

Instituto Tecnológico

de Durango

**Ing. En Sistemas Computacionales**

**Criptografía**

**Encriptado: “DES”**

**N° de Control: 12041156**

**Nombre: Mauricio Alejandro Martínez Pacheco**

**GRUPO: 7YZ**

**23 DE FEBRERO DE 2015**

**Victoria de Durango, Dgo.**

Introducción

En la clase de criptografía se nos encargó implementar el algoritmo de cifrado de DES en el lenguaje de programación Java.

En sí es un algoritmo sencillo, pero muy laborioso de implementar, por lo que realizarlo, solo implicó trabajo mecánico y no tanto intelectual, ya que el algoritmo ya estaba descrito por Stallings.

Desarrollo

Comenzando a desarrollar el algoritmo, me topo que solamente se encriptan 64 bits de texto plano y 64 de la llave, no más, ni menos. Por lo que primeramente me puse a desarrollar una manera de permitir entradas menores a 64 y mayores, esto en el texto plano. En la llave solo permito menos de 64 bits, ya que en el texto plano, más de 64 bits solo significa ejecutar el DES más veces, pero en la llave no es posible.

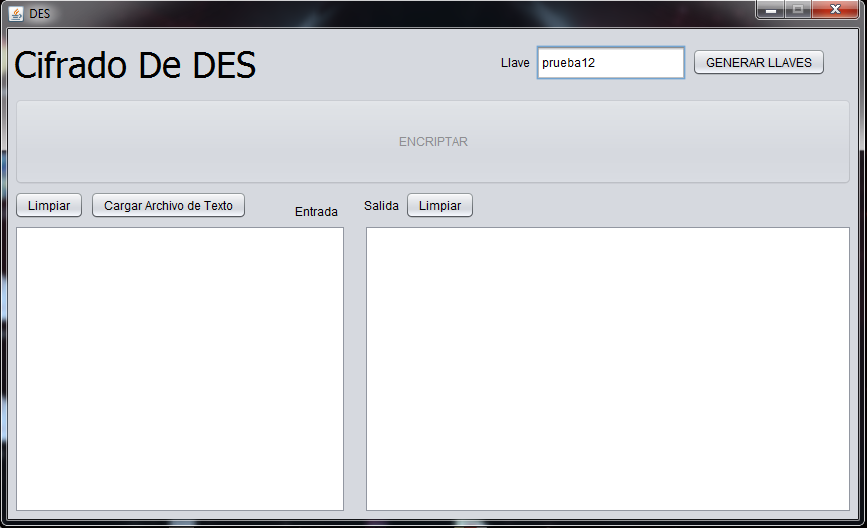
Para esto utilizo un objeto llamado Vector, éste se comporta como un arreglo dinámico. En este objeto guardo arreglos de tipo booleano, los cuales son los paquetes de 64 bits que pasaran por DES.

Vector entradabits=new Vector();

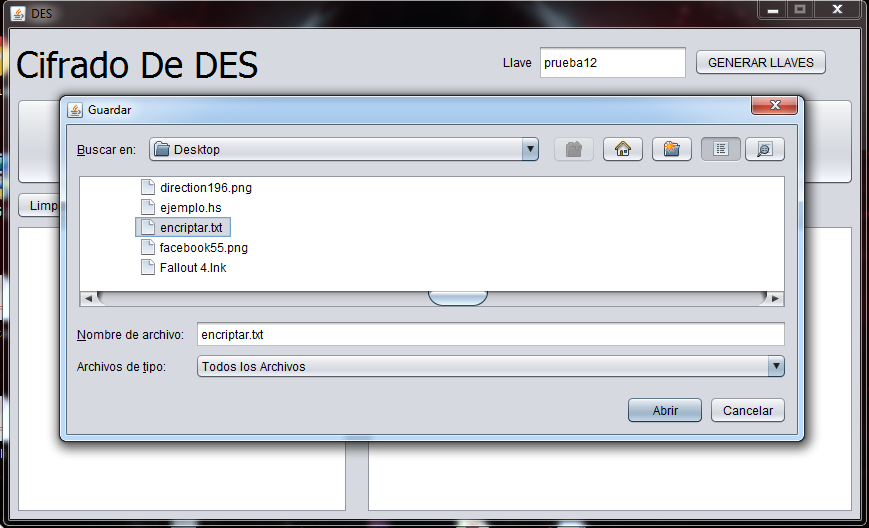
Este objeto tiene un método llamado .size() que nos retorna el número de elementos que contiene, esto nos va a servir para darle el límite al ciclo que ejecutará n veces el algoritmo de DES además de ir concatenando el encriptado en el jTextArea de salida.

