Cifrado de ficheros mediante DES-CBC

Autor: Eduard Guix Terricabras ETIS

Consultor: Antoni Martínez Balleste

Enero 2007

Índice

• Prefacio

Ha pasado mucho tiempo desde que se cifraron los primeros textos, mediante los sistemas de transposición o sustitución como el Cifrado del César. Eran unos tiempos en los que la ciencia de la Criptografía era conocida como el arte de la escritura secreta, hasta nuestros días donde la sociedad de la información la ha convertido en una necesidad indispensable. Del único objetivo que se perseguía inicialmente, el secreto del mensaje o la confidencialidad se ha pasado a ampliar sus objetivos con los de integridad y autenticidad para el tratamiento de la información electrónica, teniendo su culminación en los actuales sistemas de comercio electrónico.

En este proyecto y ante la actual explosión en los servicios de transmisión electrónica de información, se pretende hacer un análisis del sistema de cifrado DES (Data Encryption Standard), desde el punto de vista instructivo para que pueda ser utilizado en el futuro como elemento de enseñanza para facilitar la compresión de su funcionamiento.

Es de sobra conocido que la informática es una ciencia que se está aplicando a numeroso campos de la vida, desde la exploración del universo, transbordadores espaciales, hasta la cocina de casa, es por ello que la temática de un Trabajo de Fin de Carrera es muy amplia y diversa.

En el presente trabajo los campos tratados son principalmente dos, el software con la programación orientada a objetos y la Criptografía con algoritmos matemáticos. Es por ello que para su realización se han puesto en práctica los conocimientos de diversas asignaturas, en mayor o menor grado que durante los estudios de Ingeniería Técnica Informática se han ido adquiriendo.

- ## Especificación del problema

Se ha credo una utilidad para cifrar ficheros mediante un sistema basado en la contraseña entrada por el usuario. La clave para cifrar o descifrar un fichero depende únicamente de la contraseña que el usuario introduce en una interficie gráfica creada a tal efecto y que permite ejecutar las operaciones de cifrar o descifrar, como se ha mencionado anteriormente, localizar el fichero de entrada y controlar el correcto funcionamiento de la misma mediante mensajes informativos.

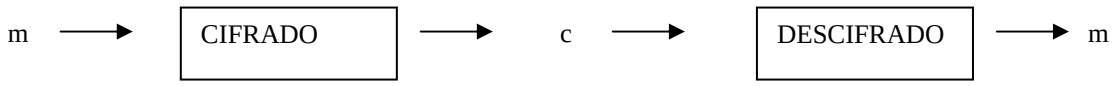
El objetivo de este trabajo no es crear una eficiente y muy potente utilidad de cifrado, puesto que de ser así, no creo que se lograse mejorar las librerías que a tal efecto se han desarrollado en el lenguaje Java, sino que se ha enfocado hacia un carácter educativo con la finalidad de mostrar los detalles de este extendido y singular sistema de cifrado.

Para la creación de esta utilidad de cifrado he escogido un sistema con clave simétrica y arquitectura de cifrado en bloque llamado DES (Data Encryption Standard). El modo de operación seleccionado de los posibles estándares es el CBC (Cipher Block Chaining) y el relleno del bloque se hace mediante PKCS5padding.

• Fundamentos

La Criptología (del griego criptos = oculto y logos = tratado) es el nombre para designar dos disciplinas: Criptografía y Criptoanálisis.
La Criptografía tiene como objetivo enmascarar determinada información de carácter secreto. Es un arte tan antiguo como la propia escritura y como tal ha permanecido durante mucho tiempo vinculada estrechamente a círculos militares y diplomáticos puesto que eran los que tenían una mayor necesidad. En cambio en la actualidad esta situación ha cambio radicalmente con el desarrollo de los sistemas de comunicación electrónicos y las ingentes cantidades de información almacenadas electrónicamente y que es necesario proteger.

El funcionamiento de un proceso criptográfico puede esquematizarse de la siguiente forma:



Dados un emisor A y un receptor B de un determinado mensaje, A transforma “m” (el mensaje original, texto claro) mediante un algoritmo y controlado por una determinada clave, en “c” un criptograma (texto cifrado). El receptor B, que tiene conocimiento de la clave, transforma el criptograma en texto claro, obteniendo así el mensaje original.

La finalidad de la Criptografía es múltiple: en primer lugar mantener la confidencialidad del mensaje, es decir la información que contiene es secreta, a continuación garantizar la autenticidad tanto del criptograma como de la pareja remitente/destinatario. La Criptografía clásica se ocupaba únicamente del primer aspecto y en la actualidad los sistemas deben garantizar ambos aspectos.

Según el tipo de algoritmo y las características de las claves empleadas podemos clasificar los sistemas Criptográficos en:

- Simétricos. Son aquellos en los que la clave de cifrado coincide con la de descifrado. Obviamente la clave debe ser secreta y solamente conocida por el emisor y el receptor y es por ello que para el intercambio de la clave se debe efectuar por un canal seguro.
- Asimétricos. Son aquellos en los que la clave de cifrado o privada es diferente de la clave de descifrado o pública. Este tipo de sistemas eliminan el problema de tener que intercambiar la clave.

Los métodos simétricos son propios de la Criptografía clásica o de clave secreta, mientras que los asimétricos corresponden a la Criptografía de clave pública que fueron introducidos por Diffie y Hellman en 1976.

En el ámbito de la Criptografía de clave secreta hay dos procedimientos que se han ido repitiendo a lo largo de la historia, que son:

- Sustitución. Establecen una correspondencia entre el alfabeto del texto claro y el del criptograma.
- Transposición. Realizan una reordenación de los símbolos del mensaje.

Como ejemplos históricos de estos cifrados podemos citar el Cifrado del César (siglo I a. C.) y la Escítala Lacedemonia (siglo V a. C.)

Siguiendo en la Criptografía clásica encontramos dos métodos de cifrado en función de cómo se trata el texto claro:

- De flujo. Dividen el mensaje a cifrar en caracteres, cifrando cada carácter mediante una función variante en el tiempo. Así, después del cifrado de cada carácter el sistema evoluciona a un nuevo estado de acuerdo con una determinada regla. Como consecuencia de ello caracteres idénticos tienen cifrados diferentes.
- De bloque. Dividen el mensaje en fragmentos usualmente de la misma longitud y les aplican una transformación independiente. Una característica de estos es que a mensajes idénticos se generan criptogramas idénticos.

Hay que tener en cuenta que todos los algoritmos de cifrado en bloque se componen de los siguientes pasos:

1. Expansión de la clave
2. Transformación inicial
3. Función criptográfica iterada “n” veces
4. Transformación final

• El estándar DES

Data Encryption Standard (DES), es un algoritmo de cifrado en bloque simétrico de longitud fija, el cual consiste en 2 permutaciones, 16 vueltas donde el mensaje de 64 bits es dividido en dos bloques de 32 bits. Después de la primera permutación, es cifrado 16 veces utilizando cada vez una subclave, la cual se genera en un proceso anterior y finalmente se aplica una permutación final.

• Historia

En 1973, el NBS (National Bureau of Standards, USA), una sección del Departamento de Defensa, organizó un concurso solicitando un sistema de encriptación. En 1974, la corporación IBM presentó, entre otras, una propuesta basada en un sistema que ya habían desarrollado llamado Lucifer, que mediante algunos cambios dio lugar al DES. La aprobación y la modificación de la propuesta se hicieron bajo la supervisión de la NSA (National Security Agency).

Fue la NSA la que impuso la longitud de la clave, que es bastante corta y que no es aconsejable en la actualidad. Hubo algunas críticas desde ciertos sectores, incluyendo los pioneros de la criptografía simétrica Diffie y Hellman, mencionando la modesta longitud de la clave y las misteriosas cajas "S" como una evidencia de la inadecuada interferencia de la NSA. La sospecha era que de una forma intencionada habían debilitado el algoritmo para que ellos pudiesen leer los mensajes cifrados.

A pesar de la polémica, el DES fue aprobado como estándar y en 1977 como FIPS PUB 46 autorizado para su uso. En 1981 el ANSI (American National Standards Institute, USA) lo adopto bajo el nombre de DEA (Data Encryption Algorithm). Fue posteriormente revisado como FIPS-46-1 en 1983 y 1988 y como FIPS-46-2 en 1993, finalmente en 1998 como FIPS-46-3 se definió una variante del mismo llamado "Triple DES" que ha perdurado hasta que en 2002 fue reemplazado por AES (Avanced Encryption Standard).

• Descripción del funcionamiento

Cifrado

El algoritmo empieza con una permutación inicial (Tabla de permutaciones Entrada) del bloque de entrada, seguido de 16 iteraciones idénticas y finaliza con una permutación de salida (Tabla de permutaciones Salida).

Tabla de permutaciones Entrada							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tabla de permutaciones Salida							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

La forma de interpretar las tablas de permutaciones es la siguiente. Se irán tomando los números de arriba hacia abajo y de izquierda a derecha que representan el orden de la tabla de salida y el numero que encontramos es la posición del BIT de la tabla de entrada.
En la primera tabla tenemos que el primer BIT de salida es el BIT que ocupa la posición número 58 de la tabla de entrada.

El bloque resultante de la permutación inicial es dividido en dos mitades iguales (E = izquierda y D = derecha), a la mitad derecha le es aplicada la función Feistel y a su resultado un XOR con la mitad izquierda que pasa a ser la mitad derecha de la siguiente iteración, la mitad derecha será la mitad izquierda de la siguiente iteración.

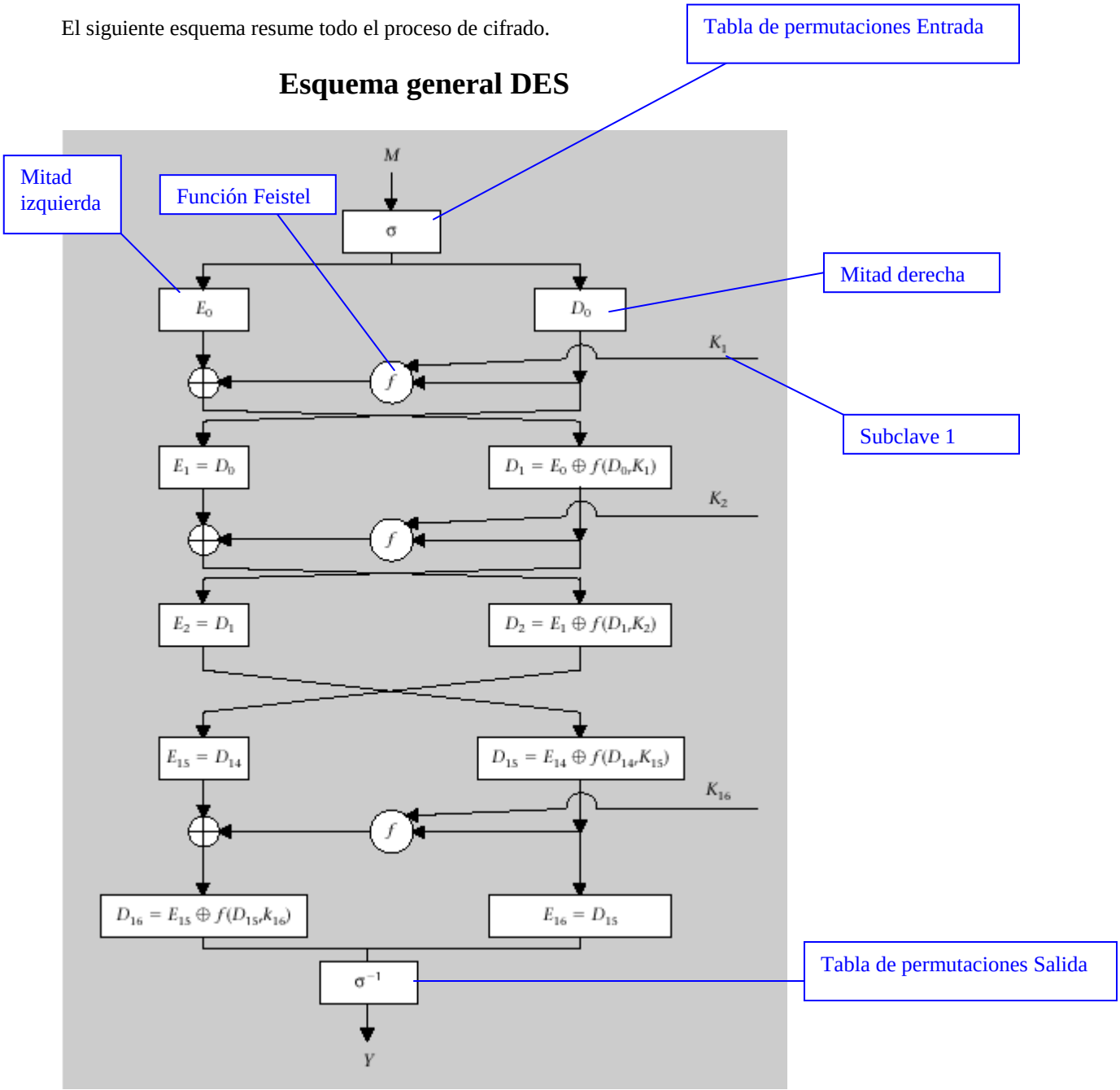
Para cada iteración se verifica que:

