

# Criptografía

Cristina Kasner Tourné  
Jose Antonio García del Saz

Curso 2015 - 2015 C1

# Índice general

<b>A Ejercicios</b>	<b>2</b>
A.1 Ejercicio 1-Seguridad perfecta . . . . .	2
A.2 Ejercicio 2-Implementación del DES . . . . .	5
A.3 Ejercicio X-Estudiar la linealidad de las cajas-S del AES . . . . .	6
A.4 Ejercicio X - SAC y BIC para las cajas-S del DES . . . . .	6

# Apéndice A

## Ejercicios

### A.1. Ejercicio 1-Seguridad perfecta

En este ejercicio nos piden implementar un programa que compruebe la seguridad perfecta.

Recordamos la definición de seguridad perfecta:

*Seguridad  
Perfecta*

**Definición A.1.1 Seguridad Perfecta.** Decimos que un criptosistema tiene seguridad perfecta si cumple:

$$P_p(x|y) = P_p(x)$$

Esto quiere decir que el conocimiento de texto cifrado no nos da ninguna información sobre el texto plano.

Para esto hemos creado un fichero *probabilidad.c* en el que implementamos las funciones que calculan esas probabilidades.

La probabilidad  $P_p(x)$  la calculamos recorriendo el fichero que tiene el texto plano y llevando la cuenta de las veces que aparece la letra *i*.

```
while ((l=fgetc(f))!=EOF){
    prob[l-65]++;
    longText++;
}
```

Luego dividimos ese número entre la longitud del texto.

La probabilidad  $P_p(x|y)$  la calculamos de la misma forma solo que contabilizando a la vez los caracteres del mensaje en plano y el mensaje cifrado.

```
while ((c=fgetc(cifrado))!=EOF){
    p = fgetc(plano);
    prob[p-65][c-65]++;
    cantLetra[c-65]++;
}
```

Y finalmente dividimos entre el número de veces que aparece en el texto cifrado la letra correspondiente.

En la práctica podemos probar la seguridad perfecta con dos casos distintos.

- claves equiprobables
- claves no equiprobables

Para generar las claves equiprobables utilizamos la función random de C.

Para las claves no equiprobables hemos escrito el siguiente código:

```
for (i=0; i < 26; i++){  
    if ((clave > m/2) == 0){  
        clave = rand() % m;  
    } else break;  
}
```

De esta forma es mucho más probable que mi clave sea una clave menor que  $m/2$  que mayor.

Además obligamos a que la b también pertenezca a las unidades de m.

Los resultados obtenidos tras probar el código son:

```
Pp(A):0.089453  
Pp(B):0.012847  
Pp(C):0.039916  
Pp(D):0.034419  
Pp(E):0.117359  
Pp(F):0.014580  
Pp(G):0.015297  
Pp(H):0.021631  
Pp(I):0.091007  
Pp(J):0.002330  
Pp(K):0.002510  
Pp(L):0.047625  
Pp(M):0.035733  
Pp(N):0.070511  
Pp(O):0.062563  
Pp(P):0.025037  
Pp(Q):0.005258  
Pp(R):0.063579  
Pp(S):0.071646  
Pp(T):0.087302  
Pp(U):0.054198  
Pp(V):0.015477  
Pp(W):0.007708  
Pp(X):0.002091  
Pp(Y):0.008306  
Pp(Z):0.001613
```

## Claves equiprobables

Pp(A A):0.100671	Pp(E A):0.100671	Pp(H A):0.015101
Pp(A B):0.081538	Pp(E B):0.129231	Pp(H B):0.018462
Pp(A C):0.103989	Pp(E C):0.109687	Pp(H C):0.017094
Pp(A D):0.098101	Pp(E D):0.132911	Pp(H D):0.020570
Pp(A E):0.073052	Pp(E E):0.112013	Pp(H E):0.024351
Pp(A F):0.108626	Pp(E F):0.116613	Pp(H F):0.017572
Pp(A G):0.077147	Pp(E G):0.128093	Pp(H G):0.018923
Pp(A H):0.110577	Pp(E H):0.104167	Pp(H H):0.020833
Pp(A I):0.094656	Pp(E I):0.116031	Pp(H I):0.027481
Pp(A J):0.078947	Pp(E J):0.120743	Pp(H J):0.013932
Pp(A K):0.096273	Pp(E K):0.111801	Pp(H K):0.026398
Pp(A L):0.070175	Pp(E L):0.110048	Pp(H L):0.014354
Pp(A M):0.093415	Pp(E M):0.116386	Pp(H M):0.022971
Pp(A N):0.099849	Pp(E N):0.122542	Pp(H N):0.031770
Pp(A O):0.077419	Pp(E O):0.124194	Pp(H O):0.030645
Pp(A P):0.083333	Pp(E P):0.100000	Pp(H P):0.022727
Pp(A Q):0.083871	Pp(E Q):0.122581	Pp(H Q):0.020968
Pp(A R):0.084084	Pp(E R):0.100601	Pp(H R):0.013514
Pp(A S):0.092476	Pp(E S):0.109718	Pp(H S):0.026646
Pp(A T):0.114600	Pp(E T):0.113030	Pp(H T):0.018838
Pp(A U):0.070444	Pp(E U):0.110260	Pp(H U):0.027565
Pp(A V):0.080247	Pp(E V):0.132716	Pp(H V):0.029321
Pp(A W):0.102326	Pp(E W):0.119380	Pp(H W):0.026357
Pp(A X):0.084530	Pp(E X):0.130781	Pp(H X):0.014354
Pp(A Y):0.101562	Pp(E Y):0.131250	Pp(H Y):0.025000
Pp(A Z):0.064955	Pp(E Z):0.125378	Pp(H Z):0.016616