La generación de llaves la manejo en un método aparte, esto con el objetivo de hacer una sola generación de las 16 llaves, para ahorrar tiempo de ejecución y por si deseamos encriptar varios textos planos con la misma llave.

Una vez que se generan las llaves, el botón de ”ENCRIPTAR” se habilita y ya podemos realizar el proceso.



La entrada la tengo en una caja de texto, la cual permite la escritura directa de caracteres o cargar un archivo de texto .txt.



Al darle al botón de encriptar se toma el contenido del jTextArea de entrada, lo convierte a un String y después mediante un ciclo se va convirtiendo carácter por carácter a un String que contiene los 8 bits en binario por ejemplo: el carácter “p” es el 112 ASCII, esto equivale a “01110000”.

private String Binario(int Decimal) {

String resultado="";

int temp=Decimal;

//Se aplica el algoritmo para convertir a binario

while(temp != 0){

if(temp %2 == 0){

resultado="0"+resultado;

}else{

resultado="1"+resultado;

}

temp = temp/2;

}

//Se le agregan ceros a la izquierda, para completar 8 bits por caracter

int cerosalaizquierda=8-resultado.length();

for(int x=0;x<cerosalaizquierda;x++){

resultado="0"+resultado;

}

return resultado;

}

Ahora se toma esa cadena y se evalúa dentro de un ciclo si su longitud es menor a 64 o mayor e igual. Si es menor de 64 se calculan cuantos ceros le falta al paquete para ser de 64 bits y se agregan a la izquierda de la cadena. Si es mayor o igual a 64 se va rellenando un arreglo booleano de 64 posiciones con true si el carácter de la cadena es ‘1’ y false si no es, después se inserta ese arreglo booleano en el objeto Vector y se le extrae a la cadena principal esos 64 bits y si queda una cadena nula se termina el ciclo, pero si aún quedan bits se repite el proceso de evaluar su longitud.

Ya teniendo el objeto Vector con los paquetes de 64 bits, se procede a llamar al método encriptar() de la clase DES. Que el algoritmo esté en otra clase nos facilita la llamada de n veces al algoritmo.

private void encriptarbtnActionPerformed(java.awt.event.ActionEvent evt) {

salida.setText("");

if(!"".equals(entrada.getText())&&!"".equals(llave.getText())){

Vector entradabits=new Vector();

boolean[] paquete;

String binario="";

//Se convierte el contenido del jTextArea a un arreglo de caracteres

String entradaarray=entrada.getText().trim().toString();

//Mediante el ciclo se convierte caracter a caracter a binario de 8 bits

for(int i=0;i<entradaarray.length();i++){

binario=binario+Binario(entradaarray.charAt(i));

}

//Ahora obtendremos los paquetes de 64 bits de la cadena ya convertida a binario

//Una variable auxiliar para almacenar el string de tamaño 64

String sub;

//Este ciclo se repetirá tantas veces como elementos tenga la cadena binario

while(!"".equals(binario)){

//Si la longitud de binario es menor a 64 ya no completamos otro paquete

if(binario.length()<64){

//Vaciamos lo que tenga el arreglo paquete para almacenar el nuevo

paquete=new boolean[64];

//Calculamos cuantos bits le hacen falta al paquete para tener una longitud de 64

int restante=64-binario.length();

//Aquí rellenamos esa cantidad restante de bits con ceros

for(int x=0;x<restante;x++){

binario="0"+binario;

}

//Ahora sacamos todos los caracteres del paquete y verificamos si es un uno o cero para llenar el arreglo de booleanos

System.out.print("Paquete : ");

for(int x=0;x<64;x++){

paquete[x] = binario.charAt(x)=='1';

System.out.print(paquete[x]+", ");

}

System.out.print("\n");

//Agregamos el arreglo paquete al objeto Vector para separarlo

entradabits.add(paquete);

//Se sale del ciclo ya que se acabaron los paquetes

break;