$$E_i = D_{i-1}, \quad D_i = E_{i-1} \oplus f(D_{i-1}, K_i),$$

donde K_i es una de la subclaves de 48 bits

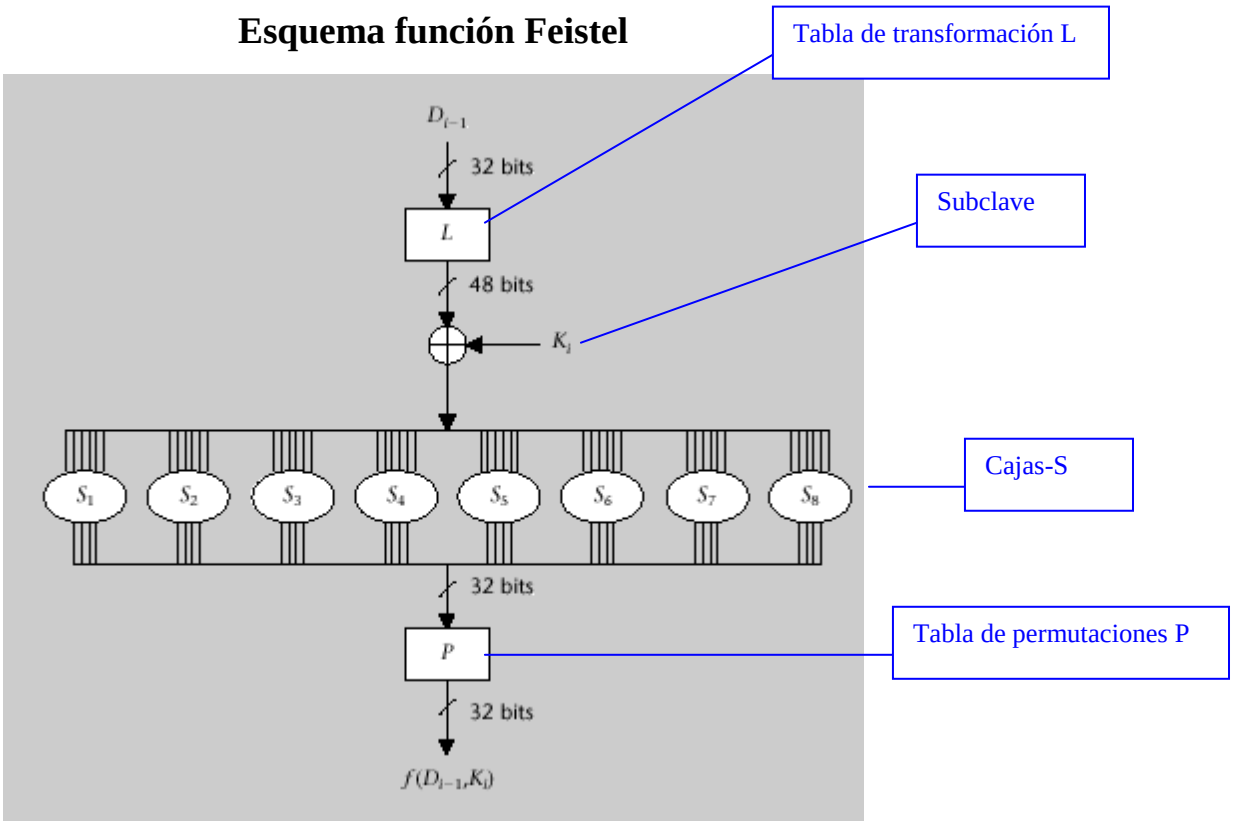
El siguiente esquema resume todo el proceso de cifrado.

Esquema general DES



Hay que tener en cuenta que en la última iteración no se intercambian las dos mitades derecha e izquierda, sino que se aplica directamente la permutación final. Es por ello que el mismo algoritmo de cifrado luego no ha de servir para descifrar los criptogramas creados.

Este es el esquema de la función Feistel para interpretar el resultado que nos devuelve.



En primer lugar se ha de transformar D_{i-1} para que tenga 48 bits, ello se consigue mediante la siguiente tabla de transformación L.

Tabla de transformación L							
32	1	2	3	4	5		
4	5	6	7	8	9		
8	9	10	11	12	13		
12	13	14	15	16	17		
16	17	18	19	20	21		
20	21	22	23	24	25		
24	25	26	27	28	29		
28	29	30	31	32	1		

A continuación aplicamos un XOR de la mitad expandida $L(D_{i-1})$ con la clave K_i . El resultado de esta operación se descompone en 8 bloques de 6 bits y cada bloque se utiliza como entrada de una de la S-Cajas que retornan 4 bits según la siguiente tabla.

							Columna										
Caja S	Fila	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S1	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S2	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	12	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S4	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S5	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S8	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

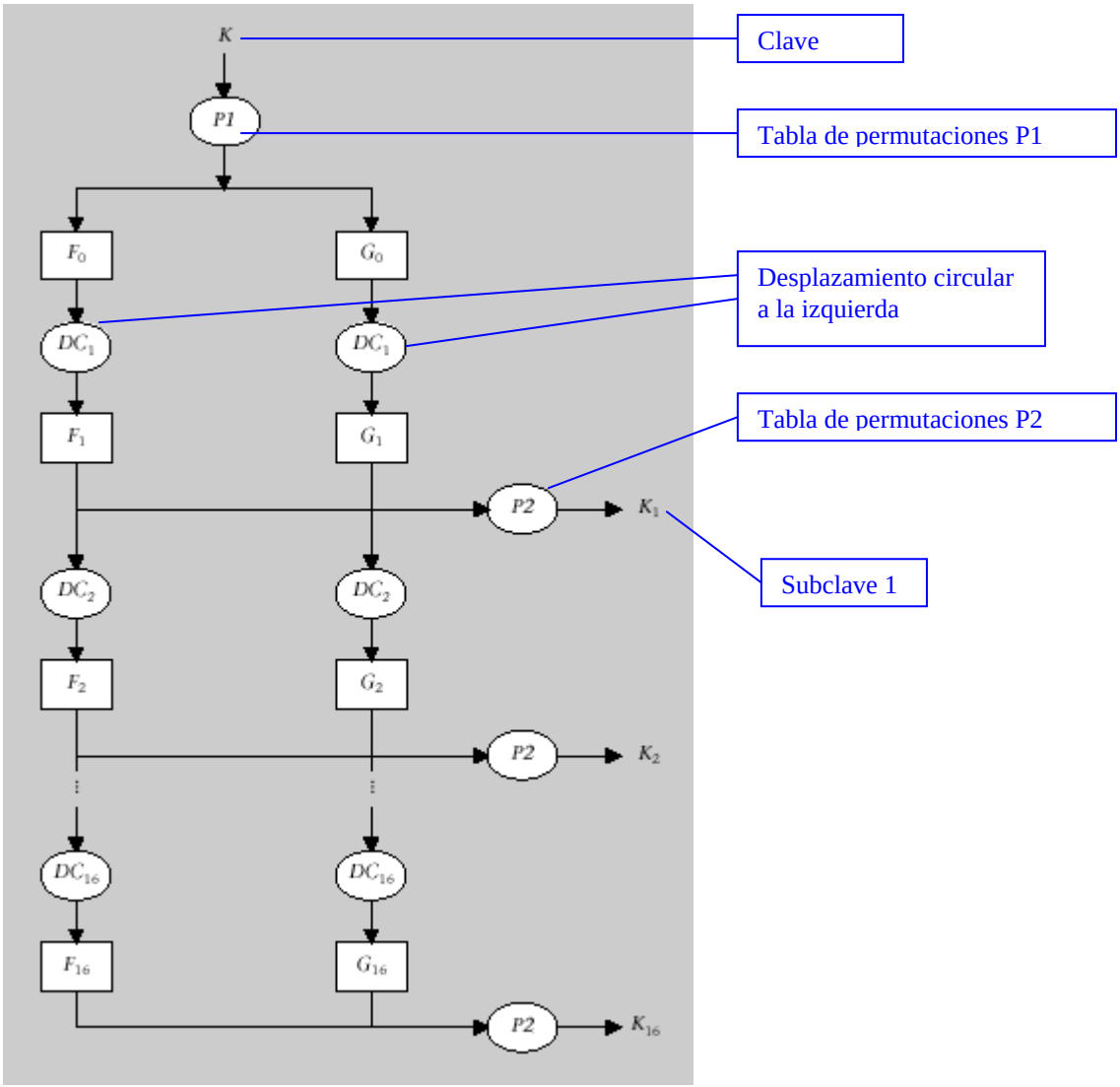
Los bloques de 4 bits resultantes se concatenan para obtener un bloque de 32 bits de longitud al cual se le aplica la permutación P con el fin de obtener el valor de $f(D_{i-1}, K_i)$

Tabla de permutaciones P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Creación subclaves

Este es el esquema general de funcionamiento del proceso de creación de las subclaves.
En él podemos observar a K la clave de entrada para la generación, P1 y P2 las permutaciones inicial y final, DC los desplazamientos circulares las Kn que son las subclaves resultantes.

Esquema generación subclaves DES



Para la generación de las subclaves, el criptosistema utiliza como entrada la clave de 64 bits de los cuales solo 56 bits son útiles para la generación de las 16 subclaves de 48 bits.
Son 16 las subclaves que se deben generar a partir de la clave de entrada ya que el criptosistema realiza 16 iteraciones para crear el cifrado y por cada una de ellas hace uso de una subclave.

Los pasos a realizar son los siguientes:

- Permutación inicial P1. De los 64 bits de la contraseña se descartan los 8 bits de paridad, por lo tanto nos queda una tabla de salida de 56 bits.

Tabla de permutaciones P1							
57	49	41	33	25	17	9	
1	58	50	42	34	26	18	
10	2	59	51	43	35	27	
19	11	3	60	52	44	36	
63	55	47	39	31	23	15	
7	62	54	46	38	30	22	
14	6	61	53	45	37	29	
21	13	5	28	20	12	4	

- Creación 2 mitades. A partir de la salida de la tabla anterior se crean dos mitades iguales, la izquierda y la derecha de 28 bits.
- Desplazamiento circular izquierda. Este desplazamiento puede ser simple o doble en función de la subclave.

Tabla de desplazamientos a la izquierda	
Iteración	Número
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

- Unir 2 mitades. De las mitades izquierda y derecha de 28 bits respectivamente, se crea una sola.
- Permutación P2. Tomando como tabla de entrada la unión de las 2 mitades se ejecuta esta permutación que dará lugar a la subclave n de 48 bits.

Tabla de permutaciones P2							
14	17	11	24	1	5		
3	28	15	6	21	10		
23	19	12	4	26	8		
16	7	27	20	13	2		
41	52	31	37	47	55		
30	40	51	45	33	48		
44	49	39	56	34	53		
46	42	50	36	29	32		

Descifrado

El proceso de descifrado es el mismo que hemos utilizado para cifrar el texto, con una única salvedad que en las 16 iteraciones que hemos de efectuar en el proceso de descifrado el orden de las subclaves es el inverso del cifrado. Es decir, en la primera iteración emplearemos la subclave 16, en la segunda la 15 hasta llegar a la iteración 16 en la que emplearemos la primera subclave.

• Vulnerabilidades

Es obvio que el método más simple de ataque es por fuerza bruta probando cada una de las posibles claves del sistema $2^{56} = 7{,}2 * 10^{16}$ claves diferentes.
En 1977 Diffie y Hellman propusieron una máquina que era capaz de hallar una clave DES en un solo día, pero su proyecto no se llevo a cabo a causa del elevado coste de fabricación. Sin embargo, la vulnerabilidad de DES quedó patente cuando en 1998 la Electronical Frontier Fondation (EFF) construyó una máquina a medida para romper DES que lo consiguió a los 2 días.

