que detecte una combinación de entradas que lo cumpla aumentará en uno la frecuencia de linealidad de la caja correspondiente. Por lo que, al final de comprobar las **256*256 combinaciones** posibles para cada una de los dos cajas de sustitución, se divide por esa cantidad y se aplica el inverso, es decir, **1 - P(L(S-box))** para obtener la probabilidad de no linealidad de la caja, que sería **P(NL(S-box))**. Falta indicar que estas S-boxes del AES están diseñadas de manera que sean no lineales en la mayoría de casos, ya que habrá ocasiones en las que sí se cumpla, debido a que generar una caja de sustitución completamente no lineal y funcional es algo prácticamente muy difícil.

Entonces, el programa consta simplemente de dos bucles anidados de 256 que generan valores de entrada x e y, con los que se realizarán las combinaciones XOR antes y después de hacer la sustitución en cada una de las dos S-boxes. Y, en la función “calcula_sbox()” se realiza la división de los bits del byte de entrada para obtener la posición del byte correspondiente a su transformación afín dentro de la tabla de la S-box. Entonces, con estas frecuencias y la probabilidad calculada, simplemente **se imprime por pantalla el resultado para cada caja** (además de otros mensajes indicando cuando se cumple la linealidad en cada caja con las entradas causantes de ello).

Por lo tanto, hemos ejecutado el programa y el resultado de las probabilidades de no linealidad de la caja directa y la inversa del AES ha sido:

```
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/p2
$ ./linealidad aes
Se cumple la linealidad en la S-box directa del AES para los valores x=3 e y=84
Se cumple la linealidad en la S-box directa del AES para los valores x=3 e y=87
Se cumple la linealidad en la S-box inversa del AES para los valores x=3 e y=169
Se cumple la linealidad en la S-box inversa del AES para los valores x=3 e y=170
Se cumple la linealidad en la S-box inversa del AES para los valores x=5 e y=57
Se cumple la linealidad en la S-box inversa del AES para los valores x=5 e y=60
Se cumple la linealidad en la S-box directa del AES para los valores x=6 e y=224
Se cumple la linealidad en la S-box directa del AES para los valores x=6 e y=230
Se cumple la linealidad en la S-box directa del AES para los valores x=7 e y=17
Se cumple la linealidad en la S-box directa del AES para los valores x=7 e y=22
Se cumple la linealidad en la S-box directa del AES para los valores x=8 e y=23
Se cumple la linealidad en la S-box directa del AES para los valores x=8 e y=31
Se cumple la linealidad en la S-box inversa del AES para los valores x=8 e y=132
Se cumple la linealidad en la S-box inversa del AES para los valores x=8 e y=140
Se cumple la linealidad en la S-box inversa del AES para los valores x=10 e y=164
Se cumple la linealidad en la S-box inversa del AES para los valores x=10 e y=174
Se cumple la linealidad en la S-box directa del AES para los valores x=10 e y=193
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
Se cumple la linealidad en la S-box directa del AES para los valores x=250 e y=129
Se cumple la linealidad en la S-box inversa del AES para los valores x=251 e y=101
Se cumple la linealidad en la S-box inversa del AES para los valores x=251 e y=158
Se cumple la linealidad en la S-box directa del AES para los valores x=253 e y=71
Se cumple la linealidad en la S-box directa del AES para los valores x=253 e y=186
Se cumple la linealidad en la S-box inversa del AES para los valores x=254 e y=65
Se cumple la linealidad en la S-box inversa del AES para los valores x=254 e y=191

La S-box directa del AES tiene frecuencia de no linealidad 0.996155
La S-box inversa del AES tiene frecuencia de no linealidad 0.996155

X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/p2
$
```

Podemos ver que el valor de la probabilidad de no linealidad de cada una de las cajas es la misma, concretamente es de un **99.6155%**, un valor bastante alto y deseado en el diseño de las cajas del AES, ya que la probabilidad de que se rompa la linealidad es muy baja. No creamos un histograma porque ambas tienen la misma y son solo dos cajas.

Este valor es el mismo en ambas S-boxes debido a que la caja directa es invertible, y corresponde a la matriz inversa. Es decir, **S-boxInversa(S-boxDirecta(b)) = b**. Esto deriva a que realmente posean un igual número de combinaciones de entradas en los que se cumple la linealidad.

B. Generación de las S-boxes AES

En este apartado se nos pide construir las S-boxes del AES de manera manual y compararlas con las reales para ver si se han calculado correctamente. Para este apartado hemos creado **SBOX_AES.c** con el código correspondiente a las funciones y **SBOX_AES.h** que sería la librería de todas ellas. La ejecución del ejecutable es de la siguiente forma:

> ./SBOX_AES {-C|-D} {-o fichero_salida}

donde hay que introducir dos argumentos de entrada obligatorios. El primero es el que indica cuál de las dos S-boxes del AES se desea generar, siendo -C para la directa y -D para la inversa. El otro argumento obligatorio es el fichero de salida donde se volcará la matriz de la caja de sustitución construida.

