

MINISTERUL EDUCAȚIEI AL REPUBLICII MOLDOVA
UNIVERSITATEA TEHNICĂ A MOLDOVEI
FACULTATEA CALCULATOARE, INFORMATICĂ ȘI MICROELECTRONICĂ
DEPARTAMENTUL INGINERIE SOFTWARE ȘI AUTOMATICĂ

LUCRARE DE LABORATOR NR.1

Disciplina : *Securitatea Informațională*

Tema : *Algoritmul de criptare DES*

A elaborat : st.gr.TI-171 f/r , Florea Cristina

A verificat : lect.univ. Poștaru Andrei

Scopul lucrării .

Studierea , analiza metodologiei și implementarea algoritmului DES.

Algoritmul DES

Tehnica de criptare DES (Data Encryption Standard) a apărut în anii '70 în Statele Unite ale Americii. Standardul de criptare a fost realizat de Biroul National de Standardizare (National Bureau of Standards) cu ajutorul Agenției Naționale de Securitate (National Security Agency). Această tehnică de criptare este una dintre cele mai utilizate. Scopul acestei tehnici este acela de a asigura o metodă standard pentru protejarea datelor comerciale neclasificate. IBM a creat în anul 1976 o primă implementare a algoritmului sub numele de Lucifer.

Acest algoritm utilizează reprezentarea binară a codurilor numerice ale caracterelor folosite pentru reprezentarea datelor care urmează a fi criptate. Algoritmul de bază DES folosește blocuri de 64 de biți de date și aplică doar două operații asupra intrării: deplasare și substituție de biți. Cheia de criptare controlează procesul de codificare a datelor. Prin aplicarea celor două operații în mod repetat și neliniar, se obține un rezultat care, practic, nu poate fi decriptat decât dacă se cunoaște cheia de criptare.

Asupra fiecărui bloc de date, algoritmul se aplică de până la 16 ori (DES standard aplică algoritmul de criptare de 16 ori).

Datorită complexității ridicate a algoritmului DES standard, în continuare vă prezentăm varianta simplificată a algoritmului DES, Simple-DES, care folosește blocuri de 8 biți de date.

Să presupunem că avem cu text cu informații personale care dorim să-l ascundem de restul lumii. Cu ajutorul unui algoritm precum DES putem codifica textul dorit utilizând o parola cunoscută doar de noi . Rezultatul codificării va fi un text de aceeași mărime cu textul inițial dar cu un conținut indescifrabil pentru cititor .

De exemplu dacă avem textul : "**Propoziția mea secreta**" acesta va fi codificat în "**3642452842864283**" folosind cheia (parola) "**ParolaMeaSecreta**" . Este evident ca un necunoscut nu va înțelege nimic din rezultatul final pe când noi , folosind parola cunoscuta doar de noi , vom putea descifra textul "**3642452842864283**" înapoi în "**Propoziția mea secreta**" .

Înainte de-a intra în amănunte va trebui să prezentăm niște noțiuni introductive pentru a putea înțelege modul de funcționare al algoritmului .

DES lucrează cu biți adică cu 0 și 1. Orice text în memoria calculatorului este format dintr-o înșiruire de 0 și 1 iar DES modifică acele cifre folosind cheia în așa fel încât doar folosind cheia să se poată descifra ce e de fapt ascuns acolo .

Orice grup de 4 biți reprezintă un număr în baza 16 2^4 sau hexa . Cu alte cuvinte , folosind doar 4 biți vom putea reprezenta doar 16 numere si anume : 1=0001 , 2=0010 , 3=0011 ... 9=1001 , A=1010 , B=1011 , .. F=1111 .

Așadar oricare ar fi textul pe calculator el este reprezentat printr-un sir de biți care daca îi grupăm în câte 4 atunci textul va fi reprezentat printr-un sir de numere în baza 16 .

De exemplu :

$$\mathbf{0101\ 1101\ 1100\ 0101} = \mathbf{5DC5}$$

Cele trei șiruri de mai sus sunt identice doar reprezentate în mod diferit .

DES lucrează cu segmente de 64 de biți care sunt totuși cu segmente de 16 numere hexa . Aceste șiruri le vom împărți în 2 segmente egale de 32 de biți .

