Universidad del Valle de Guatemala
Facultad de ingeniería
Programación de microprocesadores

Proyecto 2

Mario De León
Roberto Castillo
Hugo Román

Introducción

La privacidad es un aspecto muy importante en la vida de todos. Desde contraseñas de redes sociales hasta documentos empresariales se pueden ver amenazados hoy en día. Debido a esto se han desarrollado distintas maneras de mantener seguros nuestras pertenencias. Uno de los métodos más conocidos y utilizados antiguamente es DES. Este es un algoritmo de cifrado, es decir, un método para cifrar información. A pesar de que se ha demostrado que tiene debilidades, debido a su longitud de clave relativamente corta, se ha seguido utilizando.

Algoritmo DES

Especificaciones necesarias de algoritmo DES

El algoritmo DES (Data Encryption Standard) trata de un algoritmo cifrado, su propósito es transformar mediante una serie de operaciones en otro texto totalmente distinto al principal, lo primero a considerar es que el tamaño es de 64bits, Como funciona el algoritmo es con los 64bits solo 56 son utilizados por las operaciones y los 8 restantes solo se utilizan para comprobar si todo está en orden.Las especificaciones para incorporar el algoritmo es únicamente un dispositivo electrónico LSI especializado ya que se podrá ingresar una clave de cifrado directamente en el dispositivo si aparecer en ningún otro lado del sistema,
Por ende, FIPS PUB 46 requiere implementación de propósito especial, en pocas palabras un dispositivo electrónico o alguna memoria de solo lectura.

Antecedentes

El algoritmo DES (Data Encryption Standard, estándar de cifrado de datos) es un algoritmo desarrollado originalmente por IBM a requerimiento del NBS (National Bureau of Standards, Oficina Nacional de Estandarización, en la actualidad denominado NIST, National Institute of Standards and Technology, Instituto Nacional de Estandarización y Tecnología) de EE.UU. y posteriormente modificado y adoptado por el gobierno de EE.UU. En 1977 como estándar de cifrado de todas las informaciones sensibles no clasificadas. Posteriormente, en 1980, el NIST estandarizó los diferentes modos de operación del algoritmo.

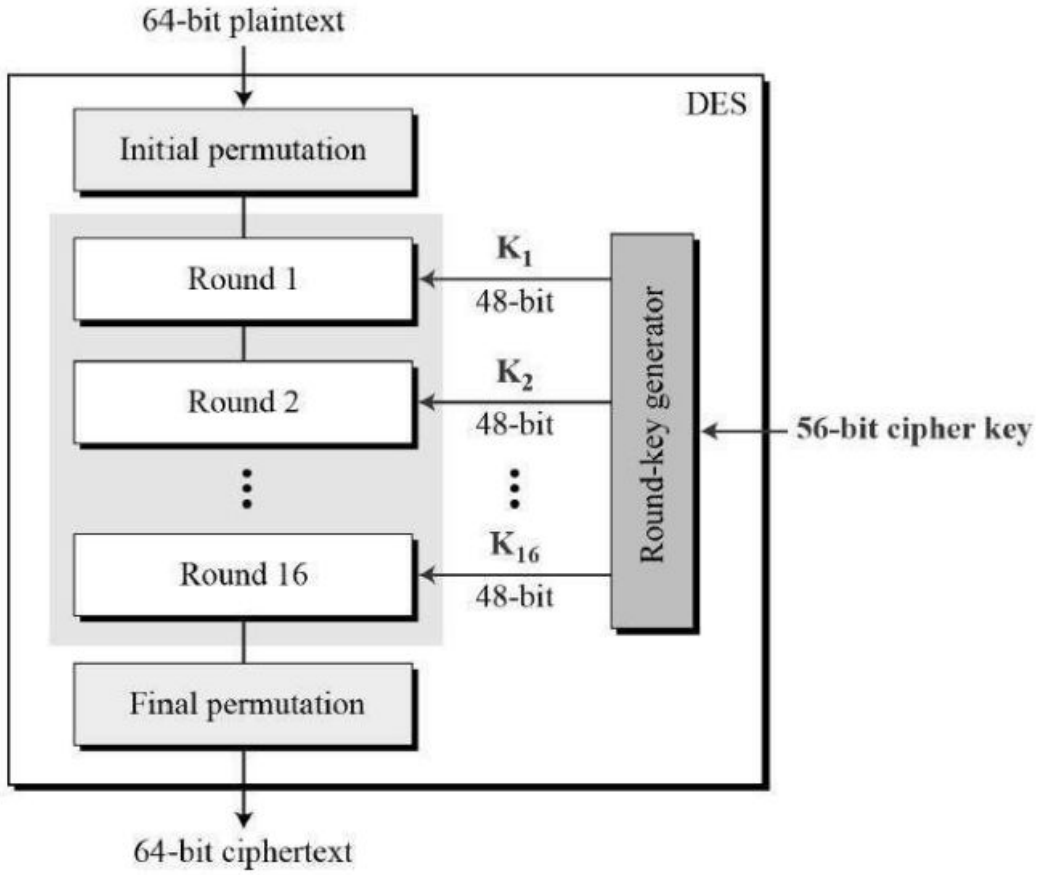
El nombre original del algoritmo, tal como lo denominó IBM, era Lucifer. Trabajaba sobre bloques de 128 bits, teniendo la clave igual longitud. Se basaba en operaciones lógicas booleanas y podía ser implementado fácilmente, tanto en software como en hardware.