}else{

//Si la longitud es mayor o igual a 64 entonces aun nos quedan paquetes por procesar

//Vaciamos lo que tenga el arreglo paquete para almacenar el nuevo

paquete=new boolean[64];

//Extraemos una subcadena de 64 posiciones de la cadena binario para obtener los 64 bits

sub=binario.substring(0, 64);

//Ahora sacamos todos los caracteres del paquete y verificamos si es un uno o cero para llenar el arreglo de booleanos

System.out.print("Paquete : ");

for(int x=0;x<64;x++){

paquete[x] = sub.charAt(x)=='1';

System.out.print(paquete[x]+", ");

}

System.out.print("\n");

//Agregamos el arreglo paquete al objeto Vector para separarlo

entradabits.add(paquete);

//Eliminamos esos 64 bits procesados de la cadena para continuar con los otros 64

binario=binario.substring(64,binario.length());

}

}

//Dependiendo del tamaño del objeto Vector es las veces que se ejecutará DES, debido que solo puede encriptar 64 bits

for(int y=0;y<entradabits.size();y++){

paquete=(boolean [])entradabits.get(y);

System.out.println("ENTRADABITS");

for(int x=0;x<64;x++){

System.out.print(paquete[x]+", ");

}

System.out.println("");

salida.setText(salida.getText()+ASCII(des.encriptar(paquete)));

}

}else{

JOptionPane.showMessageDialog(null, "INGRESA TEXTO EN LA ENTRADA O CARGA UN ARCHIVO","ERROR",JOptionPane.ERROR\_MESSAGE);

}

}

En esta clase llamada DES, el constructor va a recibir como parámetro la llave, esto para la generación única de las 16 llaves, en este constructor se inicializan todas las matrices del algoritmo y al final se manda llamar al método obtencionLlaves() en el cual se realiza el algoritmo de generación de llaves.

La generación de llaves comienza con la permutación de opción uno, la cual nos reducirá la llave de 64 a una de 56 bits, además esta misma matriz nos parte los 56 en dos mitades de 28, que decidí almacenarlas en dos arreglos booleanos de 28 bits.

Después en un ciclo de 16 vueltas se realizan los corrimientos a la izquierda de bits dependiendo del Schedule, el Schedule lo metí dentro de otro arreglo, así se sabe cuál se va a usar en la vuelta n. Después de correr los bits, se respaldan las dos mitades para la siguiente llave y se realiza la permutación de opción 2. Decidí utilizar una matriz de 16 renglones por 48 columnas que está declarada como un atributo de la clase, esto quiere decir que puede ser utilizada por todos los métodos de esa clase, así es como se puede generar las llaves una sola vez y encriptar tantos textos planos queramos con esa llave. Entonces ya realizada la permutación se almacenan las dos mitades en un renglón de la matriz de 16 llaves y así se repite el proceso.

private boolean[][] obtencionllaves(boolean[]llave){

boolean llavesgen[][]=new boolean[16][48];

boolean llaveizq[]=new boolean[28];

boolean llaveder[]=new boolean[28];

int cont=0;

//PERMUTACION OPCION 1

System.out.print("\nLlave: ");

for(int x=0;x<64;x++){

System.out.print(llave[x]+", ");

}

for (int a = 0; a < 4; a++) {

for (int b = 0; b < 7; b++) {

llaveizq[cont]=llave[APC1[a][b]-1];

llaveder[cont]=llave[BPC1[a][b]-1];

cont++;

}

}

System.out.print("\nLlave izq: ");

for(int x=0;x<28;x++){

System.out.print(llaveizq[x]+", ");

}

System.out.print("\nLlave der: ");

for(int x=0;x<28;x++){

System.out.print(llaveder[x]+", ");

}

//GENERACION DE 16 SUBLLAVES

boolean temp[]=new boolean[56];

boolean []llaveizqantes=new boolean[28];

boolean []llavederantes=new boolean[28];

llaveizqantes=llaveizq;

llavederantes=llaveder;

for(int x=0;x<16;x++){

//CORRIMIENTO A LA IZQ SEGUN EL SCHEDULE

llaveizq=rotarIzq(llaveizqantes, SCH[x]);

System.out.println("SCHEDULE: "+SCH[x]);

llaveder=rotarIzq(llavederantes, SCH[x]);

llaveizqantes=llaveizq;

llavederantes=llaveder;

