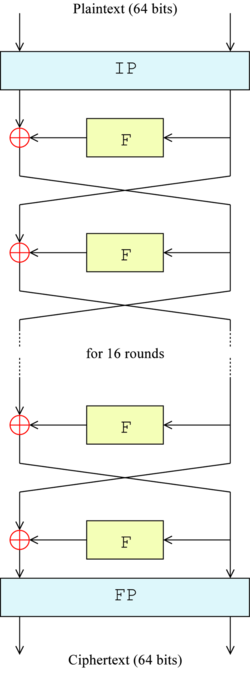
**Des Encryption**

در الگوریتم des متن ورودی به قطعات ۶۴ بیتی تبدیل می شود. کلید نیز ۶۴ بیت است ولی فقط از ۵۶ بیتش استفاده میشه و از ۸ بیت باقیش برای چک کردن parity استفاده میکنند.

مثال زیر را در نظر بگیرید:

Key = 13 34 57 79 9B BC DF F1

Input data = 01 23 45 67 89 AB CD EF

الگوریتم DES شامل ۱۶ مرحله است. متنی که قرار رمز شود ابتدا باید جایگشت مطابق بردار زیر روش انجام بشه. یعنی بیت های آن جا به جا شود.

PI = [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,

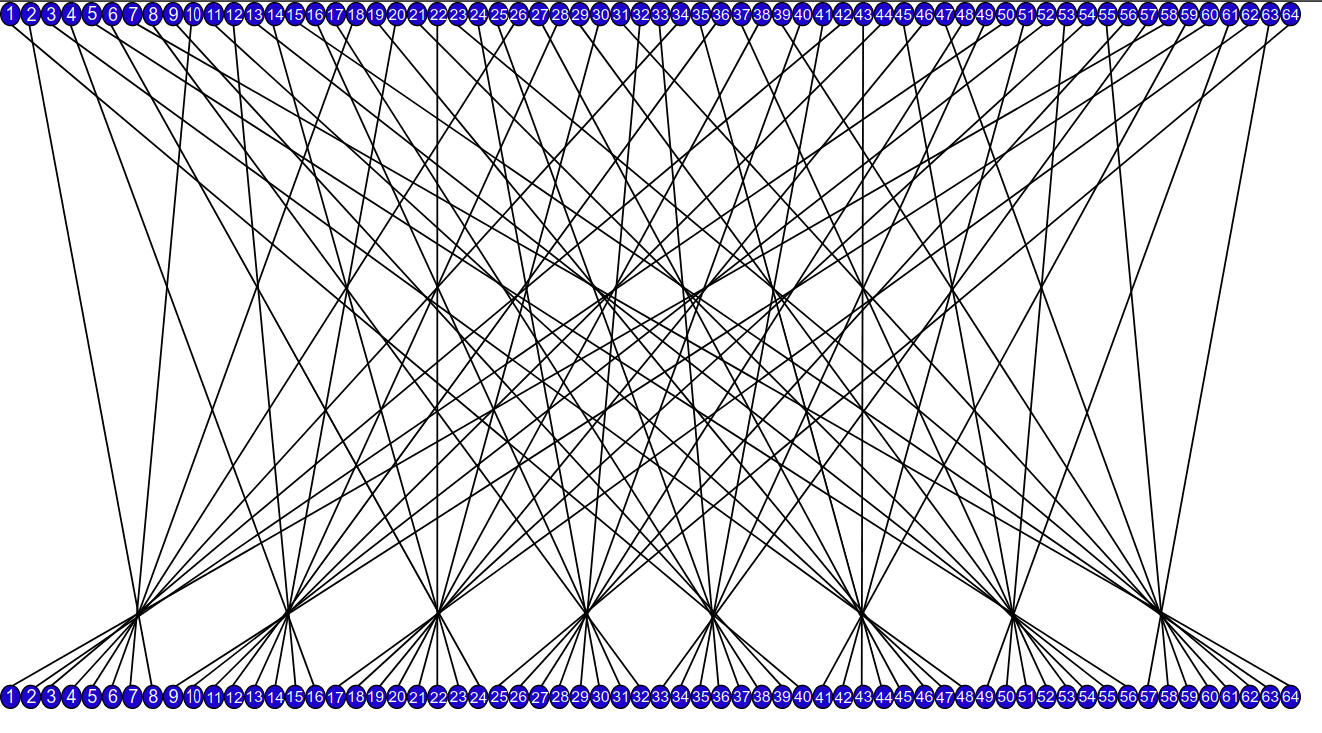
57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7]

در تصوی رزیر همانطور که مشخص است برای ساخت این بردار اولین بیت خروجی معادل ۵۸ امین بیت ورودی است و دومین بیت خروجی معادل پنجاهمین بیت ورودی است و به همین ترتیب.