De ex. avem mesajul M pe 64 de biți care îl împărțim în două segmente S și D de 32 de biți :

$$\begin{array}{l} \mathbf{M = 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000\ 1001\ 1010\ 1011\ 1100\ 1101\ 1110\ 1111} \\ \mathbf{L = 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000\ 1001\ 1010\ 1011\ 1100\ 1101\ 1110\ 1111} = \mathbf{R} \end{array}$$

Toate segmentele de 64 de biți din textul care dorim să-l ascundem vor fi transformate tot în segmente de 64 biți în mod codat însă . Pentru cazul în care textul nu are un număr multiplu al lui 64 de biți , se vor adăuga la sfârșitul mesajului biți 0 până se va forma un segment final tot de 64 de biți .

DES utilizează chei pe 64 de biți dar folosește din aceștia doar 56 de biți din cauza că ignoră biții din 8 în 8 . Așadar biții 8 , 16 , 24 , 32 , 40 , 48 , 56 , 64 sunt ignorați . Pașii algoritmului DES sunt următorii :

1. **Generarea a 16 sub chei pe 48 biți**
2. **Criptarea fiecărui segment de 64 biți**

Generarea a 16 sub chei pe 48 biți

Avem cheia cunoscută pe 64 biți. Fiecare din biți va fi numerotat începând cu 1 până la 64. Se fa efectua o permutare pe 56 biți conform următorului tabel :

PC-1

5	4	4	3	2	1	9
7	9	1	3	5	7	
1	5	5	4	3	2	1
	8	0	2	4	6	8
1	2	5	5	4	3	2
0		9	1	3	5	7
1	11	3	6	5	4	3
9			0	2	4	6
6	5	4	3	3	2	1
3	5	7	9	1	3	5
7	6	5	4	3	3	2
	2	4	6	8	0	2
1	6	6	5	4	3	2
4		1	3	5	7	9
2	1	5	2	2	1	4
1	3		8	0	2	

K = 00010011 00110100 01010111 01111001 0011011 10111100 10111 1 11110001
K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

Împărțim acum chei **K+** în 2 jumătăți **S0** și **D0** de câte 28 biți fiecare . Pe poziția 1 se află bitul 57 , pe poziția 2 vom avea bitul 49 din cheie , ș.a.m.d. Pentru cheia **K** vom obține permutarea **K+**.

C1D1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110
K1 = 000110 110000 001011 101111 111111 000111 000001 110010

Având **S0** si **D0** calculate , vom efectua apoi 16 pași generând **S1 D1** , **S2 D2** .. **S16 D16** . Generarea unei perechi **S[i] D[i]** se face folosind valoarea **S[i-1] D[i-1]** si anume prin rotirea la stânga a valorilor din segment. Nr de rotiri este prezentat în tabela de mai jos:

Nr. de iterație	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Nr. de rotiri	1	1	2	2	2	2	2	2	1	1	1	1	1	1	1	2

Cu alte cuvinte pentru a obține S1 și D1 rotim o dată dreapta segmentele S0 și D0 , pentru S2 și D2 roti de 2 ori la dreapta segmentele S1 și D1 ș.a.m.d. De ex. pentru S0 și D0 de mai sus vom obține :

S0	D0
1111000011001100101010101111	0101010101100110011110001111
S1	D1
1111000011001100101010101111	0101010101100110011110001111
S2	D2

Cele 16 chei vor fi

S0 = 1111000011001100101010101111	D0 = 0101010101100110011110001111
S1 = 1110000110011001010101011111	D1 = 1010101011001100111100011110
S2 = 1100001100110010101010111111	D2 = 0101010110011001111000111101
S3 = 0000110011001010101011111111	D3 = 0101011001100111100011110101
S4 = 0011001100101010101111111100	D4 = 0101100110011110001111010101
S5 = 1100110010101010111111110000	D5 = 0110011001111000111101010101
S6 = 0011001010101011111111000011	D6 = 1001100111100011110101010101
S7 = 1100101010101111111100001100	D7 = 0110011110001111010101010110
S8 = 0010101010111111110000110011	D8 = 1001111000111101010101011001
S9 = 0101010101111111100001100110	D9 = 0011110001111010101010110011
S10 = 0101010111111110000110011001	D10 = 1111000111101010101011001100
S11 = 0101011111111000011001100101	D11 = 1100011110101010101100110011
S12 = 0101111111100001100110010101	D12 = 0001111010101010110011001111
S13 = 0111111110000110011001010101	D13 = 0111101010101011001100111100
S14 = 1111111000011001100101010101	D14 = 1110101010101100110011110001
S15 = 1111100001100110010101010111	D15 = 1010101010110011001111000111