Hoy en día, ha quedado demostrada la fragilidad de los sistemas que trabajan con claves de 56 bits como es el caso del sistema DES ya que se ha conseguido romperlo en 22 horas y 15 minutos.
Para solucionar el problema de la longitud de la clave y continuar utilizando el sistema DES de una forma segura se utiliza el Triple DES que no es más que un triple cifrado, descifrado utilizando DES.

Existen tres ataques conocidos más rápidos que la fuerza bruta bajo ciertas condiciones que son:

- El Criptoanálisis diferencial
- El Criptoanálisis lineal
- El ataque de Davies

• Ventajas e inconvenientes

Las principales ventajas son básicamente dos, su relativa simplicidad y la facilidad de implementación, ya sea basado en hardware o software.

El sistema DES consigue cumplir el principio de confusión y difusión gracias a las acciones de sus cajas, cuyas operaciones son muy fáciles de implementar en hardware y le aportan una velocidad de cifrado y descifrado muy superior a otros algoritmos de cifrado en bloque.

Las debilidades más graves que se le atribuyen, son la poca longitud de su clave y la arbitrariedad de sus cajas, que es posible que obedezcan a la existencia de una clave maestra que permitiera descifrar los mensajes sin poseer la clave secreta.

• Diseño del programa

Una vez finalizado el análisis de sistema DES aparecieron tres tipos de clases:

1. Interficie gráfica.
   Las clases que forman este grupo son Frame1.java y FrameMsg.java.
   La primera de ellas es la que forma la interficie gráfica, donde el usuario interactúa con la aplicación.
   FrameMsg.java es una ventana con la finalidad de presentar los mensajes que el aplicativo genera.

2. Operaciones de entrada salida y cifrado de bloque.
   Lo compone la clase Run.java.
   En esta clase se encuentran los procedimientos necesarios para la lectura y escritura de ficheros binarios, tratamiento de cifrado de bloque en modo CBC y relleno utilizando PKCS5padding así como toda una serie de utilidades de conversión necesarias para los métodos implementados de cifrado.

```
  /**
   * Mètode auxiliar per omplir l'últim bloc amb el format PKCS5padding
   * @param array matriu a omplir
   * @param len llargada de la matriu
   */
  private void padding(byte[] array, int len) {
```

```
  /**
   * Metode per convertir els caracters d'un string en el seus valors hexadecimals
   * i així poder representar el text xifrat en hexadecimal
   * @param s cadena amb els caracters
   * @return cadena amb els valor hexadecimals per parelles
   */
  private String stringToHex(String s) {
```

Para las dos posibles acciones, cifrado y descifrado se han definido métodos distintos aunque gran parte del código de ambos es común. Ello se ha hecho de esta manera, para que tan solo echando un vistazo a los métodos de la clase se pueda intuir su funcionamiento y así facilitar la posible reutilización de la misma.

```
  /**
   * Mètode per invocar al sistema DES amb l'acció de xifrar
   * @param clau clau secreta de xifratge
   * @param nomFitxer fitxer de entrada que es vol xifrar
   */
  public void xifrar(String clau, String nomFitxer) {
```

3. Algoritmo de cifrado DES.
   Esta compuesto por las clases ClausDES.java y XifratDES.java.

```
/**
 * <p>Title: ClausDES</p>
 * <p>Description: Generació de les 16 subclaus necessaries pel criptosistema DES
 * Treball de Fi de Carrera ETIS</p>
 * <p>Copyright: Copyright (c) 2007</p>
 * @author Eduard Guix Terricabras
 * @version 1.0
 */

public class ClausDES {

  private char[] key; //clau generadora

  static final int NKEYS=16;     //nombre de claus generades
  static final int SIZE_KEY=8*8; //mida en bits de la taula inicial, files*columnes
  static final int SIZE_REAL_KEY=8*7; //mida en bits que realment s'utilitzen de la clau
  static final int SIZE_SUBKEYS=8*6; //mida de les subclaus generades

  static final int SIZE_P1=8*7;  //mida de la taula de la primera permutació, files*columnes
  static final int PERM_P1[] = { //posicions dels bits de la primera permutació
```

```
                  57, 49, 41, 33, 25, 17,  9,
                   1, 58, 50, 42, 34, 26, 18,
                  10,  2, 59, 51, 43, 35, 27,
                  19, 11,  3, 60, 52, 44, 36,
                  63, 55, 47, 39, 31, 23, 15,
                   7, 62, 54, 46, 38, 30, 22,
                  14,  6, 61, 53, 45, 37, 29,
                  21, 13,  5, 28, 20, 12,  4};
  static final int SIZE_P2=8*6; //mida de la taula de la segona permutació
  static final int PERM_P2[] = {//posicions dels bits de la segona permutació
                  14, 17, 11, 24,  1,  5,
                   3, 28, 15,  6, 21, 10,
                  23, 19, 12,  4, 26,  8,
                  16,  7, 27, 20, 13,  2,
                  41, 52, 31, 37, 47, 55,
                  30, 40, 51, 45, 33, 48,
                  44, 49, 39, 56, 34, 53,
                  46, 42, 50, 36, 29, 32};
  //llista amb el nombre de desplaçaments circulars a la esquerra per cada clau
  static final int DC[] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

  /**
   * subclaus creades
   */
  private char[][] keys;

  /**
   * Metode per llegir les subclaus
   * @return Vector amb les subclaus generades
   */
  public char[][] getKeys() { return this.keys; }

  /**
   * Mètode per efectuar permutacions en base un taula amb posicions
   * @param tbP taula amb les posicions de la permutació
   * @param taulaEnt taula d'entrada a permutar
   * @return taula resultant
   */
  private char[] permutar (int tbP[], char taulaEnt[]) {
      char[] taulaPerm = new char[tbP.length];

      for (int i=0; i<tbP.length; i++) {
        taulaPerm[i]=taulaEnt[(tbP[i] - 1)];
      }

      return taulaPerm;
  }

  /**
   * Metode per efectuar un desplaçament circular cap a l'esquerra
   * @param cadena nombre binari a desplaçar
   * @return resultat del desplaçament
   */
  private char[] desplaCirEsq(char[] cadena) {
      int i;
      char[] r = new char[SIZE_REAL_KEY/2];

      for (i=1; i<(SIZE_REAL_KEY/2); i++) {
        r[i-1] = cadena[i];
      }
      r[i-1] = cadena[0];

      return r;
  }

  /**
   * Mètode per dividir una matriu
   * @param cadena matriu a dividir
   * @param ED E=Esquerra, D=Dreta
   * @return la meitat
   */
  private char[] creaMeitats (char[] cadena, char ED) {
    int des = 0;
    char[] r = new char[SIZE_REAL_KEY/2];

    if (ED == 'D') des = SIZE_REAL_KEY/2;
    for (int i=0; i<(SIZE_REAL_KEY/2); i++) {
```

```
      r[i] = cadena[des + i];
    }

    return r;
  }

  /**
   * Métode invers al "creaMeitats" que el que fa es juntar-les
   * @param esque el tros esquerra
   * @param dreta el tros dreta
   * @return matriu resultat esquerra+dreta
   */
  private char[] juntaMeitats(char[] esque, char[] dreta) {
    char[] r = new char[SIZE_REAL_KEY];

    for (int i=0; i<(SIZE_REAL_KEY/2); i++) {
      r[i] = esque[i];
    }
    for (int j=0; j<(SIZE_REAL_KEY/2); j++) {
      r[(SIZE_REAL_KEY/2) + j] = dreta[j];
    }

    return r;
  }

  /**
   * Mètode que crea les subclaus pel proces de xifrat DES
   * @param key clau inicial de 8 caracters representats en binari
   */
  public ClausDES(char[] key) {
    this.key  = key;
    this.keys = new char[NKEYS][SIZE_SUBKEYS];

    char[] p1 = permutar(PERM_P1, key); //primera permutació

    char[] f1, g1;
    char[] f0 = creaMeitats(p1, 'E');
    char[] g0 = creaMeitats(p1, 'D');

    for (int i=0; i<NKEYS; i++) {
      f1 = desplaCirEsq(f0); // desplaçament circular esquerra
      g1 = desplaCirEsq(g0);
      if (DC[i] == 2) { //hi ha doble desplaçament
        f1 = desplaCirEsq(f1);
        g1 = desplaCirEsq(g1);
      }
      char[] k = permutar(PERM_P2, juntaMeitats(f1, g1)); //segona permutació
      keys[i] = k;
      f0 = f1; g0 = g1;
    }

  }//final metode de creació


}//final classe
```

