Criptografía

Cristina Kasner Tourné
Jose Antonio García del Saz

Curso 2015 - 2015 C1

Índice general

[0] Documento compilado el 25 de noviembre de 2015 a las 23:36

Capítulo I

Ejercicios

I.1. Ejercicio 1-Seguridad perfecta

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

Recordamos la definición de seguridad perfecta:

Seguridad Perfecta

Definición I.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 implemetamos 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 longitus 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<2;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(A|B):0.081538
Pp(A|C):0.103989
Pp(A|D):0.098101
Pp(A|E):0.073052
Pp(A|F):0.108626
Pp(A|G):0.077147
Pp(A|H):0.110577
Pp(A|I):0.094656
Pp(A|J):0.078947
Pp(A|K):0.096273
Pp(A|L):0.070175
Pp(A|M):0.093415
Pp(A|N):0.099849
Pp(A|O):0.077419
Pp(A|P):0.083333
Pp(A|Q):0.083871
Pp(A|R):0.084084
Pp(A|S):0.092476
Pp(A|T):0.114600
Pp(A|U):0.070444
Pp(A|V):0.080247
Pp(A|W):0.102326
Pp(A|X):0.084530
Pp(A|Y):0.101562
Pp(A|Z):0.064955
```

```
Pp(E|A):0.100671
Pp(E|B):0.129231
Pp(E|C):0.109687
Pp(E|D):0.132911
Pp(E|E):0.112013
Pp(E|F):0.116613
Pp(E|G):0.128093
Pp(E|H):0.104167
Pp(E|I):0.116031
Pp(E|J):0.120743
Pp(E|K):0.111801
Pp(E|L):0.110048
Pp(E|M):0.116386
Pp(E|N):0.122542
Pp(E|O):0.124194
Pp(E|P):0.100000
Pp(E|Q):0.122581
Pp(E|R):0.100601
Pp(E|S):0.109718
Pp(E|T):0.113030
Pp(E|U):0.110260
Pp(E|V):0.132716
Pp(E|W):0.119380
Pp(E|X):0.130781
Pp(E|Y):0.131250
Pp(E|Z):0.125378
```

```
Pp(H|A):0.015101
Pp(H|B):0.018462
Pp(H|C):0.017094
Pp(H|D):0.020570
Pp(H|E):0.024351
Pp(H|F):0.017572
Pp(H|G):0.018923
Pp(H|H):0.020833
Pp(H|I):0.027481
Pp(H|J):0.013932
Pp(H|K):0.026398
Pp(H|L):0.014354
Pp(H|M):0.022971
Pp(H|N):0.031770
Pp(H|O):0.030645
Pp(H|P):0.022727
Pp(H|Q):0.020968
Pp(H|R):0.013514
Pp(H|S):0.026646
Pp(H|T):0.018838
Pp(H|U):0.027565
Pp(H|V):0.029321
Pp(H|W):0.026357
Pp(H|X):0.014354
Pp(H|Y):0.025000
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(A|B):0.123324
Pp(A|C):0.142105
Pp(A|D):0.147583
Pp(A|E):0.162791
Pp(A|F):0.134146
Pp(A|G):0.119792
Pp(A|H):0.118734
Pp(A|I):0.152672
Pp(A|J):0.086253
Pp(A|K):0.153439
Pp(A|L):0.128266
Pp(A|M):0.141361
Pp(A|N):0.150115
Pp(A|O):0.000000
Pp(A|P):0.060345
Pp(A|Q):0.000000
Pp(A|R):0.054545
Pp(A|S):0.000000
Pp(A|T):0.060209
Pp(A|U):0.000000
Pp(A|V):0.067385
Pp(A|W):0.000000
Pp(A|X):0.039216
Pp(A|Y):0.000000
Pp(A|Z):0.030986
```

```
Pp(E|A):0.009752
Pp(E|B):0.144772
Pp(E|C):0.110526
Pp(E|D):0.134860
Pp(E|E):0.063953
Pp(E|F):0.109756
Pp(E|G):0.106771
Pp(E|H):0.129288
Pp(E|I):0.111959
Pp(E|J):0.156334
Pp(E|K):0.097884
Pp(E|L):0.118765
Pp(E|M):0.073298
Pp(E|N):0.195465
Pp(E|O):0.132686
Pp(E|P):0.135057
Pp(E|Q):0.097902
Pp(E|R):0.127273
Pp(E|S):0.112676
Pp(E|T):0.107330
Pp(E|U):0.099656
Pp(E|V):0.161725
Pp(E|W):0.121212
Pp(E|X):0.117647
Pp(E|Y):0.107744
Pp(E|Z):0.107042
```