Tras las modificaciones introducidas por el NBS, consistentes básicamente en la reducción de la longitud de clave y de los bloques, DES cifra bloques de 64 bits, mediante permutación y sustitución y usando una clave de 64 bits, de los que 8 son de paridad (esto es, en realidad usa 56 bits), produciendo así 64 bits cifrados.

Estructura básica

DES (data encryption standard) ha sido encontrado vulnerable contra cada ataque fuerte, debido a esto la popularidad de este método de encriptación se ha visto en declive. DES realiza un cifrado y descifrado por bloques, cada bloque tiene un tamaño de 64 bit. Lo que significa que el bloque de texto que entra es de 64 bit y el bloque de salida del cifrado también saldrá con el tamaño de 64. El mismo algoritmo y llave son utilizados para encriptar y desencriptar, con diferencias minúsculas.

DES es una implementación del cifrado de Feistel. Usa la estructura Feistel de 16 rondas. En la siguiente imagen podemos observar la estructura general:



Funcionamiento del algoritmo

Encriptación:

1. Se solicita una clave de 64 bits al usuario.
    a. De cada uno de los ocho bytes se elimina el octavo bit (el menos significativo).
2. Calcular las subclaves
    a. Realiza la siguiente permutación en la clave de 64 bits reduciéndose la misma a 56 bits (El bit 1, el más significativo, de la clave transformada es el bit 57 de la clave original, el bit 2 pasa a ser el bit 49.

Permuted Choice 1 (PC-1)

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

    b. Divide la clave permutada en dos mitades de 28 bits cada una. C(0) el bloque que contiene los 28 bits de mayor peso y D(0) los 28 bits restantes.

c. Calcular las 16 subclaves (Empezar con i=1) 1.2.3.1. Rotar uno o dos bits a la izquierda C(i-1) y D(i-1) para obtener C(i) y D(i), respectivamente
d. Concatenar C(i) y D(i) y permutar como se indica a continuación. Así se obtiene K(i), que tiene una longitud de 48 bits.