La primera clase como por su nombre se puede deducir es la encargada de generar las 16 subclaves necesarias para cada una de las iteraciones del proceso de cifrado que se lleva a cabo en la segunda clase XifratDES.java.

```
/**
 * <p>Title: XifratDES</p>
 * <p>Description: Algoritme de xifrat en bloc simètric DES (Data Encryption Standard)
 * Treball de Fi de Carrera ETIS</p>
 * <p>Copyright: Copyright (c) 2007</p>
 * @author Eduard Guix Terricabras
 * @version 1.0
 */

public class XifratDES {

  private char accio; //acció a realitzar X=xifrar, D=Desxifrar
  private char[][] keys; //subclaus de les iteracions

  static final int NKEYS=16; //nombre d'iteracions
```

```
  static final int SIZE_KEY=8*8; //mida en bits de la clau
  static final int SIZE_REAL_KEY=8*7; //mida en bits que realment s'utilitzen de la clau
  static final int NCAIXES=8;
  static final int BITS_E_CAIXA=6;
  static final int BITS_S_CAIXA=4;

  static final int SIZE_PE=8*8;  //mida de la taula de la permutació d'entrada, files*columnes
  static final int PERM_PE[] = { //posicions dels bits de la permutació d'entrada
                  58, 50, 42, 34, 26, 18, 10, 2,
                  60, 52, 44, 36, 28, 20, 12, 4,
                  62, 54, 46, 38, 30, 22, 14, 6,
                  64, 56, 48, 40, 32, 24, 16, 8,
                  57, 49, 41, 33, 25, 17,  9, 1,
                  59, 51, 43, 35, 27, 19, 11, 3,
                  61, 53, 45, 37, 29, 21, 13, 5,
                  63, 55, 47, 39, 31, 23, 15, 7};
  static final int SIZE_PS=8*8;  //mida de la taula de la permutació de sortida, files*columnes
  static final int PERM_PS[] = { //posicions dels bits de la permutació de sortida
                  40,  8, 48, 16, 56, 24, 64, 32,
                  39,  7, 47, 15, 55, 23, 63, 31,
                  38,  6, 46, 14, 54, 22, 62, 30,
                  37,  5, 45, 13, 53, 21, 61, 29,
                  36,  4, 44, 12, 52, 20, 60, 28,
                  35,  3, 43, 11, 51, 19, 59, 27,
                  34,  2, 42, 10, 50, 18, 58, 26,
                  33,  1, 41,  9, 49, 17, 57, 25};
  static final int SIZE_PL=8*6;  //mida de la taula de la permutació
  static final int PERM_PL[] = { //posicions dels bits de la permutació
                  32,  1,  2,  3,  4,  5,
                   4,  5,  6,  7,  8,  9,
                   8,  9, 10, 11, 12, 13,
                  12, 13, 14, 15, 16, 17,
                  16, 17, 18, 19, 20, 21,
                  20, 21, 22, 23, 24, 25,
                  24, 25, 26, 27, 28, 29,
                  28, 29, 30, 31, 32,  1};
  static final int SIZE_PP=8*4;  //mida de la taula de la permutació
  static final int PERM_PP[] = { //posicions dels bits de la permutació
                  16,  7, 20, 21,
                  29, 12, 28, 17,
                   1, 15, 23, 26,
                   5, 18, 31, 10,
                   2,  8, 24, 14,
                  32, 27,  3,  9,
                  19, 13, 30,  6,
                  22, 11,  4, 25};

  static final int CAIXES[] = {
                   14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7, //S1
                    0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
                    4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
                   15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13,

                   15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10, //S2
                    3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5,
                    0, 14,  7, 11, 10,  4, 12,  1,  5,  8, 12,  6,  9,  3,  2, 15,
                   13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9,

                   10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8, //S3
                   13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
                   13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
                    1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12,

                    7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15, //S4
                   13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
                   10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
                    3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14,

                    2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9, //S5
                   14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
                    4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
                   11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3,

                   12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11, //S6
                   10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
                    9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
                    4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13,
```

```
                    4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1, //S7
                   13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
                    1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
                    6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12,

                   13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7, //S8
                    1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
                    7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
                    2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11};

  /**
   * Constructor de la classe que crida a la classe que crea les subclaus
   * @param accio operació a fer X=xifrar, D=desxifrar
   * @param key cadena de 8 caracters que són la clau secreta
   */
  public XifratDES(char accio, char[] key) {
    this.accio = accio;

    ClausDES cDES = new ClausDES(key);
    this.keys = cDES.getKeys();
  }

  /**
   * Mètode per convertir un valor binari (4 bits) en decimal
   * @param a quart bit de major pes
   * @param b tercer
   * @param c segon
   * @param d primer bit de menor pes
   * @return valor decimal
   */
  private int binToDec(char a, char b, char c, char d) {
       if      (a == '0' && b == '0' && c == '0' && d == '0') return 0;
       else if (a == '0' && b == '0' && c == '0' && d == '1') return 1;
       else if (a == '0' && b == '0' && c == '1' && d == '0') return 2;
       else if (a == '0' && b == '0' && c == '1' && d == '1') return 3;
       else if (a == '0' && b == '1' && c == '0' && d == '0') return 4;
       else if (a == '0' && b == '1' && c == '0' && d == '1') return 5;
       else if (a == '0' && b == '1' && c == '1' && d == '0') return 6;
       else if (a == '0' && b == '1' && c == '1' && d == '1') return 7;
       else if (a == '1' && b == '0' && c == '0' && d == '0') return 8;
       else if (a == '1' && b == '0' && c == '0' && d == '1') return 9;
       else if (a == '1' && b == '0' && c == '1' && d == '0') return 10;
       else if (a == '1' && b == '0' && c == '1' && d == '1') return 11;
       else if (a == '1' && b == '1' && c == '0' && d == '0') return 12;
       else if (a == '1' && b == '1' && c == '0' && d == '1') return 13;
       else if (a == '1' && b == '1' && c == '1' && d == '0') return 14;
       else                                                   return 15;
  }

  /**
   * Metode per convertir un valor decimal en binari de 4 bits
   * @param numDec valor decimal [0..15]
   * @return valor binari [0000..1111]
   */
  private String decToBin(int numDec) {
    switch(numDec) {
       case  0: return "0000";
       case  1: return "0001";
       case  2: return "0010";
       case  3: return "0011";
       case  4: return "0100";
       case  5: return "0101";
       case  6: return "0110";
       case  7: return "0111";
       case  8: return "1000";
       case  9: return "1001";
       case 10: return "1010";
       case 11: return "1011";
       case 12: return "1100";
       case 13: return "1101";
       case 14: return "1110";
       case 15: return "1111";
      }
      return "";
  }
```

```
  /**
   * Mètode per dividir una matriu
   * @param cadena matriu a dividir
   * @param ED E=Esquerra, D=Dreta
   * @return la meitat
   */
  private char[] creaMeitats (char[] cadena, char ED) {
    int des = 0;
    char[] r = new char[SIZE_PE/2];

    if (ED == 'D') des = SIZE_PE/2;
    for (int i=0; i<(SIZE_PE/2); i++) {
      r[i] = cadena[des + i];
    }

    return r;
  }

  /**
   * Métode invers al "creaMeitats" que el que fa es juntar-les
   * @param esque el tros esquerra
   * @param dreta el tros dreta
   * @return matriu resultat esquerra+dreta
   */
  private char[] juntaMeitats(char[] esque, char[] dreta) {
    char[] r = new char[SIZE_PS];

    for (int i=0; i<(SIZE_PS/2); i++) {
      r[i] = esque[i];
    }
    for (int j=0; j<(SIZE_PS/2); j++) {
      r[(SIZE_PS/2) + j] = dreta[j];
    }

    return r;
  }

  /**
   * Mètode que realitza l'operació XOR binaria entre dos strings de nombres binaris
   * @param numBin1 primer nombre binari
   * @param numBin2 segon nombre binari
   * @return resultat de l'operació
   */
  private char[] funcioXOR (char[] numBin1, char[] numBin2) {
      char[] res = new char[numBin1.length];

      for (int i=0; i<numBin1.length; i++) {
        if (numBin1[i] == numBin2[i]) res[i] = '0'; else res[i] = '1';
      }

      return res;
  }

  /**
   * Mètode per efectuar permutacions
   * @param tbP taula amb les posicions de la permutació
   * @param taulaEnt taula de entrada a permutar
   * @return taula resultant
   */
  private char[] permutar (int tbP[], char taulaEnt[]) {
      char[] taulaPerm = new char[tbP.length];

      for (int i=0; i<tbP.length; i++) {
        taulaPerm[i]=taulaEnt[(tbP[i] - 1)];
      }

      return taulaPerm;
  }

  /**
   * Mètode que reprodueix la funció de Feistel
   * @param ordre numero d'iteració [1..16], es solament informatiu
   * @param blocD bloc d'entrada de la funció
   * @param subclau subclau de la funció
   * @return resultat
   */
  private char[] funcioF (int ordre, char[] blocD, char[] subclau) {
```

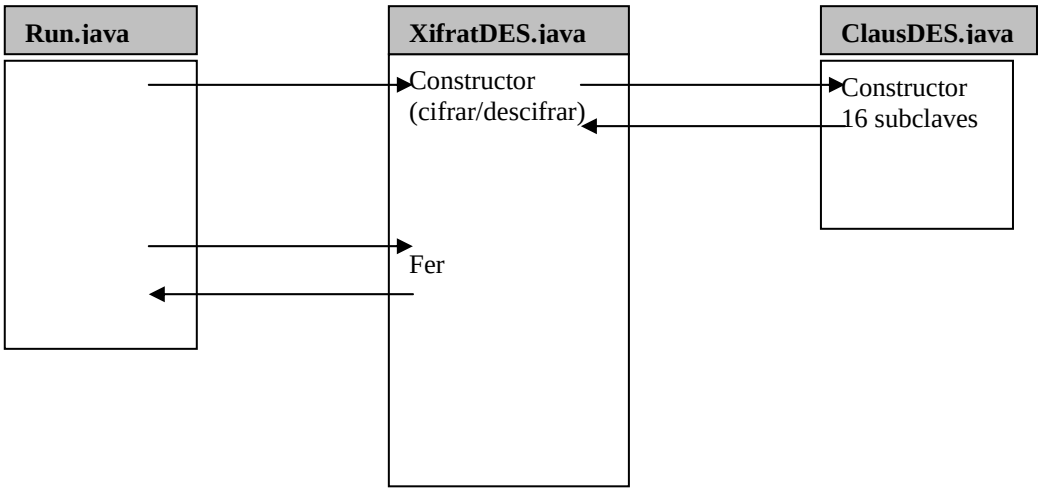
```
    char[] l = permutar(PERM_PL, blocD);

    char[] entCaixa = funcioXOR(l, subclau);

    char[] salCaixa = funcioCaixa(entCaixa);

    char[] r = permutar(PERM_PP, salCaixa);

    return r;
  }

  /**
   * Métode que reprodueix les operacions de les S-Caixes
   * @param blocEnt bloc d'entrada
   * @return resultat
   */
  private char[] funcioCaixa (char[] blocEnt) {
    int rcaixa, fila, columna;
    char[] r = new char[BITS_S_CAIXA * NCAIXES];
    char[] rc = new char[BITS_S_CAIXA];
    char bloc[] = new char [BITS_E_CAIXA];

    for (int i=0; i<NCAIXES; i++) {
      for (int j=0; j<BITS_E_CAIXA; j++) bloc[j] = blocEnt[(i*BITS_E_CAIXA)+j];

      fila = binToDec('0', '0', bloc[0], bloc[5]);
      columna = binToDec(bloc[1], bloc[2], bloc[3], bloc[4]);
      rcaixa = CAIXES[(i*64) + (fila*16) + columna];
      rc = decToBin(rcaixa).toCharArray();

      for (int k=0; k<BITS_S_CAIXA; k++) r[(i*BITS_S_CAIXA)+k] = rc[k];
    }

    return r;
  }

  /**
   * Mètode que realitza el xifratge/desxifratge del bloc de text
   * @param textChBin bloc d'entrada en binari
   * @return resultat de l'operació
   */
  public char[] fer(char[] textChBin) {
    int loopini, loopend, loopinc;

    if (this.accio == 'X') {
      loopini = 0;
      loopend = NKEYS;
      loopinc = 1;
    }
    else {
      loopini = NKEYS -1;
      loopend = -1;
      loopinc = -1;
    }

    char[] pe = permutar(PERM_PE, textChBin); //permutació d'entrada

    char[] f;
    char[] e1 = new char[SIZE_PS/2];
    char[] d1 = new char[SIZE_PS/2];
    char[] e0 = creaMeitats(pe, 'E'); //dues meitats inicials
    char[] d0 = creaMeitats(pe, 'D');

    for (int i=loopini; i!=loopend; i+=loopinc) {
      e1 = d0;
      f = funcioF(i+1, d0, this.keys[i]);
      d1 = funcioXOR(e0, f);

      e0 = e1; d0 = d1;
    }

    char[] ps = permutar(PERM_PS, juntaMeitats(d1, e1)); //permutació de sortida

    return ps;
  }

}//final clase
```

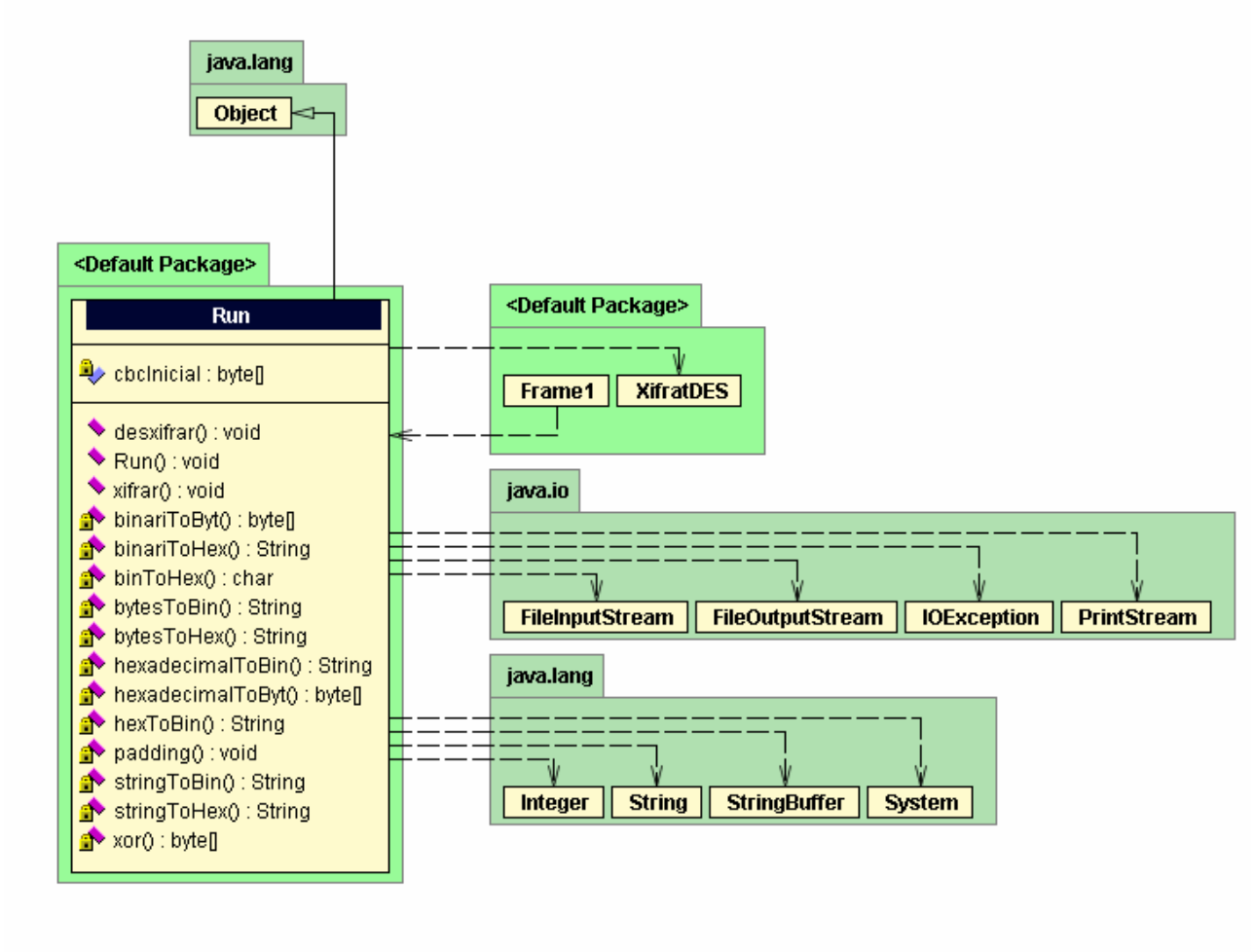
El funcionamiento de esta última clase es bien sencillo, en primer lugar hay que llamar al método constructor de la clase pasando como parámetros la acción a ejecutar, cifrar o descifrar y la correspondiente cadena que forma la clave secreta, acto seguido ya podemos ir invocando al método “fer” con una matriz de 64 bits que forma cada uno de los bloques del fichero de entrada para irlos cifrando. Es en el método constructor de la clase donde se utiliza la clase ClausDES.java para obtener las subclaves.
Para no tener un código demasiado extenso en la clase XifratDES.java el cifrado y el descifrado se realizan en el mismo método “fer” gracias a la inicialización de forma diferente de las variables que controlan el proceso.