```
Pp(H|A):0.062352
Pp(H|B):0.029491
Pp(H|C):0.010526
Pp(H|D):0.012723
Pp(H|E):0.023256
Pp(H|F):0.009756
Pp(H|G):0.013021
Pp(H|H):0.021108
Pp(H|I):0.015267
Pp(H|J):0.010782
Pp(H|K):0.013228
Pp(H|L):0.026128
Pp(H|M):0.018325
Pp(H|N):0.001050
Pp(H|O):0.012945
Pp(H|P):0.011494
Pp(H|Q):0.010490
Pp(H|R):0.025974
Pp(H|S):0.024648
Pp(H|T):0.010471
Pp(H|U):0.024055
Pp(H|V):0.024259
Pp(H|W):0.026515
Pp(H|X):0.012255
Pp(H|Y):0.016835
Pp(H|Z):0.008451
```

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

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

I.3. Ejercicio 3c y 4b -Estudiar la linealidad de las cajas-S del AES y del DES

Las cajas de sustitución (cajas S) constituyen la piedra angular en criptografía para lograr que los cifradores por bloque exhiban la ineludible propiedad de no linealidad.

En efecto, si la o las cajas S de un determinado cifrador por bloque no alcanzan una alta no linealidad, entonces se considera que tal algoritmo no podrá ofrecer una seguridad adecuada para impedir que información confidencial pueda ser develada por entidades no autorizadas.

Dada su definición, es claro que el número de funciones booleanas elegibles para diseñar una caja S de n bits de entrada y m bits de salida está dado por $2^{m \cdot 2^n}$, de tal manera que aun para valores moderados de n y m el tamaño del espacio de búsqueda de este problema tiene un tamaño desmesurado.

Sin embargo, no todas las funciones booleanas son apropiadas para construir buenas cajas S.

Además de la ya mencionada propiedad de no linealidad,algunas de las principales propiedades criptográficas requeridas para dichas funciones booleanas incluyen: balance, alto grado algebraico, criterio de avalancha estricto, orden de inmunidad, etcétera.

Durante el desarrollo de esta práctica hemos aprovechado las cajas S de los algoritmos DES y AES para crear dos rutinas en C que demuestran la no linealidad de estas funciones booleanas. Estas dos rutinas han sido llamadas **linealidadSBoxesDES** y **linealidadSBoxesAES**

Ambas rutinas reciben como argumento de ejecución "-n N", donde N es un número natural que indica el número de veces que se quiere probar la no linealidad. La salida de estos dos programas se incluye a continuación para N=10:

Para comprobar la nonlinealidad de las SBOXES en ambos casos, basta con fijarse en que se cumple la siguiente igualdad:

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

Donde K sería lo que llamaríamos constante de no linealidad (si fuese 0 la función sería lineal).

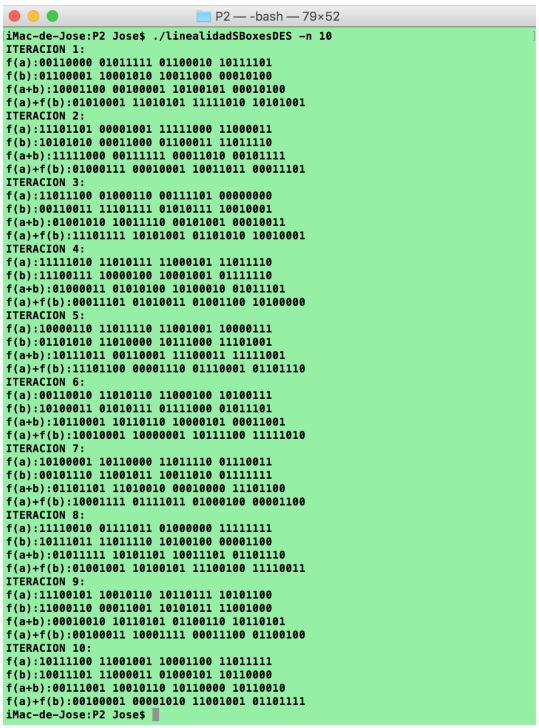


Figura I.1: *No linealidad del DES*

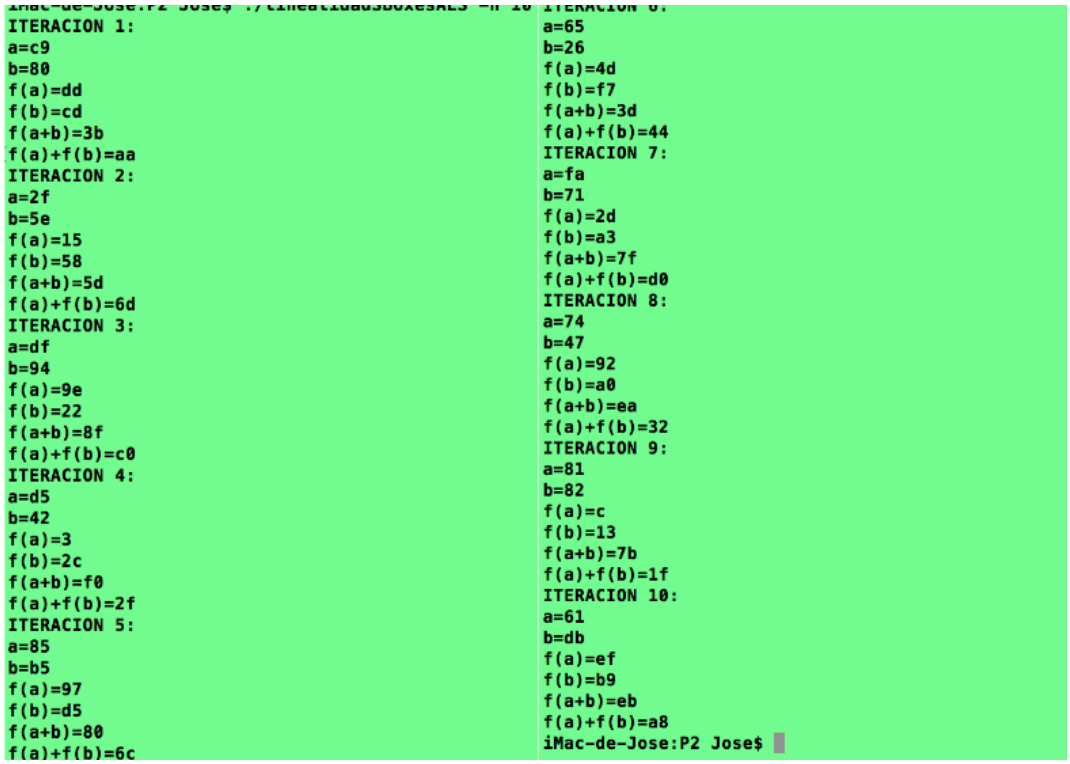


Figura I.2: *No linealidad del AES*

En resumen, los principales criterios para construir unas buenas S-Boxes son los siguientes:

- **Balance**:Esta propiedad es muy deseable para evitar ataques cripto-diferenciales tales como los introducidos por A. Shamir contra el algoritmo DES
- **Alta no linealidad**: Esta propiedad reduce el efecto de los ataques por cripto-análisis lineal. Como se discutió antes, la no linealidad de una función booleana puede ser calculada directamente de la transformada de Walsh-Hadamard
- **Autocorrelación**: Este valor es proporcional al desbalance de todas las derivadas de primer orden de la función booleana. Valores pequeños son considerados como buenos mientras que un valor grande es considerado un símbolo de debilidad. Las funciones curvas gozan de una autocorrelación mínima, por lo que optimizan esta propiedad.• Indicador absoluto: Indicador absoluto de una función booleana denotado por M(f) está dado por $|r_max|$ el máximo valor absoluto en $r_{\hat{f}}(s)$. Se considera que una función booleana con un M(f) pequeño es criptográficamente deseable. Nuevamente las funciones curvas son las mejores, ya que su indicador absoluto es 0.
- **Efecto avalancha**: Está relacionado con la autocorrelación y se define con respecto a un bit específico de entrada tal que al complementarlo resulta en un cambio en el bit de salida con una probabilidad de 1/2. El criterio de avalancha estricto (SAC por sus siglas en inglés), requiere los efectos avalancha de todos los bits de entrada. Se dice que una función booleana satisface el criterio de avalancha estricto si al complementar un solo bit de entrada resulta en un cambio en un bit de salida con una probabilidad de 1/2. Puede demostrarse fácilmente que una función booleana f con función de autocorrelación $r_{\hat{f}}(s)$, satisface el criterio de avalancha estricto si y sólo si $r_{\hat{f}}(s)$= 0 para toda s con peso de Hamming H(s)=1

I.4. Ejercicio 3a - SAC y BIC para las cajas-S del DES

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

- **SAC** (Strict avalanch 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.

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

  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)

  $$\forall i,j,k \ P(c_i c_j|\overline{b_k}) = P(c_i|\overline{b_k}) \cdot P(c_j|\overline{b_k})$$

Hemos hecho un programa que genera entradas aleatorias para lass cajas-S del DES. Recorremos cada una de esas entradas, cambiando uno a uno sus bits y vemos cómo reacciona la salida ante estos cambios.

Los resultados que aparecen por pantalla para comprobar el criterio SAC están en el siguiente formato:

$$prob0[i][j] = ...$$
$$prob1[i][j] = ...$$

Que son la probabilidad de que el bit i sea 0 o 1 si cambio el bit j.

Los resultados que nos dan son cercanos a 0.5 pero no son completamente satisfactorios.