S16 = 1111000011001100101010101111

D16 = 0101010101100110011110001111

În acest moment avem n perechi SiDi , i=1 .. 16 : S1D1 , S1D2 , ... S16D16. În continuare se va construi cheile Ki, i=1 .. 16, fiecare Ki fiind formată din permutarea dubletului SiDi conform următorului table :

PC-2

1	1	11	2	1	5
4	7		4		
3	2	1	6	2	1
	8	5		1	0
2	1	1	4	2	8
3	9	2		6	
1	7	2	2	1	2
6		7	0	3	
4	5	3	3	4	5
1	2	1	7	7	5
3	4	5	4	3	4
0	0	1	5	3	8
4	4	3	5	3	5
4	9	9	6	4	3
4	4	5	3	2	3
6	2	0	6	9	2

Deci pentru cheia Ki bitul 1 va fi bitul 14 din perechea SiDi , bitul 2 va fi bitul 17 din perechea SiDi ș.a.m.d. De exemplu pentru S1D1 vom avea K1 ca in desenul de mai jos :

C1D1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

K1 = 000110 110000 001011 101111 111111 000111 000001 110010

Avem în acest moment cele 16 sub chei care trebuiau calculate și anume K1 din desen și

K2 = 011110 011010 111011 011001 110110 111100 100111 100101
K3 = 010101 011111 110010 001010 010000 101100 111110 011001
K4 = 011100 101010 110111 010110 110110 110011 010100 011101
K5 = 011111 001110 110000 000111 111010 110101 001110 101000
K6 = 011000 111010 010100 111110 010100 000111 101100 101111
K7 = 111011 001000 010010 110111 111101 100001 100010 111100
K8 = 111101 111000 101000 111010 110000 010011 101111 111011
K9 = 111000 001101 101111 101011 111011 011110 011110 000001
K10 = 101100 011111 001101 000111 101110 100100 011001 001111
K11 = 001000 010101 111111 111111 110111 101101 001110 000110
K12 = 011101 010111 000111 110101 100101 000110 011111 101001
K13 = 100101 111100 010111 010001 111110 101011 101001 000001
K14 = 010111 110100 001110 110111 111100 101110 011100 111010
K15 = 101111 111001 000110 001101 001111 010011 111100 001010
K16 = 110010 110011 110110 001011 000011 100001 011111 110101

2. Criptarea fiecărui segment de 64 biți

Fiecare segment de 64 biți urmează să fie codat într-un segment tot de 64 biți folosind sub cheile generate mai sus. În continuare sunt pașii efectuați pentru toate segmentele de 64 biți .

Fie dat segmentul M , primul segment din text care urmează a fi criptat.

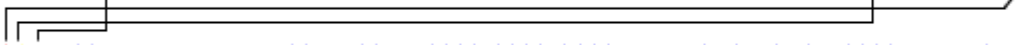
M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

IP

5	5	4	3	2	1	1	2
---	---	---	---	---	---	---	---

8	0	2	4	6	8	0	
6	5	4	3	2	2	1	4
0	2	4	6	8	0	2	
6	5	4	3	3	2	1	6
2	4	6	8	0	2	4	
6	5	4	4	3	2	1	8
4	6	8	0	2	4	6	
5	4	4	3	2	1	9	1
7	9	1	3	5	7		
5	5	4	3	2	1	11	3
9	1	3	5	7	9		
6	5	4	3	2	2	1	5
1	3	5	7	9	1	3	
6	5	4	3	3	2	1	7
3	5	7	9	1	3	5	

De exemplu , pentru M de mai sus va rezulta segmentul IP de mai jos

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

Segmentul IP rezultat îl vom împărți apoi in doua segmente egale de 32 de biți L0 si R0

L0 = 1100 1100 0000 0000 1100 1100 1111 1111
R0 = 1111 0000 1010 1010 1111 0000 1010 1010

În cele ce urmează vom folosi semnul + pentru adunarea in baza 2 ($0+0=0$, $0+1=1+0=1$, $1+1=0$).Vom efectua 16 pași pornind de la L0 si R0 după cum urmează :

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

când $n=1..16$ si L0 respectiv R0 sunt cunoscute . Din relația de recurenta mai trebuie doar stabilit ce întoarce funcția f . Evident ca va întoarce u segment de 32 de biți care va fi apoi adunat la L_{n-1} .Rezultatul final va fi o pereche L16 si R16 .

Pentru a putea explica modul de funcționare a funcției f trebuie întâi definita funcția E .