//JUNTAR LOS DOS ARREGLOS PARA HACER LA PERMUTACION OPCION 2

for(int b=0;b<28;b++){

temp[b]=llaveizq[b];

}

cont=0;

for(int b=28;b<56;b++){

temp[b]=llaveder[cont];

cont++;

}

cont=0;

//PERMUTACION OPCION 2

boolean permutado2[]=new boolean[48];

for (int a = 0; a < 6; a++) {

for (int b = 0; b < 8; b++) {

permutado2[cont] = temp[PC2[a][b]-1];

if(cont<48){

cont++;

}

}

}

//GUARDAR LAS SUBLLAVES EN EL ARREGLO DE 16

for(int y=0;y<48;y++){

llavesgen[x][y]=permutado2[y];

}

}

return llavesgen;

}

Dentro de la clase esta el método encriptar(), el cual es en donde está todo el algoritmo de las redes de feistel, etc. Este método recibe como parámetro el arreglo booleano que contiene el paquete de 64 bits que se va a encriptar.

Primeramente en el método se realiza la permutación inicial, aquí es donde implemente estos for anidados para realizarlo más eficientemente, me parece muy limpia esta implementación, porque otros compañeros tenían muchas más líneas de código.

int cont=0;

for(int a=0;a<8;a++){

for(int b=0;b<8;b++){

permutado[cont]=paquete[IP[a][b]-1];

System.out.print(permutado[cont]+", ");

cont++;

}

}

Después hago la primera separación de las mitades, en donde cada una será de 32 bits. Aquí empiezan las 16 rondas, que simplemente están en un for de 16 vueltas. Al principio de la ronda, hago un backup de la mitad derecha en una variable auxiliar, para que al final de la ronda pase a ser la mitad izquierda, esto porque si se asigna la mitad derecha a la izquierda desde el inicio se pierde la mitad izquierda para la operación XOR.

boolean[] izq=new boolean[32];

boolean[] der=new boolean[32];

boolean[] derantes=new boolean[32];

//Separacion del paquete para mitad izq y der

for(int x=0;x<32;x++){

izq[x]=permutado[x];

der[x]=permutado[x+32];

}

A continuación es la permutación de expansión, que esta igual en un for como la inicial, solo que claro, se manda llamar la matriz de expansión.

cont=0;

char bit='1';

for (int a = 0; a < 8; a++) {

for (int b = 0; b < 6; b++) {

if(der[E[a][b]-1]==false){

bit='0';

}else{

bit='1';

}

System.out.println("Bit: "+bit+" Es el de la pos: "+(E[a][b]-1));

expansion[cont] = der[E[a][b]-1];

//System.out.print(expansion[cont]+", ");

cont++;

}

}

En seguida viene el XOR con la llave de la ronda

for (int a = 0; a < 48; a++) {

xor[a] = expansion[a] != llaves[i][a];

}

La tabla de verdad del XOR es que si son iguales es false y si son diferentes es true, entonces así se puede expresar sin la necesidad de if, ya que la expresión retorna booleano. El índice i es el de la ronda actual.

Después siguen las S-Boxes. Aquí declare las S-Boxes como un arreglo tridimensional, en donde se puede ver como un vector de matrices, esto para que el proceso de S-Box sea solo un ciclo de 8 vueltas.

Para preparar la entrada de 8 paquetes de 6 bits, dividí el arreglo principal en una matriz de 8 renglones por 6 columnas.

cont=0;

for (int a = 0; a < 8; a++) {

for (int b = 0; b < 6; b++) {

entradasbox[a][b] = xor[cont];

cont++;

}

}

Ahora tomo la posición 0 y la 5 que son el primero y el ultimo, y representan la fila de la sbox, también los de las posiciones 1 a la 4 que representan las columnas. Con un método los convierto a decimal. Así ya tengo la S-Box que se va a usar (con el contador del ciclo de 8 vueltas), la fila y la columna.