Permuted Choice 2 (PC-2)

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

3. Procesar el bloque de datos de 64 bits.
   a. Obtiene un bloque de datos de 64 bits. Si el bloque contiene menos de 64 bits debe ser completado para poder continuar con el siguiente paso.
   b. Realizar la siguiente permutación del bloque:

Initial Permutation (IP)

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

   i.
   c. Dividir el bloque resultante en dos mitades de 32 bits cada una. L(0) el bloque que contiene los 32 bits de mayor peso y R(0) el resto.
   d. Aplicar las 16 subclaves obtenidas . E(R(i)). Expandir R(i) de 32 a 48 bits

Expansion (E)

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

   i.
   ii. Se realiza E(R(i-1)) Xor K(i). Or-exclusiva del resultado del paso anterior.
   iii. A Partir de E(R(i-1)) Xor K(i) en ocho bloques de seis bits. B(1) representa a los bits 1-6, B(2) representa a los bits 7-12,..., B(8) representa a los bits 43- 48.
   iv. Realizar S(1)(B(1)), S(2)(B(2)),..., S(8)(B(8)). Sustituir todos los B(j) por los valores correspondientes de las S-Cajas o tablas de sustitución (Substitution Boxes, S-Boxes) de 6*4 bits, según se indica en los subapartados que siguen. Todos los valores de las S-Cajas se consideran de 4 bits de longitud

e. Realizar P[S(1)(B(1))... S(2)(B(8))]. Concatenar los bloques B(1) a B(8) y permutar los 32 bits (cuatro bits cada B(j)) en función de esta tabla.

i.

Permutation 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

4. Cajas DES

Fila	Columna															S-Caja
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S_1
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	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S_2
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	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	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
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	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S_4
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	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S_5
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	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S_6
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	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S_7
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	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S_8
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	

5. Realizar P[S(1)(B(1))... S(2)(B(8))]. Concatenar los bloques B(1) a B(8) y permutar los 32 bits (cuatro bits cada B(j)) en función de esta tabla:

a.

Permutation 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

b. R(i). Realizar una or-exclusiva entre el valor resultante y L(i-1). Este valor será R(i). Por tanto, R(i)=L(i-1) Xor P[S(1)(B(1))... S(2)(B(8))
c. Realizar la permutación final.

i.

Final Permutation (IP**-1)

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

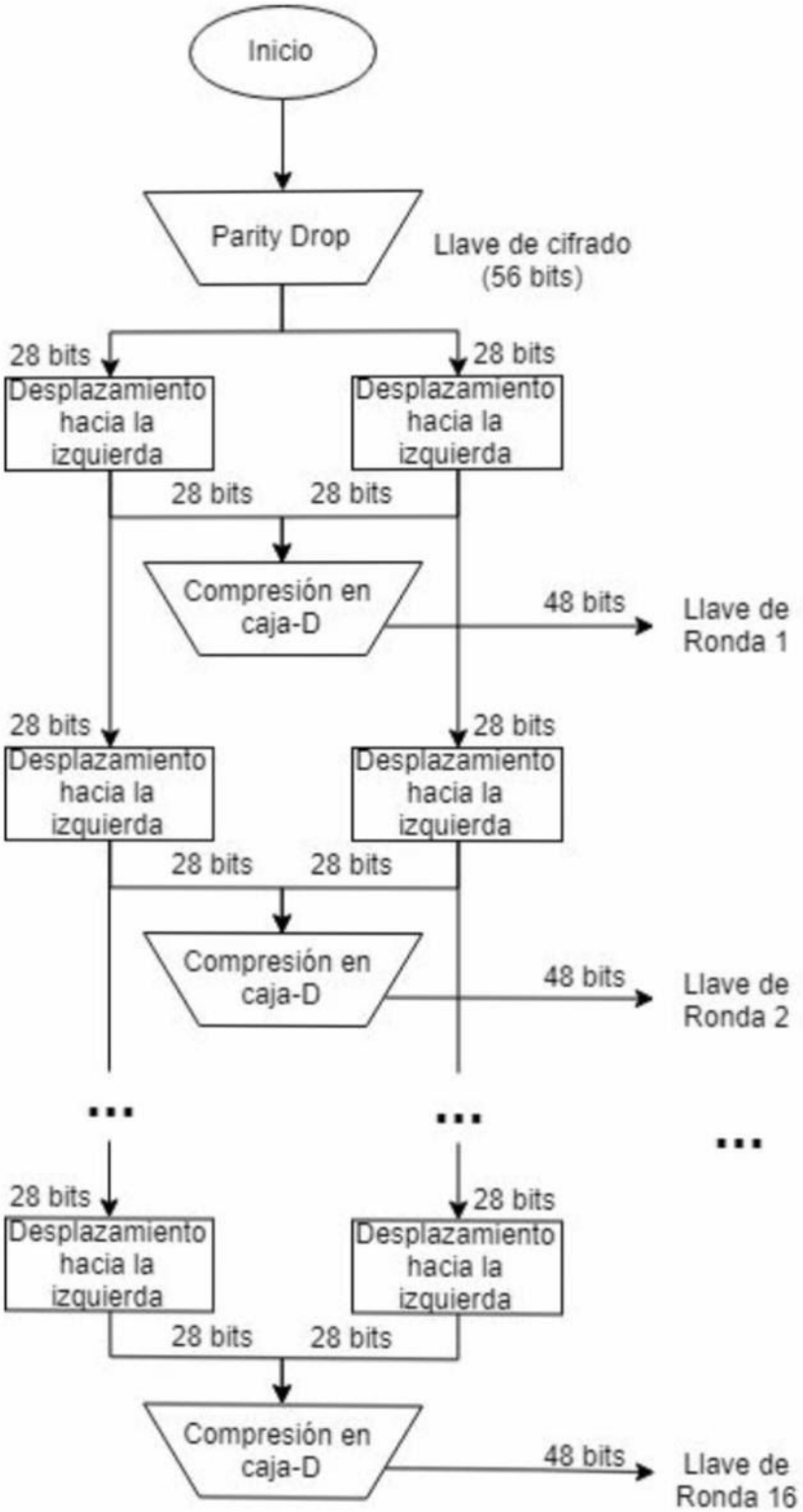
Desencriptación:

- Usar el mismo proceso descrito con anterioridad pero empleando las subclaves en orden inverso, esto es, en lugar de aplicar K(1) para la primera iteración aplicar K(16), K(15) para la segunda y así hasta K(1).

Modificación al DES básico para poder realizarlo mediante pthreads

Algoritmo narrativo (variante de DES):
1. DES-X
2. Se realizará un XOR para tener una llave extra de 64 bits al texto original antes de aplicarle DES.
3. Aplicar DES.
4. Luego realizar una segunda llave de 64 bits después de aplicar DES.

Diagrama del nuevo algoritmo



Subrutina y funciones usadas

Para ingresar los archivos de texto, los txt deben llamarse “encryptedText.txt” & “decryptedText.txt” respectivamente.

1. Convierte un número a base 10

```
int convertTo10(const string& input, int base){
    if(base < 2 || base > 36)
        return 0;

    bool isNegative = (input[0] == '-');

    int startIndex = input.length()-1;
    int endIndex   = isNegative ? 1 : 0;

    int val = 0;
    int digitVal = 1;

    for(int i = startIndex; i >= endIndex; --i)
    {
        char c = input[i];

        if(c >= 'a' && c <= 'z')
            c -= ('a' - 'A');

        if(c >= '0' && c <= '9')
            c -= '0';
        else
            c = c - 'A' + 10;

        if(c >= base)
            return 0;

        val += c * digitVal;

        digitVal *= base;
    }

    if(isNegative)
        val *= -1;

    return val;
}
```

2. Convierte texto a binario
    a. El parámetro int n recibe la “key”.

```
string toBin(int n){
    string r;
    while(n!=0) {r=(n%2==0 ?"0":"1")+r; n/=2;}
    while(r.size()<8){r = "0" + r;}
    return r;
}
```

3. Obtiene las rondas usando la llave
    a. Se pasa como parámetro la llave secreta

```
int getShift(string key){
    int shift = convertTo10(key, 36)%10;
    return shift;
}
```