E primește un segment pe 32 de biți , in cazul nostru R_{n-1} si întoarce un segment de 48 de biți folosind tabela de extindere :

E BIT-SELECTION TABLE

3	1	2	3	4	5
2					
4	5	6	7	8	9
8	9	1	11	1	1
		0		2	3
1	1	1	1	1	1
2	3	4	5	6	7
1	1	1	1	2	2
6	7	8	9	0	1
2	2	2	2	2	2
0	1	2	3	4	5
2	2	2	2	2	2
4	5	6	7	8	9
2	2	3	3	3	1
8	9	0	1	2	

Adică pe prima poziție a segmentului de 48 de biți rezultat va fi bitul 32 din segmentul R_{n-1} , pe a doua va fi bitul 1 din R_{n-1} ș.a.m.d. De exemplu pentru R_0 de mai sus vom avea :

$R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$

$E(R_0) = 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$

Având acum valoarea lui $E(R_n)$, adunam la aceasta (în baza 2) cheia In (sub cheia calculată la pasul 1). În urma adunării avem un segment de 48 de biți care va trebui să-l transformăm înapoi într-un segment de 32 de biți. Pentru acest lucru împărțim segmentul de 48 de biți în segmente de 6 biți notate cu $B_1, B_2 \dots B_8$:

$K_n + E(R_{n-1}) = B_1\ B_2\ B_3\ B_4\ B_5\ B_6\ B_7\ B_8$

Pentru a restrânge la 4 biți fiecare segment $B_1, B_2 \dots$ și B_8 vom aplica funcțiile $S_1, S_2 \dots S_8$ astfel încât rezultatul va fi :

$S_1(B_1)\ S_2(B_2)\ S_3(B_3)\ S_4(B_4)\ S_5(B_5)\ S_6(B_6)\ S_7(B_7)\ S_8(B_8)$

Rolul acestor funcții S_n este de a lua un segment de 6 biți și a-l transforma în unul de 4 biți astfel încât rezultatul să fie un segment mare de 32 de biți.

Iată cum funcționează funcția S_1 : pentru segmentul $B_1 = b_1b_2b_3b_4b_5b_6$, am notat cu $b_1 \dots b_6$ biții lui B_1 . Primul bit și ultimul din B_1 , adică b_1 și b_6 se lipsesc în b_1b_6 formând un număr pe 2 biți care în baza 10 poate fi 0,1,2 sau 3. Biții b_2, b_3, b_4 , și b_5 formează la rândul lor un număr care în baza zece poate fi 0,1 .. 14 sau 15. Numărul b_1b_6 reprezintă numărul liniei iar numărul $b_2b_3b_4b_5$ reprezintă numărul coloanei în tabelul funcției S_1 care este :

S1

	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
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

De exemplu pentru segmentul B=011011 vom avea linia 01 = 1 si coloana 1101 = 13 , aşadar coordonatele 1,13 in tabelul S1 .

Rezultatul funcţiei S1 va fi aşadar un număr de la 0 la 16 care este reprezentat pe 4 biţi , deci , 4 biţi.

Funcţiile S2 , .. S8 se folosesc la fel având doar tabele diferite . Iată aceste tabele :

S2

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

S3

10	0	9	1	6	3	15	5	1	1	1	7	11	4	2	8
			4						3	2					
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	1	14	7
													0		
1	10	1	0	6	9	8	7	4	1	1	3	11	5	2	1
		3							5	4					2

S4

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

S5

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

S6

12	1	1	1	9	2	6	8	0	1	3	4	14	7	5	11
		0	5						3						
10	15	4	2	7	12	9	5	6	1	1	14	0	11	3	8
										3					
9	14	1	5	2	8	12	3	7	0	4	10	1	1	11	6
		5											3		
4	3	2	1	9	5	15	10	11	1	1	7	6	0	8	1
			2						4						3

S7

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

S8

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

		4							2						
--	--	---	--	--	--	--	--	--	---	--	--	--	--	--	--

De exemplu pentru

K1 + E(R0) = 011000 010001 011110 111010 100001 100110 010100 100111.

avem

**S1(B1)S2(B2)S3(B3)S4(B4)S5(B5)S6(B6)S7(B7)S8(B8)=
01011100100000101011010110010111**

Pasul final pentru obținerea valorii lui f este efectuarea unei permutări asupra șirului rezultat în urma aplicării funcțiilor S1 .. S8 , permutare conform tabelul :