Sboxes[a][binarioADecimal(dosbits)][binarioADecimal(cuatrobits)]

Esto regresará un numero entero del 0 al 15, hice un método que convierte este entero a un arreglo de booleanos de 4 posiciones en binario.

private int binarioADecimal(boolean []bits){

int decimal=0;

String bitstring="";

int longitud=bits.length;

int potencia = longitud - 1;

//Convertimos el arreglo booleano a un String

for(int x=0;x<longitud;x++){

if(bits[x]==true){

bitstring=bitstring+"1";

}else{

bitstring=bitstring+"0";

}

}

//Conversion del string binario a numero entero

for(int x=0;x<longitud;x++){

if(bitstring.charAt(x)=='1'){

decimal+= Math.pow(2, potencia);

}

potencia --;

}

return decimal;

}

for (int a = 0; a < 8; a++) {

//Separación de los 4 bits de enmedio para la columna de la sbox

dosbits[0]=entradasbox[a][0];

cuatrobits[0]=entradasbox[a][1];

cuatrobits[1]=entradasbox[a][2];

cuatrobits[2]=entradasbox[a][3];

cuatrobits[3]=entradasbox[a][4];

dosbits[1]=entradasbox[a][5];

boolean temp[]=Binario(Sboxes[a][binarioADecimal(dosbits)][binarioADecimal(cuatrobits)]);

for(int b=0;b<4;b++){

sboxeado[cont]=temp[b];

cont++;

}

}

Después sigue la permutación p que es como las otras permutaciones en un ciclo for.

boolean []permutacionp=new boolean[32];

cont=0;

for(int a=0;a<4;a++){

for(int b=0;b<8;b++){

permutacionp[cont]=sboxeado[P[a][b]-1];

cont++;

}

}

Luego el resultado se le aplica un XOR con la mitad izquierda y esto queda en la mitad derecha, el backup que hicimos desde el inicio de la ronda se asigna como la nueva izquierda y comienza la ronda de nuevo.

for(int x=0;x<32;x++){

der[x]=permutacionp[x] != izq[x];

}

for(int x=0;x<32;x++){

izq[x]=derantes[x];

}

Al finalizar las 16 rondas, se hace el intercambio de mitades, la izquierda pasa a ser la derecha y la derecha la izquierda. Y por ultimo, se realiza la permutación inicial inversa.

//PEGAR MITADES

for(int x=0;x<32;x++){

paquete[x]=der[x];

paquete[x+32]=izq[x];

}

//PERMUTACION INVERSA

boolean [] permutadofinal=new boolean[64];

cont=0;

for(int a=0;a<8;a++){

for(int b=0;b<8;b++){

permutadofinal[cont]=paquete[IPR[a][b]-1];

cont++;

}

}

El resultado lo convierto a un String de 64 caracteres “booleanos”. Y este String es el valor que regresa el método, esto para pasarlo a la interfaz gráfica en donde hay un método para convertir ese String a ASCII.

//Concatenado del arreglo a un string

for(int x=0;x<64;x++){

if(permutadofinal[x]==true){

encriptado=encriptado+"1";

}else{

encriptado=encriptado+"0";

}

}

return encriptado;

Este método, separa el String en 8 substrings de 8 caracteres, ya que un ASCII es de 8 bits. Después se realiza la conversión de esos 8 bits a entero, y ese entero ya es el carácter, ya que java expresa los caracteres como valores enteros. Y voy concatenando dichos caracteres en un String y ya terminado se muestra en el jTextField de salida.

private String ASCII(String binario){

String caracter="";

String caracteres[]=new String[8];

String ascii="";

int valor=0;

//Separamos la cadena binario en 8 subcadenas de 8 bits

int cont=8;

for(int a=0;a<8;a++){

caracteres[a]=binario.substring(cont-8, cont);

System.out.println("Caracter "+(a+1)+" :"+caracteres[a]);

cont=cont+8;

}

//Conversion del string binario a numero entero

for(int a=0;a<8;a++){

int longitud=caracteres[a].length();

int potencia = longitud - 1;

valor=0;

for(int b=0;b<longitud;b++){

if(caracteres[a].charAt(b)=='1'){

valor+= Math.pow(2, potencia);

}

potencia --;

}

System.out.println("Caracter "+(a+1)+" :"+valor+" ASCII: "+(char)valor);

ascii=ascii+(char)valor;

}

return ascii;

}

Conclusión

El algoritmo es muy sencillo, pero muy laborioso. Además muy complicado de estar comprobando que los valores que van dando son los correctos. Para la comprobación utilicé el ejemplo del libro convirtiendo los valores hexadecimales del libro a binario