4. Realiza la operación xor de las rondas

```
string xorStr(string str1, string str2){
    string xorstr = "";
    for(int x=0; x<str1.size();x++){
        xorstr += (str1[x]!=str2[x]?"1":"0");
    }
    return xorstr;
}
```

5. Struct de thread de los datos

```
struct thread_data {
    char key; //Encrypting key
    int id;  // Thread ID
    int shift; //shift for the encrypting offset
    char data; //Data vector
};
```

6. Función que guarda todos los resultados de los datos

```
void fillData(int index, char data){
    index = index%4;
    if(index == 0){
        res1 = data;
    } else if(index == 1){
        res2 = data;
    } else if(index == 2){
        res3 = data;
    } else if(index == 3){
        res4 = data;
    }
}
```

7. Función que hace el decrypt / encrypt respectivamente

```
void *decrypt(void *threadarg){
    struct thread_data *my_data = (struct thread_data *)threadarg;
    char key = my_data->key;
    char text = my_data->data;
    int thread_UID = my_data->id;
    int shift = my_data->shift;
    //Converting the key to binary
    string binKey = toBin(int(key));
    //Offset shift ascii mod 256
    string temp = toBin(int(text));
    //xor char from key to char from string
    temp = xorStr(temp, binKey);
    // Xored bin string back to ascii in dec
    text=(convertTo10(temp, 2)-shift)%256;
    fillData(thread_UID, text);


    }
```

Nivel de paralelismo(s) utilizado

1) Nivel de bit: Todos los computadores utilizan el paralelismo a nivel de bit.
2) Nivel de proceso: Se utilizó el paralelismo a nivel de proceso. En este paralelismo varios procesos componen una tarea. Son “bloques” definidos que funcionan de una manera específica. El número de hilos utilizados en el programa son 4.

3) El paralelismo a nivel de tarea también estuvo presente. Varias tareas independientes simultáneas, como la lectura de un texto y escritura de su contenido en varias variables, logran formar parte de un programa determinado. La interacción directa de estas tareas no estuvo presente. Así mismo, la lectura del contenido en una variable y la escritura en el texto de salida al mismo tiempo también es paralelismo a nivel de tarea.

Discusión

Para la realización de este proyecto, se tomó como base el algoritmo de encriptación/desencriptación DES. Tomando como base DES, se realizó un algoritmo propio donde se realizaron como mínimo 3 modificaciones distintas al algoritmo original. De igual manera, se utilizaron pthread con el objetivo de mejorar el rendimiento del programa.

Como puntos a mejorar, se puede considerar una realización más práctica de las funciones de encriptado y desencriptado, así mismo, un mejor uso y asignación de hilos para que el programa consiga un mejor rendimiento.

Conclusiones

1. DES utiliza también una clave criptográfica para modificar la transformación, de modo que el descifrado sólo puede ser realizado por aquellos que conozcan la clave concreta utilizada en el cifrado. La clave mide 64 bits, aunque en realidad, sólo 56 de ellos son empleados por el algoritmo.
2. Gracias al paralelismo empleado en el programa se logra agilizar la ejecución de instrucciones y funciones. Este procesamiento permite realizar varias tareas, como el ahorro en el tiempo de ejecución.
3. A pesar de que el algoritmo DES fue la base para la realización de este proyecto, se considera un nuevo algoritmo, más eficiente y práctico que el DES.

Referencias

- B. Schneier, Applied Chyptography. 2Ed. John Wiley & Sons, Inc. 1996 ebook (appliedcryptography-2nd-ed-b-schneier.pdf)
- B.Scheneir, “12.2 Description of DES” Applied Chyptography. 2Ed. John Wiley & Sons, Inc. 1996 (DESschneier.pdf)

- “Chapter 6 Date Encryption Standard (DES)” Hands on Experience Computer Security. Cleveland State University. Lecture notes. [online] https://academic.csuohio.edu/yuc/security/
- Subrutinas y funciones usadas. Extraído de: Code from: https://gist.github.com/JohannesMP/a911b7dc02bb0586e3111a0cbd2dc0e2*/