P			
1	7	2	2
6		0	1
2	1	2	1
9	2	8	7
1	1	2	2
	5	3	6
5	1	3	1
	8	1	0
2	8	3	1
		1	0
3	2	3	9
2	7		
1	1	3	6
9	3	0	
2	11	4	2
2			5

Spre exemplu pentru

**S1(B1)S2(B2)S3(B3)S4(B4)S5(B5)S6(B6)S7(B7)S8(B8) =
0101 1100 1000 0010 1011 0101 1001 0111**

vom obține valoare finală a lui f : **f = 0010 0011 0100 1010 1010 1001 1011 1011**

Rezultatul celor 16 pași conform

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

vor fi doua segmente L16 si R16 . Cele doua segmente se lipesc apoi invers formând un segment de 64biti R16L16 . Asupra acestui segment se efectuează o permutare finala conform tabelul următor :

IP-1

4	8	4	1	5	2	6	3
0		8	6	6	4	4	2
3	7	4	1	5	2	6	3
9		7	5	5	3	3	1
3	6	4	1	5	2	6	3
8		6	4	4	2	2	0
3	5	4	1	5	2	6	2
7		5	3	3	1	1	9
3	4	4	1	5	2	6	2
6		4	2	2	0	0	8
3	3	4	11	5	1	5	2
5		3		1	9	9	7
3	2	4	1	5	1	5	2
4		2	0	0	8	8	6
3	1	4	7	4	1	5	2
3		1		9	7	7	5

Rezultatul permutării va fi blocul de 64 de biți criptat .

De exemplu pentru L16 si R16 obținute

L16 = 0100 0011 0100 0010 0011 0010 0011 0100

R16 = 0000 1010 0100 1100 1101 1001 1001 0101

segmentul inversat va fi

R16L16 = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100

iar după permutare

IP-1 = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

care in baza 16 este

85E813540F0AB405 .

Aplicând pentru toate segmentele de 64 de biți pasul 2 descris mai sus se obține textul criptat .

Operația de decriptare se face similar inversând ordinea operațiilor .

Concluzie .

La realizarea acestei lucrări de laborator , au fost studiați pașii de lucru al algoritmului DES. Acesta este un algoritm simetric , utilizat și în ziua de astăzi , doar că din punct de vedere a securității informaționale acest algoritm are un cod mai ușor de decriptat și deseori companiile folosesc tehnici de criptare mai contemporane.

Codul sursă .

```
using System;
using System.Text;
using System.IO;
using System.Collections;

namespace Cipher
{
    class DES
    {
        private byte[] IP, IP1, P, E, PC1, PC2, LeftShifts;
        private byte[,] Ss;

        protected string Key = null;
        private BitArray[] Keys;

        public DES()
        {
            InitializeTables();
        }

        protected BitArray Encrypt64Bit(BitArray block)
        {
            block = Table(IP, block);

            BitArray Left = new BitArray(32),
                Right = new BitArray(32),
                Temp;

            int tmp = block.Length / 2;
            Copy(block, 0, Left, 0, tmp); //Left=block[0-31];
            Copy(block, tmp, Right, 0, tmp); //Right=block[32-63];

            for (int i = 1; i <= 16; i++)
            {
                Temp = Left;
                Left = Right;
                Right = Temp.Xor(F(Right, Keys[i - 1]));
            }

            Copy(Right, 0, block, 0, 32); //block[0-31]=Right;
            Copy(Left, 0, block, 32, 32); //block[32-63]=Left;

            block = Table(IP1, block);

            return block;
        }

        protected BitArray Decrypt64Bit(BitArray block)
        {
            block = Table(IP, block);

            BitArray Left = new BitArray(32),
                Right = new BitArray(32),
                Temp;

            int tmp = block.Length / 2;
            Copy(block, 0, Left, 0, tmp); //Left=block[0-31];
            Copy(block, tmp, Right, 0, tmp); //Right=block[32-63];

            for (int i = 1; i <= 16; i++)
```



```

    {
        Temp = Left;
        Left = Right;
        Right = Temp.Xor(F(Right, Keys[16 - i]));
    }

    Copy(Right, 0, block, 0, 32); //block[0-31]=Right;
    Copy(Left, 0, block, 32, 32); //block[32-63]=Left;

    block = Table(IP1, block);

    return block;
}

private BitArray F(BitArray R, BitArray K)
{
    R = Table(E, R);
    BitArray B = R.Xor(K);
    BitArray S = new BitArray(8 * 4);

    int x, y;
    BitArray Temp;
    for (int i = 0; i < 8; i++)
    {
        x = (B[i * 6 + 0] ? 2 : 0) + (B[i * 6 + 5] ? 1 : 0);
        y = (B[i * 6 + 1] ? 8 : 0) + (B[i * 6 + 2] ? 4 : 0) +
            (B[i * 6 + 3] ? 2 : 0) + (B[i * 6 + 4] ? 1 : 0);

        Temp = new BitArray(new byte[] { Ss[i, 16 * x + y] });
        Copy(Temp, 0, S, i * 4, 4);
    }

    S = Table(P, S);
    return S;
}

protected void PermuteKeys(string Key)
{
    if (Key.Length != 8)
    {
        throw new Exception("Key Size Must Be 8 Characters Long..");
    }

    this.Key = Key;
    BitArray key = new BitArray(Encoding.ASCII.GetBytes(Key));
    key = Table(PC1, key);

    BitArray C = new BitArray(28),
        C_Old = new BitArray(28),
        D = new BitArray(28),
        D_Old = new BitArray(28);

    Copy(key, 0, C, 0 * 28, 28);
    Copy(key, 28, D, 0 * 28, 28);

    Keys = new BitArray[16]; //16 keys, each one of which has 48 bits

    BitArray Temp = new BitArray(28 * 2), TempKey;

    //generate 16 keys
    for (int i = 1; i <= 16; i++)
    {
        C_Old = C;
        D_Old = D;
        //left shifts