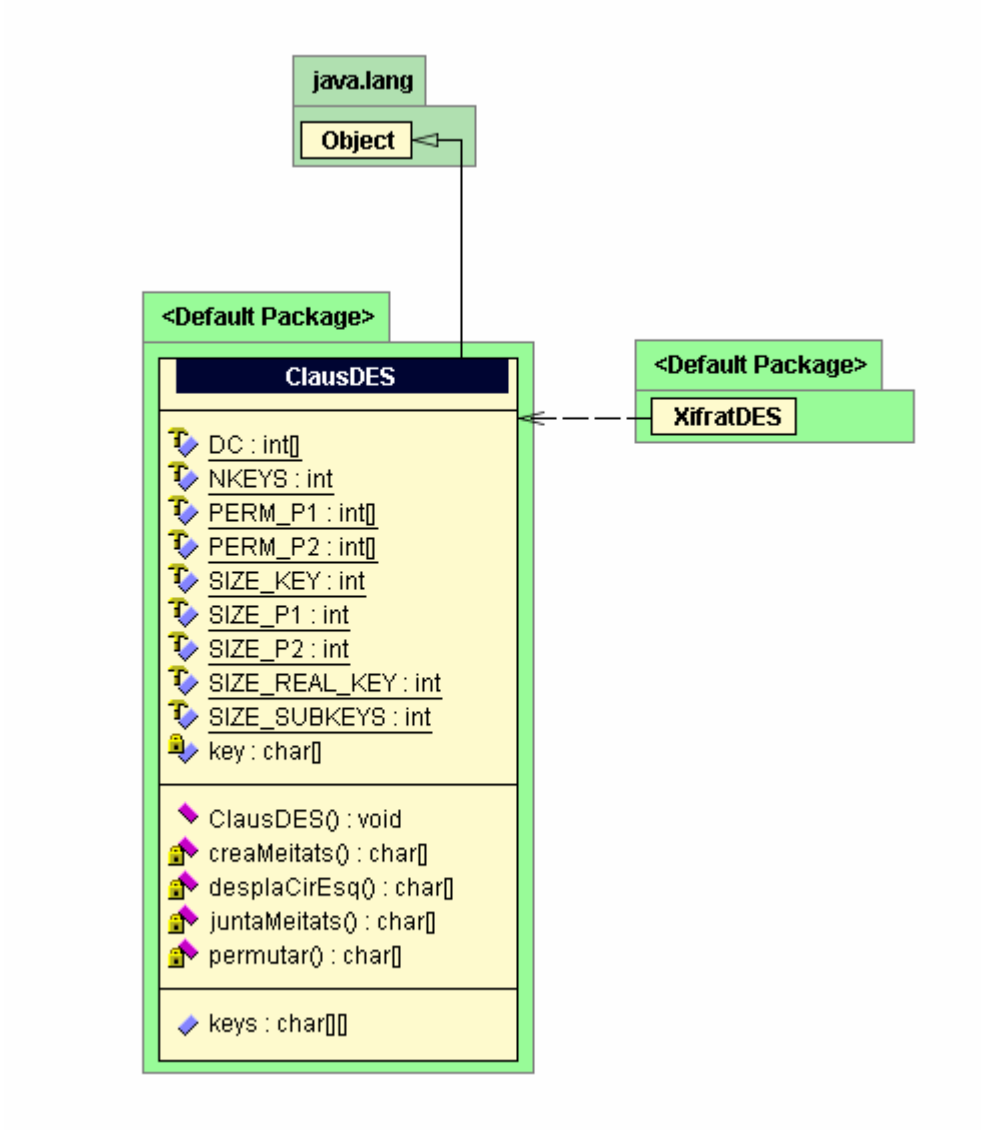
A parte de la clases mencionadas anteriormente existe la la clase Application1.java que es la que lanza la ejecución del aplicativo.

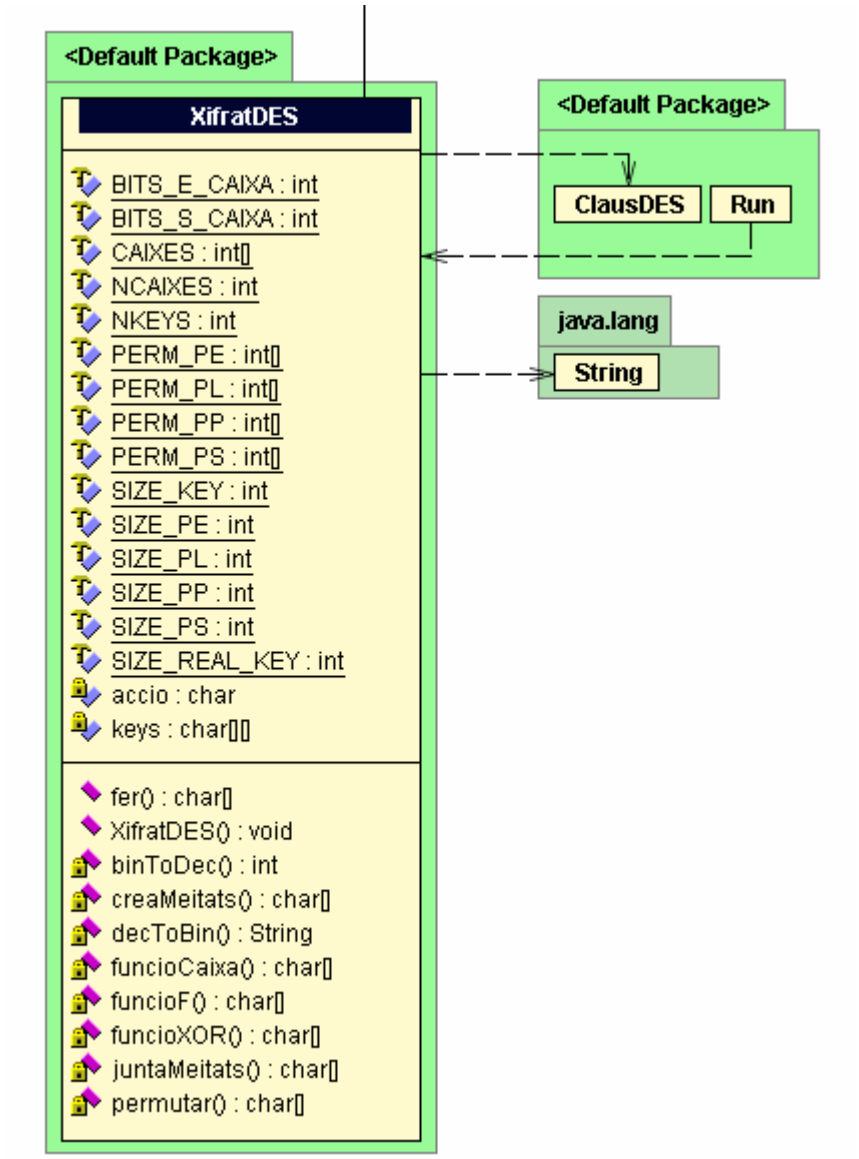
Se ha de tener en cuenta, que todo el código fuente que se ha escrito, se encuentra repleto de comentarios para facilitar su comprensión. Al margen de ello, además se ha generado una documentación de todas las clases que intervienen en el proyecto que tiene su ubcicación en el subdirectorio /doc.

A continuación para una mejor comprensión del funcionamiento global se muestra un pequeño esquema del funcionamiento del proceso de cifrado / descifrado, con las tres clases que intervienen asi como también sus diagramas UML.




java.lang
Object
<Default Package>
Run
cbcInicial : byte[]
desxifrar() : void
Run() : void
xifrar() : void
binariToByt() : byte[]
binariToHex() : String
binToHex() : char
bytesToBin() : String
bytesToHex() : String
hexadecimalToBin() : String
hexadecimalToByt() : byte[]
hexToBin() : String
padding() : void
stringToBin() : String
stringToHex() : String
xor() : byte[]
<Default Package>
Frame1
XifratDES
java.io
FileInputStream
FileOutputStream
IOException
PrintStream
java.lang
Integer
String
StringBuffer
System


java.lang
Object
<Default Package>
ClausDES
DC : int[]
NKEYS : int
PERM_P1 : int[]
PERM_P2 : int[]
SIZE_KEY : int
SIZE_P1 : int
SIZE_P2 : int
SIZE_REAL_KEY : int
SIZE_SUBKEYS : int
key : char[]
ClausDES() : void
creaMeitats() : char[]
desplaCirEsq() : char[]
juntaMeitats() : char[]
permutar() : char[]
keys : char[][]
<Default Package>
XifratDES


<Default Package>
XifratDES
BITS_E_CAIXA : int
BITS_S_CAIXA : int
CAIXES : int[]
NCAIXES : int
NKEYS : int
PERM_PE : int[]
PERM_PL : int[]
PERM_PP : int[]
PERM_PS : int[]
SIZE_KEY : int
SIZE_PE : int
SIZE_PL : int
SIZE_PP : int
SIZE_PS : int
SIZE_REAL_KEY : int
accio : char
keys : char[][]
fer() : char[]
XifratDES() : void
binToDec() : int
creaMeitats() : char[]
decToBin() : String
funcioCaixa() : char[]
funcioF() : char[]
funcioXOR() : char[]
juntaMeitats() : char[]
permutar() : char[]
<Default Package>
ClausDES
Run
java.lang
String

- ## Aspectos concretos de la implementación

Cabe destacar que las clases principales del aplicativo de cifrado, que son ClausDES y XifratDES en lugar de operar con números binarios como seria de esperar lo hacen con tiras de unos y ceros. Esto es así ya que como he comentado en la especificación del problema la finalidad que persigue esta utilidad es poder hacer un seguimiento del proceso de cifrado que emplea este algoritmo.

Es evidente que mediante las cadenas de caracteres que representan los números binarios sobre los cuales opera DES, es mucho mas sencillo llevar a cabo un seguimiento de las distintas etapas y pasos que se van realizando.
En cambio, para el método de tratado de los bloques CBC y para el relleno, que no son parte integrante del sistema de cifrado ya se opera con bytes.

- ## Manual de instalación y funcionamiento

Para facilitar el proceso de instalación y ejecución de la utilidad de cifrado de ficheros, se ha creado una librería llamada “DES_Encryption.jar”.

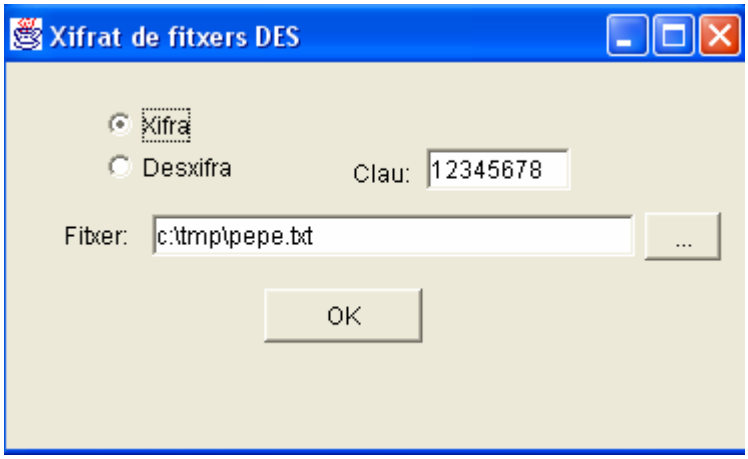
Estos son los pasos necesarios para la instalación:

1. Creación de una carpeta que albergará el aplicativo.
2. Copia de la librería “DES_Encryption.jar” en la carpeta del aplicativo.

