

# 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 . . . . .	3

# 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]++;
    longText++;
}
```

Y finalmente volvemos a dividir entre la longitud del texto.

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.

Los resultados obtenidos tras probar el código son:

## 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.

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 e

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

## 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.

# Índice alfabético

Seguridad Perfecta, [2](#)