En estos ejemplos podemos ver que las probabilidades condicionadas son muy parecidas y todas tienen un valor muy cercano al de la probabilidad de la letra.

Una de las razones por las que el resultado no es exacto es que la función rand() de C no da resultados exactamente equiprobables.

## Claves no equiprobables

Pp(A A):0.013889	Pp(E A):0.009752	Pp(H A):0.062352
Pp(A B):0.123324	Pp(E B):0.144772	Pp(H B):0.029491
Pp(A C):0.142105	Pp(E C):0.110526	Pp(H C):0.010526
Pp(A D):0.147583	Pp(E D):0.134860	Pp(H D):0.012723
Pp(A E):0.162791	Pp(E E):0.063953	Pp(H E):0.023256
Pp(A F):0.134146	Pp(E F):0.109756	Pp(H F):0.009756
Pp(A G):0.119792	Pp(E G):0.106771	Pp(H G):0.013021
Pp(A H):0.118734	Pp(E H):0.129288	Pp(H H):0.021108
Pp(A I):0.152672	Pp(E I):0.111959	Pp(H I):0.015267
Pp(A J):0.086253	Pp(E J):0.156334	Pp(H J):0.010782
Pp(A K):0.153439	Pp(E K):0.097884	Pp(H K):0.013228
Pp(A L):0.128266	Pp(E L):0.118765	Pp(H L):0.026128
Pp(A M):0.141361	Pp(E M):0.073298	Pp(H M):0.018325
Pp(A N):0.150115	Pp(E N):0.195465	Pp(H N):0.001050
Pp(A O):0.000000	Pp(E O):0.132686	Pp(H O):0.012945
Pp(A P):0.060345	Pp(E P):0.135057	Pp(H P):0.011494
Pp(A Q):0.000000	Pp(E Q):0.097902	Pp(H Q):0.010490
Pp(A R):0.054545	Pp(E R):0.127273	Pp(H R):0.025974
Pp(A S):0.000000	Pp(E S):0.112676	Pp(H S):0.024648
Pp(A T):0.060209	Pp(E T):0.107330	Pp(H T):0.010471
Pp(A U):0.000000	Pp(E U):0.099656	Pp(H U):0.024055
Pp(A V):0.067385	Pp(E V):0.161725	Pp(H V):0.024259
Pp(A W):0.000000	Pp(E W):0.121212	Pp(H W):0.026515
Pp(A X):0.039216	Pp(E X):0.117647	Pp(H X):0.012255
Pp(A Y):0.000000	Pp(E Y):0.107744	Pp(H Y):0.016835
Pp(A Z):0.030986	Pp(E Z):0.107042	Pp(H Z):0.008451

Vemos como estos resultados se distan mucho más que cuando utilizabamos claves probables.

## A.2. Ejercicio 2-Implementación del DES

Para implementar el DES hemos creado un fichero que se llama *funcionesDES.c* en el que están todas las funciones necesarias para el método.

Como sabemos el DES funciona a través de permutaciones y sustituciones (cajas-S), por esto no es ninguna sorpresa que el grueso de las funciones de *funcionesDES.c* sean permutaciones y sustituciones.

Vamos a explicar ambas.

### ■ Permutaciones

La idea de las funciones de permutación es la siguiente:

Guardo el número que leo en la matriz de permutación, que es la posición del bit que va a ir en esa posición.

Miro si ese bit es un 0 (`positions[bit%8]` tiene un 1 en el bit que estoy mirando).

Si es un 0 no hago nada ya que he inicializado `permutation` a 0.

Si no es un cero meto en `desired bit` un 1 en la posición apropiada y hago un XOR con el byte que ya hubiera en `permutation`, de forma que solo cambio el bit deseado.