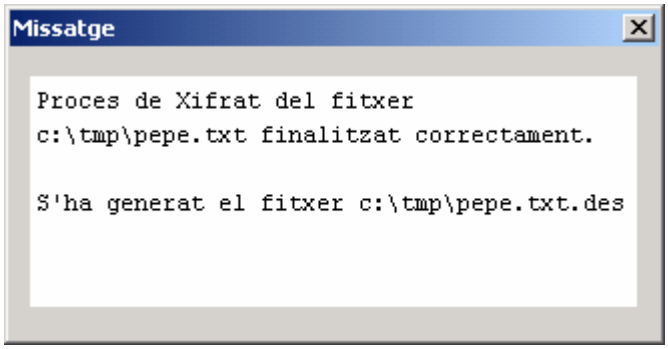
Para su ejecución debemos lanzar el intérprete de Java desde la carpeta del aplicativo con los siguientes argumentos:

java -classpath DES_Encryption.jar Application1

Acto seguido, nos tiene que aparecer la siguiente pantalla.



En ella podemos ver un radio button con que podemos seleccionar la operación a realizar, un campo de texto donde introducimos la clave secreta de hasta 8 caracteres y otro campo de texto en el que especificaremos mediante la ruta relativa o absoluta el fichero de entrada de cifrado o descifrado según la operación seleccionada.
Existe la posibilidad de abrir una ventana de búsqueda de ficheros para localizar el fichero de entrada navegando por el sistema de ficheros.



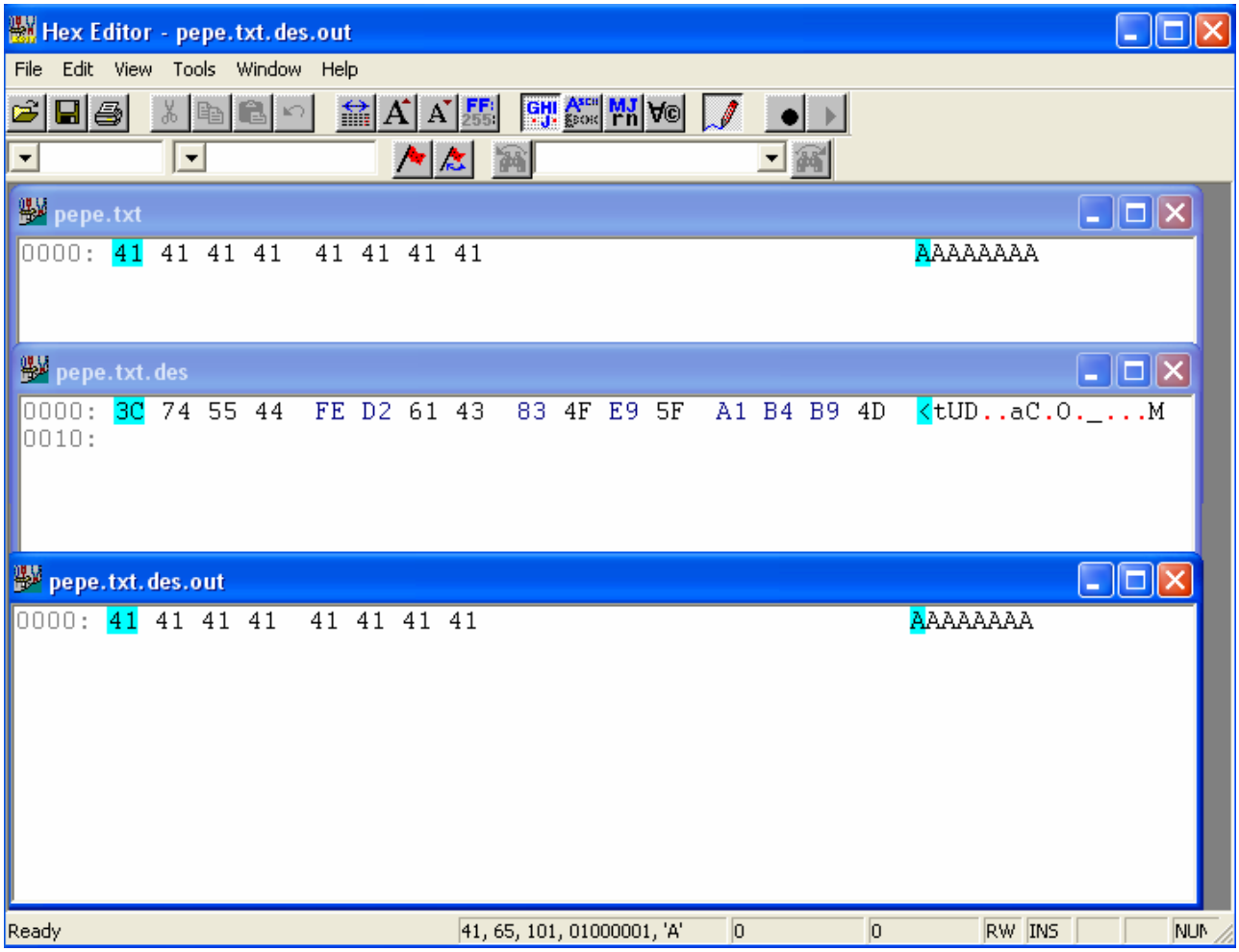
Esta es una pantalla de ejemplo que nos aparecerá al finalizar correctamente el cifrado de un fichero. En ella se puede observar que cuando ciframos un fichero al fichero generado le añade la extensión “.des”. En el caso del proceso de descifrado la extensión que añade al fichero generado es la “.out”.

Este aplicativo ha sido probado en los siguientes escenarios:

- Intel Celeron 2,0 GHz + 1Gb RAM + Windows 2000 Profesional SP4 + Java 1.3.1
- Intel Centrino 1,7 Ghz + 512Mb RAM + Windows XP Home SP2 + Java 1.3.1

- ## Juego de pruebas

En la siguiente imagen se puede observar los tres ficheros que se han generado durante los procesos de cifrado y descifrado utilizando la utilidad “DES Encryption”. El primero de ellos es el texto claro, el segundo el texto cifrado y el tercero el descifrado del anterior.



Detalle del proceso de creación de las subclaves

A continuación se detalla el proceso de creación de la primera de las 16 subclaves necesarias para el proceso de cifrado o descifrado. Para generar las 15 restantes basta con repetir el mismo ciclo a partir de la anterior subclave.

Siguiendo el esquema de generación de subclaves DES se parte de la clave introducida por el usuario, en nuestro caso: **K=”12345678”.**

Tabla de K									
0	0	1	1	0	0	0	~~1~~	=	1
0	0	1	1	0	0	1	~~0~~	=	2
0	0	1	1	0	0	1	~~1~~	=	3
0	0	1	1	0	1	0	~~0~~	=	4
0	0	1	1	0	1	0	~~1~~	=	5
0	0	1	1	0	1	1	~~0~~	=	6
0	0	1	1	0	1	1	~~1~~	=	7
0	0	1	1	1	0	0	~~0~~	=	8

Aplicamos la permutación P1 a la tabla de K.

Tabla P1						
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	1	1	1
1	1	1	1	1	1	1
0	1	1	0	0	1	1
0	0	1	1	1	1	0
0	0	1	0	0	0	0
0	0	0	1	1	1	1

Tenemos dos mitades a las que según la iteración (Ver la tabla de desplazamientos a la izquierda en el apartado de creación de subclaves dentro de la descripción del funcionamiento de DES) les aplicaremos un desplazamiento circular a la izquierda.

```
F0=0000000-0000000-0011111-1111111
G0=0110011-0011110-0010000-0001111
```

Desplazamiento circular a la izquierda DC1=1

```
F1=0000000-0000000-0111111-1111110
G1=1100110-0111100-0100000-0011110
```

Tabla F1+G1						
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	1	1	1	1	1	1
1	1	1	1	1	1	0
1	1	0	0	1	1	0
0	1	1	1	1	0	0
0	1	0	0	0	0	0
0	0	1	1	1	1	0

Aplicamos la permutación P2 a la tabla anterior.

Tabla P2					
0	1	0	1	0	0
0	0	0	0	1	0
1	1	0	0	1	0
1	0	1	1	0	0
0	1	0	1	0	1
1	1	0	0	1	0
1	0	1	0	1	1
0	0	0	0	1	0

Con lo cual obtenemos la primera de las subclaves.

Subclave K1=010100-000010-110010-101100-010101-110010-101011-000010

Para generar las siguientes repetimos el proceso. A partir de F1 y G1, aplicamos el desplazamiento circular a la izquierda correspondiente obtenemos así F2 y G2 aplicamos la permutación P2 y obtenemos la siguiente subclave y sucesivamente las restantes.
En el apartado siguiente, “Detalle del proceso de cifrado” podemos encontrar las 16 subclaves que se generan, con los nombres de subclau 1, subclau 2, etc…