Pero antes, se nos pide hacer los algoritmos de **Euclides** y **Euclides Extendido** en **GF(2^8)** del AES, ya que vamos a necesitarlos para hallar el inverso multiplicativo de cada uno de los valores de entrada posibles. AES trabaja en el espacio binario, por lo que ahora los valores son realmente polinomios (a diferencia de la primera práctica), que como mucho pueden ser de grado 7 (correspondiente a un byte), ya que el polinomio irreducible del AES es de grado 8, de tal manera que corresponde a $m(x) = x^8 + x^4 + x^3 + x + 1$ (que equivale a 0x011B en hexadecimal).

Con esto dicho, hemos creado el método “calcular_inverso_multiplicativo()” que aplica ambos algoritmos dentro, de tal manera que con **Euclides**, el que obtiene el

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

máximo común divisor entre dos polinomios, se comprueba que el polinomio de entrada sea coprimo con $m(x)$ (que supuestamente lo tiene que ser, ya que $m(x)$ es irreducible) cumpliendo, como ya sabemos, que $\text{mcd}(m(x), p) = 1$, indicando que el polinomio posee inverso multiplicativo en $\text{GF}(2^8)$. Y, tras esto se procede a aplicar **Euclides Extendido**, que se encarga de hallar este inverso multiplicativo mediante recursión de las etapas obtenidas en la factorización de los restos en Euclides, donde se cumple al final que $1 = (p^{-1}) * p \% m(x)$, siendo p^{-1} el inverso que buscamos.

Para hacer ambos hemos tenido que crear una función que se encargue de las **divisiones de polinomios**, ya que no es equivalente a dividir un byte entre otro sin más, como hemos visto en teoría. Por lo tanto, hemos creado “dividir_m_polinomio()” que divide el irreducible $m(x)$ entre un polinomio p indicado (ya que este se sale de un byte hemos creado esta función aparte); y la función “dividir_polinomios()” para la división entre dos polinomios dados dentro del espacio de $\text{GF}(2^8)$. Ambas funciones devuelven el polinomio del resto y el polinomio del cociente de la operación.

Pero estas no son las únicas, también hemos creado “producto_polinomios()” necesario en la recursión de Euclides Extendido, que equivale a un algoritmo que aplica la **multiplicación entre dos polinomios** utilizando **xtime**, como hemos visto en teoría, de manera que es muy eficiente. La función “xtime()” es la encargada de obtener el xtime de un polinomio. Y, otra función creada para esta parte ha sido “calcula_grado()” que indica, simplemente, cual es el grado de un polinomio (siempre por debajo de 8).

Con todo esto, se ha implementado correctamente cada uno de estos dos algoritmos para la obtención del inverso multiplicativo del polinomio. Por lo tanto, ya podemos hablar del **esquema general** del programa. Lo que se realiza es un bucle de **256** para generar todos los valores posibles de **entradas** en la caja de sustitución del AES y poder calcular el byte correspondiente para la caja. Y, si se está generando la **caja directa**, se procede a calcular el **inverso multiplicativo** de la entrada con la función anteriormente nombrada y luego se le aplica la **transformación afín** (como se ha explicado en el apartado A). En el caso de la **caja inversa**, primero se aplica la **transformación afín inversa** y al resultado se le halla el **inverso multiplicativo**.

Por lo tanto, lo único que falta es una función que calcule la transformación afín directa y otra para la inversa. Para la **transformación afín directa** hemos creado el método “transformacion_afin()”, que aplica la fórmula explicada en el apartado anterior, donde $y = X * b + C$ (siendo b un polinomio/byte de entrada). Esta fórmula equivale a lo siguiente, que es lo que hemos aplicado en la función al ser más

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

eficiente: $y = b \text{ XOR } (b \ll 1) \text{ XOR } (b \ll 2) \text{ XOR } (b \ll 3) \text{ XOR } (b \ll 4) \text{ XOR } C$ donde \ll es un desplazamiento a la izquierda circular y C un polinomio constante que equivale a 0x63 en hexadecimal. Este desplazamiento circular lo hacemos en el método “desplaza_izquierda_circular()”, indicando cuánto se desplaza el polinomio. Y, para la transformación afín inversa hemos creado la función “transformacion_afin_inversa()”, que aplica la otra fórmula donde $y = Y*b + D$. Esta fórmula equivale, como antes, a otra ecuación más eficiente que es: $b \text{ XOR } (b \ll 1) \text{ XOR } (b \ll 3) \text{ XOR } (b \ll 6) \text{ XOR } D$ donde D es un polinomio constante que equivale a 0x05 en hexadecimal.