Adjuntamos el código de una de las funciones de permutación para que se entienda mejor.

```
int bit, newpos;
unsigned char desiredbit;
for (bit = 0; bit < 56; bit++) {

    newpos = ((int)PC1[bit]) - 1;
    desiredbit = input[newpos/8] & Positions[newpos
        %8];
    if (desiredbit != 0) {
        desiredbit = Positions[bit%8];
        permutation[bit/8] = desiredbit ^
            permutation[bit/8];
    }
}
```

### ■ Sustituciones → Cajas-S

En esta función básicamente vamos rotando los bits para obtener la fila y la columna y obtenemos el resultado de las cajas-S de la siguiente forma:

```
if ((caja % 2) == 0) {
    aux = S_BOXES[caja][posrow][poscolaux] << 4;
} else {
    output[caja/2] = S_BOXES[caja][posrow][poscolaux];
    output[caja/2] = output[caja/2] ^ aux;
    aux = 0;
}
```

Esto es porque nuestra función devuelve una cadena de caracteres, cada caracter son 8 bits pero las cajas-S devuelven 4 bits por lo que vamos guardando los resultados de dos en dos.

Para encriptar se tiene que ejecutar el código tal y como indica el enunciado.

La clave para desencriptar aparece por terminar cuando encriptas.

Vamos a ver un ejemplo:

```
→ P2 git:(master) X ./desECB -C -S 123 -i cifraDES.png -o imagencif.png
Su clave generada aleatoriamente en hexadecimal es:71e759d614fdc861
→ P2 git:(master) X ./desECB -D -k 71e759d614fdc861 -S 123 -i imagencif.png -o imagenDescifrada.png
```

### A.3. Ejercicio X-Estudiar la linealidad de las cajas-S del AES

Recordamos que una función lineal es aquella función  $f$  que cumple que :

$$f(a + b) = f(a) + f(b)$$

Es sencillo comprobar que las cajas-S no cumplen esto. Hemos creado un sencillo programa al que le pasas dos bytes  $a$  ,  $b$  y aplicamos la caja-S a cada uno de ellos y sumamos los resultados.

A su vez el programa suma  $a + b$  y aplica la caja-S a dicha suma y vemos que los resultados son distintos.

La no linealidad de las cajas-S del AES es muy importante ya que las hacen resistentes al criptoanálisis lineal.

### A.4. Ejercicio X - SAC y BIC para las cajas-S del DES

Primero vamos a explicar qué es cada uno de estos principios.

- **SAC** (Strict avalanche criterion)

Este principio dice que la probabilidad de cambio debe estar equidistribuida.

Esto es, que si cambio un bit de entrada, la probabilidad de que un bit de salida sea 0 o 1 es la misma.

$$\forall i, j \quad P(c_i = 1 | \overline{b_j}) = P(c_i = 0 | \overline{b_j}) = \frac{1}{2}$$

Siendo  $\overline{b_j}$  que he cambiado el bit  $b_j$

- **BIC** (bit independence criterion)

Busca que no haya dependencia entre los bits de salida (si cambia  $c_3$ , no condiciona a que cambie o no  $c_2$ )

$$\forall i, j, k \ P(c_i c_j | \bar{b}_k) = P(c_i | \bar{b}_k) \cdot P(c_j | \bar{b}_k)$$

Para comprobar esto hemos hecho un programa que cuenta la probabilidad de 0 y 1 de cada bit de salida de las cajas-S para las 64 posibles entradas.

Vemos que las probabilidades son todas 0.5, por lo que **se cumple el criterio SAC**.

Para BIC no comprobamos que  $P(c_i c_j | \bar{b}_k) = P(c_i | \bar{b}_k) \cdot P(c_j | \bar{b}_k)$ , sino que

$$P(c_i c_j c_k c_l | \bar{b}_k) = P(c_i | \bar{b}_k) \cdot P(c_j | \bar{b}_k) \cdot P(c_k | \bar{b}_k) \cdot P(c_l | \bar{b}_k)$$

Para esto calculamos  $P(c_i c_j c_k c_l | \bar{b}_k)$ , que nos sale 0.0625.

Las probabilidades  $P(c_i | \bar{b}_k)$  ya las habíamos calculado para el SAC y habíamos visto que son todas iguales, por lo que si hacemos  $(0.5)^4$  vemos que nos da 0.0625.

Por lo que queda comprobado que también cumple BIC.

Adjuntamos dos pantallazos de la salida. En la imagen de la izquierda vemos las probabilidades de cada bit de salida de ser 0 o 1.

En la imagen de la derecha vemos las probabilidades  $P(c_i c_j c_k c_l | \bar{b}_j)$

```
prob0 [0]: 0.500000
prob1 [0]: 0.500000
prob0 [1]: 0.500000
prob1 [1]: 0.500000
prob0 [2]: 0.500000
prob1 [2]: 0.500000
prob0 [3]: 0.500000
prob1 [3]: 0.500000
```

```
probs[0] :0.062500
probs[1] :0.062500
probs[2] :0.062500
probs[3] :0.062500
probs[4] :0.062500
probs[5] :0.062500
probs[6] :0.062500
probs[7] :0.062500
probs[8] :0.062500
probs[9] :0.062500
probs[10] :0.062500
probs[11] :0.062500
probs[12] :0.062500
probs[13] :0.062500
probs[14] :0.062500
probs[15] :0.062500
```



# Índice alfabético

Seguridad Perfecta, [2](#)