Detalle del proceso de cifrado

A continuación se detalla el proceso completo de cifrado de un bloque en el que se pueden ver las 16 iteraciones que efectua el algoritmo.

```
Text:01234567 ← texto a cifrar
Clau:12345678 ← clave de cifrado

clave de cifrado en binario
clau         : 0011000100110010000110011001101000011010100110110001101110011100 0

texto a cifrar en binario
text         : 0011000000110001001100100011001100110100001101010011011000110111

permutación inicial del texto a cifrar (Tabla permutaciones Entrada)
perm inicial : 0000000011111111111100001010101000000000111111110000000011001100

E0: 0000000011111111111100001010101 0 ← mitad izquierda
D0: 0000000011111111000000001100110 0 ← mitad derecha


a partir de aquí empiezan las 16 iteraciones
============================================

*** inicio función de Feisel

resultado de la aplicación de la transformación L a la mitad derecha (Tabla Transf. L)
perm extesa 1: 0000000000001011111111110100000000001011001011000

subclau     1: 0101000000101100101011000101011100101010110000 10 ← subclave primera

resultado del XOR entre la permutación extendida y la subclave
XOR         1: 0101000000111011010100101101011100111100100110 10

resultado de cada una de las Cajas-S
caixa 1: 0110
caixa 2: 1101
caixa 3: 1000
caixa 4: 0010
caixa 5: 0000
caixa 6: 1110
caixa 7: 1111
caixa 8: 0000

concatenación del resultado de las Cajas-S
unio caixes 1: 01101101100000100000111011110000

resultado de la aplicación de la perumtación P (Tabla Permutaciones P) a las Cajas-S
perm P      1: 00010010011110001100011100011001

*** final función de Feisel

resultado del XOR con la mitad izquierda y la salida de la función de Feisel
XOR         1: 00010010100001110011011110110011

concatenación del XOR y la mitad derecha
Resultat FK-1: 0001001010000111001101111011001100000000111111110000000011001100

cruzado de las mitades izquierda y derecha de FK-1
Switch Res. 1: 0000000011111111000000001100110000100101000011100110111 10110011

E1: 0000000011111111000000011001100 ← mitad izquierda
D1: 0001001010000111001101111 0110011 ← mitad derecha
```

```
aquí empieza la segunda, tercera,.. iteración realizando los mismos pasos que se han
efectuado en la primera iteración
perm extesa 2: 10001010010101000000111010011010111111010100110
subclau     2: 01010000101011001010010001010000101000011010000111
XOR         2: 11011010111110001010101011001010010111101110000 1
caixa 1: 0111
caixa 2: 0010
caixa 3: 0110
caixa 4: 1011
caixa 5: 1001
caixa 6: 0010
caixa 7: 0010
caixa 8: 0010
unio caixes 2: 01110010011010111001001000100010
perm P      2: 11100001011000111000011001000110
XOR         2: 11100001100111001000011010001010
Resultat FK-2: 11100001100111001000011010001010000100101000011100110111110110011
Switch Res. 2: 00010010100001110011011110110011111000011001110010000110100010 10

perm extesa 3: 01110000001111001111100101000000110101000101010 1
subclau     3: 11010000101011000010011011110110100001001000110 0
XOR         3: 10100000100100001101111110110110010100000110110 01
caixa 1: 1101
caixa 2: 1111
caixa 3: 0111
caixa 4: 1001
caixa 5: 0010
caixa 6: 0010
caixa 7: 0000
caixa 8: 0000
unio caixes 3: 11011111011110010010001000000000
perm P      3: 11000100101010011100000011010110
XOR         3: 11010110001011101111011101100101
Resultat FK-3: 11010110001011101111011101100101111000011001110010000110100010 10
Switch Res. 3: 11100001100111001000011010001010110101100010111011111011101100101

perm extesa 4: 11101010110000010101110101111010111010110000 1011
subclau     4: 11100000101001100010011001001000001101111100 1011
XOR         4: 00001010011001110111011001100101101110011000 0000
caixa 1: 0100
caixa 2: 1011
caixa 3: 1111
caixa 4: 0111
caixa 5: 1011
caixa 6: 1111
caixa 7: 0101
caixa 8: 1101
unio caixes 4: 01001011111101111011111101011101
perm P      4: 11111111011110011111100110101100
XOR         4: 00011110111001010111111100100110
Resultat FK-4: 00011110111001010111111100100110110101100010111011111011101100101
Switch Res. 4: 11010110001011101111011101100101000111101110010101111111100100110

perm extesa 5: 00001111110101110000101010111111111010010000 1100
subclau     5: 11100000100101100010011000111110111100000010 1001
XOR         5: 11101111010000010010110010000001000110010010 0101
caixa 1: 0000
caixa 2: 1100
caixa 3: 1001
caixa 4: 0111
caixa 5: 0100
caixa 6: 0110
caixa 7: 1011
caixa 8: 1110
unio caixes 5: 00001100100101110100011010111110
perm P      5: 10001110011011100001010100111001
XOR         5: 01011000010000001110001001011100
```

```
Resultat FK-5: 0101100001000000111000100101110000011110111001010111111100100110
Switch Res. 5: 0001111011100101011111111001001100101110000100000011100001001011100

perm extesa 6: 0010111100000010000000010111000001000010111111000
subclau     6: 1110000010010010011100100110001001011101011000010
XOR         6: 1100111111001000011100110001001000011111110011010
caixa 1: 1011
caixa 2: 0000
caixa 3: 1101
caixa 4: 0100
caixa 5: 0100
caixa 6: 0100
caixa 7: 0010
caixa 8: 0000
unio caixes 6: 10110000110101000100010000100000
perm P      6: 00000100100001010001011100001010
XOR         6: 00011010011000000110100000101100
Resultat FK-6: 000110100110000000110100000101100010110000010000000111000100101110 0
Switch Res. 6: 0101100001000000111000100101110000011010011000000011010000010110 0

perm extesa 7: 0000111101000011000000000011010100000000101011000
subclau     7: 1010010011010010011100101000110010101001001110 10
XOR         7: 1010101110010001011100101011100110101000011000 10
caixa 1: 0110
caixa 2: 0000
caixa 3: 0000
caixa 4: 0001
caixa 5: 1000
caixa 6: 0111
caixa 7: 0110
caixa 8: 1011
unio caixes 7: 01100000000000011000011101101011
perm P      7: 10001001001100101010111000001000
XOR         7: 11010001011100100100110001010100
Resultat FK-7: 1101000101110010010011000101010000011010011000000011010000010 1100
Switch Res. 7: 0001101001100000001101000000101100110100001011100100100110001010100

perm extesa 8: 0110101000101011101001000010010110000001010101001
subclau     8: 1010011001010011010100101110010101011111001010000
XOR         8: 1100110001111000111101101100000011011110011111001
caixa 1: 1011
caixa 2: 0111
caixa 3: 1010
caixa 4: 1110
caixa 5: 1111
caixa 6: 1001
caixa 7: 0101
caixa 8: 0011
unio caixes 8: 10110111101011101111100101010011
perm P      8: 01110011110101100111101111010110
XOR         8: 01101001101101100001001111111010
Resultat FK-8: 0110100110110110000100111111101011010000101110010010011000101010 0
Switch Res. 8: 1101000101110010010011000101010000110100110110110000100111111101 0

perm extesa 9: 0011010100111101101011000000101001111111111110100
subclau     9: 0010011001010011010100111100101110011010010000000
XOR         9: 0001001101101110111111111110000011111001011011010 0
caixa 1: 1101
caixa 2: 0110
caixa 3: 0101
caixa 4: 1110
caixa 5: 1111
caixa 6: 1011
caixa 7: 0111
caixa 8: 1010
unio caixes 9: 11010110010111101111101101111010
perm P      9: 01111111111101111011010011010010
XOR         9: 10101110100001011111100010000110
```

```
Resultat FK-9: 10101110100001011111100010000110011010011011011000010011111111010
Switch Res. 9: 01101001101101100001001111111010101011101000010111111000010000110

perm extesa 10: 010101011101010000001011111111110001010000001101
subclau     10: 001011110101000101010001110100001100011100111100
XOR         10: 011110101000010101011010001011111101001100110001
caixa 1: 0111
caixa 2: 1010
caixa 3: 0101
caixa 4: 1100
caixa 5: 0111
caixa 6: 1000
caixa 7: 1000
caixa 8: 1111
unio caixes 10: 01111010010111000111100010001111
perm P      10: 01111100000011111001101011100011
XOR         10: 00010101101110011000100100011001101011101000010111111000010000110
Resultat FK-10: 00010101101110011000100100011001101011101000010111111000010000110
Switch Res. 10: 10101110100001011111100010000110000101011011100110001001000110001

perm extesa 11: 10001010101111011111001111000101001010011110010
subclau     11: 00001111010000011101100100011001000111101000110 0
XOR         11: 10000101111111000010101011011100001101100111111 0
caixa 1: 1111
caixa 2: 0101
caixa 3: 1011
caixa 4: 1011
caixa 5: 1001
caixa 6: 1111
caixa 7: 0010
caixa 8: 1000
unio caixes 11: 11110101101110111001111100101000
perm P      11: 10111101111000001110011101011110
XOR         11: 00010011011001010001111111011000
Resultat FK-11: 00010011011001010001111111011000000101011011100110001001000110001
Switch Res. 11: 00010101101110011000100100011001000100110110010100011111111011000

perm extesa 12: 00001010011010110000101010001111111111111011110000
subclau     12: 00011111010000011001100111011000011100000 10110001
XOR         12: 00010101001010101001001101010111100011100 1000001
caixa 1: 0111
caixa 2: 0111
caixa 3: 1111
caixa 4: 0111
caixa 5: 1111
caixa 6: 0001
caixa 7: 1110
caixa 8: 0001
unio caixes 12: 01110111111101111111000111100001
perm P      12: 11100101010101011111111110010111
XOR         12: 11110000111011000111011010001110
Resultat FK-12: 11110000111011000111011010001110000100110110010100011111111011000
Switch Res. 12: 00010011011001010001111111011000111100000111011000111011010001110

perm extesa 13: 01111010000101110101100000111010110101000101 1101
subclau     13: 00011111000010011000100100100011011010100010 1101
XOR         13: 01100101000111101101000100011001101111100111 0000
caixa 1: 1001
caixa 2: 1100
caixa 3: 0101
caixa 4: 0100
caixa 5: 0001
caixa 6: 1011
caixa 7: 1110
caixa 8: 0000
unio caixes 13: 10011100010101000001101111100000
perm P      13: 00110100101110010011010000010011
XOR         13: 00100111110111000010101111001011
```

```
Resultat FK-13: 00100111110111000010101111001011111100001110110001110110100011110
Switch Res. 13: 11110000111011000111011010001110001001111101110000101011111001011

perm extesa 14: 10010000111111101111100000010101011111001010110
subclau     14: 00011011001010001000110110110010001110011001000 10
XOR         14: 10001011110101100111010110100111010001111100010 0
caixa 1: 0001
caixa 2: 1110
caixa 3: 1100
caixa 4: 0101
caixa 5: 0001
caixa 6: 0100
caixa 7: 0110
caixa 8: 1000
unio caixes 14: 00011110110001010001010001101000
perm P      14: 11101000000110010001010100011010
XOR         14: 00011000111101010110001110010100
Resultat FK-14: 000110001111101010110001110010100001001111101110000101011111001011
Switch Res. 14: 00100111110111000010101111001011000110001111010101100011100101 00

perm extesa 15: 00001111000101111010101010110000011111001010100 0
subclau     15: 00011001001011001000110010100101000000110011011 1
XOR         15: 00010110001110110010011000010101011111111001111 1
caixa 1: 0111
caixa 2: 1000
caixa 3: 0011
caixa 4: 0000
caixa 5: 0010
caixa 6: 1110
caixa 7: 0010
caixa 8: 0010
unio caixes 15: 01111000001100000010111000100010
perm P      15: 00010100001010101000011010001110
XOR         15: 00110011111101101010110101000101
Resultat FK-15: 001100111111011010101101010000101000110001111010101100011100101 00
Switch Res. 15: 000110001111101010110001110010100001100111111101101010110101000101

perm extesa 16: 10011010011111111101011010101010110101010000001010
subclau     16: 01010001001011001000110010100111010000011110000000
XOR         16: 11001011010100110010000111110010111010011100 1010
caixa 1: 1100
caixa 2: 0111
caixa 3: 1111
caixa 4: 0011
caixa 5: 0000
caixa 6: 0011
caixa 7: 1000
caixa 8: 1111
unio caixes 16: 11000111111100110000001110001111
perm P      16: 11001100111000111111010010011010 1
XOR         16: 11010100000101101000101010100001
Resultat FK-16: 110101000001011010001010101000010011001111110110101011010100010 1
Switch Res. 16: 001100111111011010101101010001011101010000010110100010101010000 1

final de las iteraciones
========================

permutación final de FK-16 (Tabla permutaciones Salida)
perm final : 1000101110110100011110100000110011110000101010010110001001101101


Text xifrat hex:8BB47A0CF0A9626D
Text xifrat bin:1000101110110100011110100000110011110000101010010110001001101101
```