Una vez tenemos la transformación afín directa e inversa, el programa ya está completo. Lo que falta es hallar la posición de la matriz de la S-box donde se va a introducir el valor hallado. Para esto utilizamos las funciones “fila()” y “columna()” que hallan la fila mirando los 4 primeros bits del polinomio/byte inicial (el del principio sin haber aplicado nada) y la columna mirando los 4 últimos bits. Entonces, se modifica una estructura, reservada al inicio, llamada “matriz” con todos estos valores y al final se imprime en el fichero de salida indicado.

Falta indicar que se ha supuesto que el polinomio 0x00 es sí mismo inverso, ya que no se puede calcular, dando al final el valor 0x63 con la transformación (e igual de manera inversa al 0x63 le corresponde el byte 0x00).

A continuación, se va a ejecutar el programa dos veces, una para generar la caja de sustitución directa y otra para la inversa, de manera que:

```
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/p2
$ ./SBOX_AES -C -o directa.txt
Generando S-box Directa del AES...
Memoria liberada correctamente...
```

```
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/p2
$ cat directa.txt -
63 7c 77 7b f2 6b 6f c5 30 1 67 2b fe d7 ab 76
ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 c0
b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15
4 c7 23 c3 18 96 5 9a 7 12 80 e2 eb 27 b2 75
9 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 2f 84
53 d1 0 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 cf
d0 ef aa fb 43 4d 33 85 45 f9 2 7f 50 3c 9f a8
51 a3 40 8f 92 9d 38 f5 bc b6 da 21 10 ff f3 d2
cd c 13 ec 5f 97 44 17 c4 a7 7e 3d 64 5d 19 73
60 81 4f dc 22 2a 90 88 46 ee b8 14 de 5e b db
e0 32 3a a 49 6 24 5c c2 d3 ac 62 91 95 e4 79
e7 c8 37 6d 8d d5 4e a9 6c 56 f4 ea 65 7a ae 8
ba 78 25 2e 1c a6 b4 c6 e8 dd 74 1f 4b bd 8b 8a
70 3e b5 66 48 3 f6 e 61 35 57 b9 86 c1 1d 9e
e1 f8 98 11 69 d9 8e 94 9b 1e 87 e9 ce 55 28 df
8c a1 89 d bf e6 42 68 41 99 2d f b0 54 bb 16
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/p2
$ ./SBOX_AES -D -o inversa.txt
Generando S-box Inversa del AES...

Memoria liberada correctamente...

X541@LAPTOP-DanMat27 /c/users/x541/desktop/cripto-master/p2
```

```
$ cat inversa.txt
52 9 6a d5 30 36 a5 38 bf 40 a3 9e 81 f3 d7 fb
7c e3 39 82 9b 2f ff 87 34 8e 43 44 c4 de e9 cb
54 7b 94 32 a6 c2 23 3d ee 4c 95 b 42 fa c3 4e
8 2e a1 66 28 d9 24 b2 76 5b a2 49 6d 8b d1 25
72 f8 f6 64 86 68 98 16 d4 a4 5c cc 5d 65 b6 92
6c 70 48 50 fd ed b9 da 5e 15 46 57 a7 8d 9d 84
90 d8 ab 0 8c bc d3 a f7 e4 58 5 b8 b3 45 6
d0 2c 1e 8f ca 3f f 2 c1 af bd 3 1 13 8a 6b
3a 91 11 41 4f 67 dc ea 97 f2 cf ce f0 b4 e6 73
96 ac 74 22 e7 ad 35 85 e2 f9 37 e8 1c 75 df 6e
47 f1 1a 71 1d 29 c5 89 6f b7 62 e aa 18 be 1b
fc 56 3e 4b c6 d2 79 20 9a db c0 fe 78 cd 5a f4
1f dd a8 33 88 7 c7 31 b1 12 10 59 27 80 ec 5f
60 51 7f a9 19 b5 4a d 2d e5 7a 9f 93 c9 9c ef
a0 e0 3b 4d ae 2a f5 b0 c8 eb bb 3c 83 53 99 61
17 2b 4 7e ba 77 d6 26 e1 69 14 63 55 21 c 7d
```