```
prob0 [19][27]: 0.450000
prob1 [19][27]: 0.550000
prob0 [16][28]: 0.371000
prob1 [16][28]: 0.629000
prob0 [17][28]: 0.593000
prob1 [17][28]: 0.407000
prob0 [18][28]: 0.626000
prob1 [18][28]: 0.374000
prob0 [19][28]: 0.410000
prob1 [19][28]: 0.590000
prob0 [16][29]: 0.402000
prob1 [16][29]: 0.598000
prob0 [17][29]: 0.575000
prob1 [17][29]: 0.425000
prob0 [18][29]: 0.571000
prob1 [18][29]: 0.429000
prob0 [19][29]: 0.634000
prob1 [19][29]: 0.366000
```

El criterio BIC lo hemos calculado no como pone en la definición sino:

$$P(c_i c_j c_k c_l | \overline{b_k}) = P(c_i | \overline{b_k}) \cdot P(c_j | \overline{b_k}) \cdot P(c_k | \overline{b_k}) \cdot P(c_l | \overline{b_k})$$

El resultado lo escribimos en el formato $probsij[i][j] = ...$, donde i va de 0 a 15 , que son los posibles valores que puedo escribir con $c_i c_j c_k c_l$ y j va de 0 a 47, que son los bits de la entrada.