```

```

        for (int j = 0; j < 28; j++)
        {
            C[j] = C_Old[(j + LeftShifts[i - 1]) % 28];
            D[j] = D_Old[(j + LeftShifts[i - 1]) % 28];
        }

        Copy(C, 0, Temp, 0, 28);
        Copy(D, 0, Temp, 28, 28);

        TempKey = Table(PC2, Temp);

        //save the key
        Keys[i - 1] = new BitArray(8 * 6);
        Copy(TempKey, 0, Keys[i - 1], 0, 48);
    }
}

private BitArray Table(byte[] Table, BitArray block)
{
    int tmp = Table.Length;
    BitArray Result = new BitArray(tmp);
    for (int i = 0; i < tmp; i++)
    {
        Result[i] = block[Table[i] - 1];
    }
    return Result;
}

private void Copy(BitArray sourceArray, int sourceIndex,
    BitArray destinationArray, int destinationIndex, int lenght)
{
    for (int i = 0; i < lenght; i++)
    {
        if (sourceIndex + i < sourceArray.Length)
        {
            destinationArray[destinationIndex + i] =
sourceArray[sourceIndex + i];
        }
        else
        {
            destinationArray[destinationIndex + i] = false;
        }
    }
}

protected void InitializeTables()
{
    IP = new byte[8 * 8]{
2,      58,    50,    42,    34,    26,    18,    10,
4,              60,    52,    44,    36,    28,    20,    12,
6,              62,    54,    46,    38,    30,    22,    14,
8,              64,    56,    48,    40,    32,    24,    16,
1,        57,    49,    41,    33,    25,    17,     9,
3,        59,    51,    43,    35,    27,    19,    11,
5,        61,    53,    45,    37,    29,    21,    13,
7,        63,    55,    47,    39,    31,    23,    15,

    };
}

```

```

E = new byte[8 * 6] {
    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
};

P = new byte[8 * 4]{
    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
};

IP1 = new byte[8 * 8] {
32,    40,    8,   48,   16,   56,   24,   64,
31,    39,    7,   47,   15,   55,   23,   63,
30,    38,    6,   46,   14,   54,   22,   62,
29,    37,    5,   45,   13,   53,   21,   61,
28,    36,    4,   44,   12,   52,   20,   60,
27,    35,    3,   43,   11,   51,   19,   59,
26,    34,    2,   42,   10,   50,   18,   58,
25,    33,    1,   41,    9,   49,   17,   57,
};

PC1 = new byte[8 * 7]{
    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
};

PC2 = new byte[8 * 6] {
    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
};

LeftShifts = new byte[16] { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2,
2, 1 };

```

```

Ss = new byte[8, 4 * 16]
{
    {
        14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,
9, 0,  7,
        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,
        3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,
9,11,  5,
        0, 14,  7, 11, 10,  4, 13,  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,
        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,
        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,
        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
    } ,

    {

```

```

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

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

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

```