Parece que se han generado correctamente (se ve desfasado porque hay bytes que poseen un 0 a la izquierda, por lo que no se imprime ese 0 y queda así la fila). Si comparamos con las siguientes S-boxes del fichero “aes_tables.h” se puede observar que ambas cajas de sustitución han sido generadas de manera correcta, al ser iguales.

```
7 static const char DIRECT_SBOX[ROWS_PER_SBOX][COLUMNS_PER_SBOX] = {
8     { 0x63 , 0x7c , 0x77 , 0x7b , 0xf2 , 0x6b , 0x6f , 0xc5 , 0x30 , 0x01 , 0x67 , 0x2b , 0xfe , 0xd7 , 0xab , 0x76 },
9     { 0xca , 0x82 , 0xc9 , 0x7d , 0xfa , 0x59 , 0x47 , 0xf0 , 0xad , 0xd4 , 0xa2 , 0xaf , 0x9c , 0xa4 , 0x72 , 0xc0 },
10    { 0xb7 , 0xfd , 0x93 , 0x26 , 0x36 , 0x3f , 0xf7 , 0xcc , 0x34 , 0xa5 , 0xe5 , 0xf1 , 0x71 , 0xd8 , 0x31 , 0x15 },
11    { 0x04 , 0xc7 , 0x23 , 0xc3 , 0x18 , 0x96 , 0x05 , 0x9a , 0x07 , 0x12 , 0x80 , 0xe2 , 0xeb , 0x27 , 0xb2 , 0x75 },
12    { 0x09 , 0x83 , 0x2c , 0x1a , 0x1b , 0x6e , 0x5a , 0xa0 , 0x52 , 0x3b , 0xd6 , 0xb3 , 0x29 , 0xe3 , 0x2f , 0x84 },
13    { 0x53 , 0xd1 , 0x00 , 0xed , 0x20 , 0xfc , 0xb1 , 0x5b , 0x6a , 0xcb , 0xbe , 0x39 , 0x4a , 0x4c , 0x58 , 0xcf },
14    { 0xd0 , 0xef , 0xaa , 0xfb , 0x43 , 0x4d , 0x33 , 0x85 , 0x45 , 0xf9 , 0x02 , 0x7f , 0x50 , 0x3c , 0x9f , 0xa8 },
15    { 0x51 , 0xa3 , 0x40 , 0x8f , 0x92 , 0x9d , 0x38 , 0xf5 , 0xbc , 0xb6 , 0xda , 0x21 , 0x10 , 0xff , 0xf3 , 0xd2 },
16    { 0xcd , 0x0c , 0x13 , 0xec , 0x5f , 0x97 , 0x44 , 0x17 , 0xc4 , 0xa7 , 0x7e , 0x3d , 0x64 , 0xd5 , 0x19 , 0x73 },
17    { 0x60 , 0x81 , 0x4f , 0xdc , 0x22 , 0x2a , 0x90 , 0x88 , 0x46 , 0xee , 0xb8 , 0x14 , 0xde , 0x5e , 0x0b , 0xdb },
18    { 0xe0 , 0x32 , 0x3a , 0xa0 , 0x49 , 0x06 , 0x24 , 0x5c , 0xc2 , 0xd3 , 0xac , 0x62 , 0x91 , 0x95 , 0xe4 , 0x79 },
19    { 0xe7 , 0xc8 , 0x37 , 0x6d , 0x8d , 0xd5 , 0x4e , 0xa9 , 0x6c , 0x56 , 0xf4 , 0xea , 0x65 , 0x7a , 0xae , 0x08 },
20    { 0xba , 0x78 , 0x25 , 0x2e , 0x1c , 0xa6 , 0xb4 , 0xc6 , 0xe8 , 0xdd , 0x74 , 0x1f , 0x4b , 0xbd , 0x8b , 0x8a },
21    { 0x70 , 0x3e , 0xb5 , 0x66 , 0x48 , 0x03 , 0xf6 , 0x0e , 0x61 , 0x35 , 0x57 , 0xb9 , 0x86 , 0xc1 , 0x1d , 0x9e },
22    { 0xe1 , 0xf8 , 0x98 , 0x11 , 0x69 , 0xd9 , 0x8e , 0x94 , 0x9b , 0x1e , 0x87 , 0xe9 , 0xce , 0x55 , 0x28 , 0xdf },
23    { 0x8c , 0xa1 , 0x89 , 0xd0 , 0xbf , 0xe6 , 0x42 , 0x68 , 0x41 , 0x99 , 0x2d , 0x0f , 0xb0 , 0x54 , 0xbb , 0x16 }
```

Laura Sánchez Herrera

Daniel Mateo Moreno

Pareja: 5

```
25
26 static const char INVERSE_SBOX[ROWS_PER_SBOX][COLUMNS_PER_SBOX] = {
27 { 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb },
28 { 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb },
29 { 0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e },
30 { 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25 },
31 { 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92 },
32 { 0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84 },
33 { 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0xa0, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06 },
34 { 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b },
35 { 0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73 },
36 { 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e },
37 { 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b },
38 { 0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4 },
39 { 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f },
40 { 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef },
41 { 0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61 },
42 { 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d }
43 };
44
```