```
probbsij[0][16] = 0.064000
probbsij[1][16] = 0.044000
probbsij[2][16] = 0.043000
probbsij[3][16] = 0.044000
probbsij[4][16] = 0.049000
probbsij[5][16] = 0.052000
probbsij[6][16] = 0.065000
probbsij[7][16] = 0.039000
probbsij[8][16] = 0.053000
probbsij[9][16] = 0.057000
probbsij[10][16] = 0.052000
```

Vemos que este resultado tampoco es muy bueno ya que nos debería dar un número cercano a $0.0625 = 0.5^4$

I.5. Ejercicio 4c - SAC y BIC para las cajas-S del AES

El programa que hemos implementado para comprobar estos criterios es muy parecido al del DES.

En este caso hemos obtenido resultados mucho más satisfactorios.

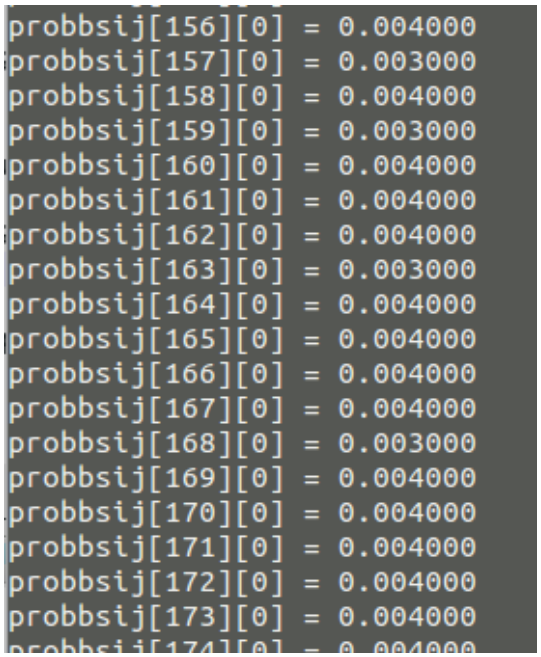
Los resultados de SAC son los siguientes:

```
prob0 [0][0]: 0.502000
prob1 [0][0]: 0.498000
prob0 [1][0]: 0.499000
prob1 [1][0]: 0.501000
prob0 [2][0]: 0.500000
prob1 [2][0]: 0.500000
prob0 [3][0]: 0.505000
prob1 [3][0]: 0.495000
prob0 [4][0]: 0.499000
prob1 [4][0]: 0.501000
prob0 [5][0]: 0.498000
prob1 [5][0]: 0.502000
prob0 [6][0]: 0.499000
prob1 [6][0]: 0.501000
prob0 [7][0]: 0.500000
prob1 [7][0]: 0.500000
prob0 [0][1]: 0.502000
prob1 [0][1]: 0.498000
prob0 [1][1]: 0.504000
prob1 [1][1]: 0.496000
prob0 [2][1]: 0.502000
prob1 [2][1]: 0.498000
```

Sabemos que la S-caja del Rijndael no satisface de manera directa el criterio de avalancha estricto,pero si dentro de una pequeña región de error.

Dicho error es 0.125 y nuestros resultados están dentro de este error.

Los resultados del BIC son los siguientes:



Vemos que $0.0039 = 0.5^8$ por lo que el AES cumple el principio de BIC.

Índice alfabético