Detalle del proceso de descifrado

A continuación se detalla el proceso completo de descifrado de un bloque que corresponde al del proceso de cifrado anterior.

```
Text hex:8BB47A0CF0A9626D ← texto a descifrar
Clau:12345678 ← clave de cifrado/descifrado

clave de cifrado/descifrado en binario
clau         : 0011000100110010001100110011010000110101001101100011011100111000

texto a descifrar en binario
text         : 1000101110110100011110100000110011110000101010010110001001101101

permutación inicial del texto cifrado (Tabla permutaciones Entrada)
perm inicial : 1101010000010110100010101010000100110011111101101010110101000101

E0: 11010100000101101000101010100001 ← mitad izquierda
D0: 00110011111101101010110101000101 ← mitad derecha

a partir de aquí empiezan las 16 iteraciones en orden inverso
============================================================

*** inicio función de Feisel

Resultado de la aplicación de la transformación L a la mitad derecha (Tabla Transf. L)
perm extesa 16: 1001101001111111101011010101010110101010000001010

subclau     16: 0101000100101100100011001010011101000011111000000 ← subclave última

resultado del XOR entre la permutación extendida y la subclave
XOR         16: 1100101101010011001000011111001011101001110010010

resultado de cada una de las Cajas-S
caixa 1: 1100
caixa 2: 0111
caixa 3: 1111
caixa 4: 0011
caixa 5: 0000
caixa 6: 0011
caixa 7: 1000
caixa 8: 1111

concatenación del resultado de las Cajas-S
unio caixes 16: 11000111111100110000001110001111

resultado de la aplicación de la permutación P (Tabla permutaciones P) a las Cajas-S
perm P      16: 11001100111000111110100100110101

*** final función Feisel

resultado del XOR con la mitad izquierda y la salida de la función Feisel
XOR         16: 00011000111101010110001110010100

concatenación del XOR y la mitad derecha
Resultat FK-16: 0001100011110101011000111001010000110011111101101011010100010 1

cruzado de las mitades izquierda y derecha de FK-16
Switch Res. 16: 0011001111110110101011010100010100011000111101010110000111001010 0

E1: 00110011111101101010110101000101 ← mitad izquierda
D1: 00011000111101010110000111001010 0 ← mitad derecha
```

```
aquí empieza la segunda, tercera,.. iteración realizando los mismo pasos que se han
efectuado en la primera iteración
perm extesa 15: 000011110001011110101010101100000111110010101000
subclau     15: 000110010010110010001100101001010000001100110111
XOR         15: 000101100011101100100110000101010111111110011111
caixa 1: 0111
caixa 2: 1000
caixa 3: 0011
caixa 4: 0000
caixa 5: 0010
caixa 6: 1110
caixa 7: 0010
caixa 8: 0010
unio caixes 15: 011110000011000000101111000100010
perm P      15: 000101000010101010000011010001110
XOR         15: 001001111101110000101011110010110001100011110101010110001110010100
Resultat FK-15: 001001111101110000101011110010110001100011110101010110001110010100
Switch Res. 15: 000110001111101010110001110010100001001111101110000010101111001011

perm extesa 14: 100100001111111011111000000101010111111001010110
subclau     14: 000110110010100010001101101100100011100110010010
XOR         14: 100010111101011001110101101001110100011111000100
caixa 1: 0001
caixa 2: 1110
caixa 3: 1100
caixa 4: 0101
caixa 5: 0001
caixa 6: 0100
caixa 7: 0110
caixa 8: 1000
unio caixes 14: 000111101100010100001010001101000
perm P      14: 111010000001100100001010100011010
XOR         14: 111100001110110001110110100011100
Resultat FK-14: 111100001110110001110110100011100010011111101110000101011111001011
Switch Res. 14: 001001111101110000101011110010111111000011101100011101101000111 0

perm extesa 13: 011110100001011101011000001110101101010001011101
subclau     13: 000111110000100110001001001000110110101000101101
XOR         13: 011001010001111011010001000110011011111001110000
caixa 1: 1001
caixa 2: 1100
caixa 3: 0101
caixa 4: 0100
caixa 5: 0001
caixa 6: 1011
caixa 7: 1110
caixa 8: 0000
unio caixes 13: 100111000101010000011011111100000
perm P      13: 001101001011100100110100000010011
XOR         13: 000100110110010100011111111011000
Resultat FK-13: 000100110110010100011111111011000111100001110110001110110100011110
Switch Res. 13: 111100001110110001110110100011100001001101100101000111111101100 0

perm extesa 12: 000010100110101100001010100011111111111011110000
subclau     12: 000111110100000110011001110110000111000010110001
XOR         12: 000101010010101010010011010101111000111001000001
caixa 1: 0111
caixa 2: 0111
caixa 3: 1111
caixa 4: 0111
caixa 5: 1111
caixa 6: 0001
caixa 7: 1110
caixa 8: 0001
unio caixes 12: 011101111111011111110000111100001
perm P      12: 111001010101010111111111110010111
XOR         12: 000101011011100110001001000110001
```

```
Resultat FK-12: 00010101101110011000100100011001000100110110010100011111111011000
Switch Res. 12: 00010011011001010001111111011000000101011011100110001001000011001

perm extesa 11: 10001010101111011111001111000101001010100011110010
subclau     11: 00001111010000011101100100011001000111101000110 0
XOR         11: 10000101111111000010101011011100001101100111111 0
caixa 1: 1111
caixa 2: 0101
caixa 3: 1011
caixa 4: 1011
caixa 5: 1001
caixa 6: 1111
caixa 7: 0010
caixa 8: 1000
unio caixes 11: 11110101101110111001111100101000
perm P      11: 10111101111000001110011101011110
XOR         11: 10101110100001011111100010000110
Resultat FK-11: 10101110100001011111100010000110000101011011100110001001000011001
Switch Res. 11: 00010101101110011000100100011001101011101000010111111000100 00110

perm extesa 10: 01010101110101000000101111111111000101010000001101
subclau     10: 00101111010100010101000111010000110001110011110 0
XOR         10: 01111010100001010101101000101111111010011001100 01
caixa 1: 0111
caixa 2: 1010
caixa 3: 0101
caixa 4: 1100
caixa 5: 0111
caixa 6: 1000
caixa 7: 1000
caixa 8: 1111
unio caixes 10: 01111010010111000111100010001111
perm P      10: 01111100000011111001101011100011
XOR         10: 01101001101101100001001111111010
Resultat FK-10: 01101001101101100001001111111010101011101000010111111000100 00110
Switch Res. 10: 10101110100001011111100010000110011010011011011000010011111111010

perm extesa 9: 00110101001111011010110000001010011111111110100
subclau     9: 00100110010100110101001111001011100110100100000 0
XOR         9: 00010011011011101111111111000001111001011011 0100
caixa 1: 1101
caixa 2: 0110
caixa 3: 0101
caixa 4: 1110
caixa 5: 1111
caixa 6: 1011
caixa 7: 0111
caixa 8: 1010
unio caixes 9: 11010110010111101111101101111010
perm P      9: 01111111111101111011010011010010
XOR         9: 11010001011100100100110001010100
Resultat FK-9: 11010001011100100100110001010100011010011011011000010011111111010
Switch Res. 9: 01101001101101100001001111111010110100010111001001001100010101 00

perm extesa 8: 01101010001010111010010000100101100000101010 1001
subclau     8: 10100110010100110101001011100101010111100101 0000
XOR         8: 11001100011110001111011011000000110111001111 1001
caixa 1: 1011
caixa 2: 0111
caixa 3: 1010
caixa 4: 1110
caixa 5: 1111
caixa 6: 1001
caixa 7: 0101
caixa 8: 0011
unio caixes 8: 10110111101011101111100101010011
perm P      8: 01110011110101100111101111010110
XOR         8: 00011010011000000110100000101100
```

```
Resultat FK-8: 0001101001100000011010000010110011010001011100100100110001010100
Switch Res. 8: 1101000101110010010011000101010000011010011000000110100000101100

perm extesa 7: 0000111101000011000000000011010100000000101011000
subclau     7: 1010010011010010011100101000110010101001001111010
XOR         7: 1010101110010001011100101011100110101000011000010
caixa 1: 0110
caixa 2: 0000
caixa 3: 0000
caixa 4: 0001
caixa 5: 1000
caixa 6: 0111
caixa 7: 0110
caixa 8: 1011
unio caixes 7: 01100000000000011000011101101011
perm P      7: 10001001001100101010111000001000
XOR         7: 01011000010000001110001001011100
Resultat FK-7: 0101100001000000111000100101110000011010011000000110100000101100
Switch Res. 7: 0001101001100000011010000010110001011000001000000111000100101100

perm extesa 6: 0010111100000010000000010111000001000010111111000
subclau     6: 1110000010010010011100100110001001011101011000010
XOR         6: 1100111111001000011100110001001000011111110011010
caixa 1: 1011
caixa 2: 0000
caixa 3: 1101
caixa 4: 0100
caixa 5: 0100
caixa 6: 0100
caixa 7: 0010
caixa 8: 0000
unio caixes 6: 10110000110101000100010000100000
perm P      6: 00000100100001010000101110000101 0
XOR         6: 0001111011100101011111111001001 10
Resultat FK-6: 000111101110010101111111100100110010110000100000011100010010111 00
Switch Res. 6: 010110000100000011100010010111000001111011100101010111111001001 10

perm extesa 5: 0000111111010111000010101011111111110100100001100
subclau     5: 1110000010010110001001100011111011111000000101001
XOR         5: 1110111101000001001011001000000100011001001001 01
caixa 1: 0000
caixa 2: 1100
caixa 3: 1001
caixa 4: 0111
caixa 5: 0100
caixa 6: 0110
caixa 7: 1011
caixa 8: 1110
unio caixes 5: 00001100100101110100011010111110
perm P      5: 10001110011011100001010100111001
XOR         5: 1101011000101110111101101100101
Resultat FK-5: 110101100010111011111011011001010001111011100101011111111001001 10
Switch Res. 5: 000111101110010101111111100100110110101100010111011111011011001 01

perm extesa 4: 1110101011000001010111010111101011101011000010 11
subclau     4: 1110000010100110001001100100100000110111111001011
XOR         4: 0000101001100111011110110011001011011100110000000
caixa 1: 0100
caixa 2: 1011
caixa 3: 1111
caixa 4: 0111
caixa 5: 1011
caixa 6: 1111
caixa 7: 0101
caixa 8: 1101
unio caixes 4: 01001011111101111011111101011101
perm P      4: 11111111011110011111100110101100
XOR         4: 1110000110011100100001101000 1010
```

```
Resultat FK-4: 11100001100111001000011010001010110101100010111011111011101100101
Switch Res. 4: 11010110001011101111011101100101111000011001110010000011010001010

perm extesa 3: 01110000001111001111100101000000110101010001010101
subclau     3: 11010000101011000010011011110110100001001000110 0
XOR         3: 10100000100100001101111110110110010100000110110 01
caixa 1: 1101
caixa 2: 1111
caixa 3: 0111
caixa 4: 1001
caixa 5: 0010
caixa 6: 0010
caixa 7: 0000
caixa 8: 0000
unio caixes 3: 11011111011110010010001000000000
perm P      3: 11000100101010011100000011010110
XOR         3: 00010010100001110011011110110011
Resultat FK-3: 00010010100001110011011110110011111000011001110010000011010001010
Switch Res. 3: 11100001100111001000011010001010000100101000011100110111110110011

perm extesa 2: 10001010010101000000111010011010111110110100110
subclau     2: 01010000101011001010010001010000101000110100011 1
XOR         2: 11011010111110001010101011001010010111011100001
caixa 1: 0111
caixa 2: 0010
caixa 3: 0110
caixa 4: 1011
caixa 5: 1001
caixa 6: 0010
caixa 7: 0010
caixa 8: 0010
unio caixes 2: 01110010011010111001001000100010
perm P      2: 11100001011000111000011001000110
XOR         2: 00000000111111110000000011001100
Resultat FK-2: 00000000111111110000000011001100000100101000011100110111110110011
Switch Res. 2: 00010010100001110011011110110011000000000111111110000000011001100
perm extesa 1: 00000000000101111111111101000000000010110011011000
subclau     1: 01010000001011001010110001010111001010101100001 0
XOR         1: 01010000001110110101001011010111001111100100110 10
caixa 1: 0110
caixa 2: 1101
caixa 3: 1000
caixa 4: 0010
caixa 5: 0000
caixa 6: 1110
caixa 7: 1111
caixa 8: 0000
unio caixes 1: 01101101100000100000111011110000
perm P      1: 00010010011110001100011100011001
XOR         1: 00000000111111111111100001010101
Resultat FK-1: 00000000111111111111100001010101000000000111111111000000011001100
Switch Res. 1: 00000000111111110000000011001100000000000111111111111000010101010

final de las iteraciones
========================

permutación final de FK-1 (Tabla de permutaciones Salida)
perm final: 00110000001100010011001000110011001101000011010100110110001101 11

Text desxifrat:01234567
Text desxifrat hex:3031323334353637
```

- ## Herramientas utilizadas

Como herramientas utilizadas para la implementación del proyecto he utilizado:

1. Imendio Planner 0.13. Gestor de proyectos para el entorno del escritorio Gnome GNU/Linux.
2. MS Word 97 SR-1. Editor de textos MS Windows.
3. MS PowerPoint 97 SR-1. Creación de diapositivas para presentaciones MS Windows.
4. Borland JBuilder 6.0. Entorno de desarrollo en Java MS Windows.

• Comentarios y conclusiones

Es curioso constatar que a pesar de haberse publicado una enorme cantidad de información sobre el criptoanálisis de DES mas que de ningún otro cifrado de bloque, y de ello hace ya cerca de 30 años, el ataque al que es mas vulnerable sigue siendo por fuerza bruta.

Al principio de este trabajo se establecieron unos objetivos. Estos eran el realizar un aplicativo de cifrado de ficheros basado en el muy conocido sistema DES y que además debido a su diseño modular, fuera sencillo el seguir su funcionamiento obteniendo así otro de los objetivos perseguidos, que la implementación del sistema de cifrado tuviera un carácter didáctico y que para futuros estudios de este algoritmo pueda serle de utilidad a algún estudiante. He de señalar además, que no he considerado necesario complicar mas el aplicativo añadiendo una serie de rutinas de llamada al sistema de ficheros para logar encaminar al proceso de cifrado el contenido de un directorio que se alejan del objetivo marcado para este proyecto.
Creo que ambos objetivos se han cumplido, se ha creado un sistema de cifrado de ficheros basado en DES y se ha llevado a cabo su implementación de una forma clara y que ayuda a comprender como opera. Lo que no puedo asegurar es que el carácter instructivo que he intentado imprimirle al trabajo sea de utilidad en los futuros estudios y trabajos de Criptografía.

Como satisfacción personal, me gustaría que algún día el presente trabajo le pudiera ser útil a alguien para poder seguir fácilmente el hilo del algoritmo y llegar a entender como consigue su objetivo, que es enmascarar un texto claro.
De hecho, he de señalar que a mi personalmente me ha servido para conocer de una forma mas profunda este simple y a la vez eficiente sistema de cifrado.

• Bibliografía

Fúster Sabater, Amparo. (1997).
Técnicas criptográficas de protección de datos. Madrid: Ra-ma.

Herrera Joancomartí, Jordi. (2006).
Xifres de clau compartida: xifres de bloc (Mòdul 4). UOC Materials assignatura Criptografia.

Pastor Franco, José / Sarasa López, M.Angel
Criptografía digital fundamentos y aplicaciones.

Lima Diaz, Felipe. (1998).
Manual avanzado de Java. Anaya Multimedia.

Martinez, Rafael
http://www.monografias.com
Explicación del cifrado en bloques simétrico DES.

http://es.wikipedia.org
Data Encryption Standard