ابتدا متن را به معادل دودویی آن تبدیل میکنیم:

def string\_to\_bit\_array(text):

array = list()

for char in text:

binval = binvalue(char, 8)

array.extend([int(x) for x in list(binval)])

return array

با اجرای کد بالا متن ورودی به معادل دودویی آن که در زیر آورده شده تبدیل می شود.

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

سپس مقدار دودویی را به دو نیمه چپ و راست که هر کدام ۳۲ بیت است تقسیم میکنیم:

def nsplit(s, n):

return [s[k:k+n] for k in range(0, len(s), n)]

تابع بالا به منظور تبدیل یک مقدار دودویی به بخش هایی با طول بیت مساوی استفاده میشود.

با اجرای کد:

g, d = nsplit(block, 32)

دو نیمه چپ و راست تولید می شود. که خروجی زیر را تولید میکند:

L = 0000 0001 0010 0011 0100 0101 0110 0111

R = 1000 1001 1010 1011 1100 1101 1110 1111

همانطور که توضیح دادیم الگوریتم DES شامل ۱۶ مرحله است که در هر مرحله از یک کلید که از کلید اصلی بدست آمده استفاده میکنیم. کلید اصلی الگوریتم des ۶۴ بیتی است ولی فقط از ۵۶ بیتش استفاده میشه و بیتهای مضرب ۸ شامل ۸ و ۱۶ و ۲۴ و ۳۲ و ۴۰ و ۴۸ و ۵۶ و ۶۴ برای parity استفاده میشود.

مراحل تولید کلید:

مرحله اول:

نیاز است که ۱۶ تا زیر کلید هر کدام به طول ۴۸ بیت تولید بشه.

ابتدا جدول Permuted Choice1 رو شرح میدهیم که به مخفف PC-1 آورده میشود.

CP\_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]

اولین عدد جدول فوق همانطور که ملاحظه می نمایید 57 است. یعنی اینکه 57 امین بیت کلید اصلی به اولین بیت کلید جای گردانی شده (که به آن +K می گوییم) تبدیل می شود. سپس 49 امین بیت کلید اصلی به دومین بیت کلید جای گردانی شده تبدیل می شود و همین طور الی آخر. بنابراین با توجه به اعداد مثال فوق داریم :

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

K--->(PC-1)--->K+

K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

مانند مثال بالا بعد از اعمال جدول CP\_1 به کلید اصلی عبارت K+ تولید می شود.

مرحله دوم

در این مرحله K+ رو هم به دو نیمه ۲۸ بیتی تقسیم میکنیم:

C0 = 1111000 0110011 0010101 0101111

D0 = 0101010 1011001 1001111 0001111

خب همانطور که گفتیم نیازه که ۱۶ کلید تولید بشه. این ۱۶ تا کلید را از C0,D0 تولید می کنیم. برای تولید این ۱۶ تا زیر کلید از جدول زیر استفاده می کنیم:

|  |  |
| --- | --- |
| **Number of Rotate left shift** | **#** |
| 1 | 1 |
| 1 | 2 |
| 2 | 3 |
| 2 | 4 |
| 2 | 5 |
| 2 | 6 |
| 2 | 7 |
| 2 | 8 |
| 1 | 9 |
| 2 | 10 |
| 2 | 11 |
| 2 | 12 |
| 2 | 13 |
| 2 | 14 |
| 2 | 15 |
| 1 | 16 |

جدول بالا به ما میگه که در مرحله اول تولید کلید عبارات C0,D0 یک بار به سمت چپ شیفت می شوند. در مرحله دوم نیز یک بار و تا ال آخر. که خروجی زیر تولید میشود:

C0 = 1111000011001100101010101111

D0 = 0101010101100110011110001111

C1 = 1110000110011001010101011111

D1 = 1010101011001100111100011110

C2 = 1100001100110010101010111111

D2 = 0101010110011001111000111101

C3 = 0000110011001010101011111111

D3 = 0101011001100111100011110101

C4 = 0011001100101010101111111100

D4 = 0101100110011110001111010101

C5 = 1100110010101010111111110000

D5 = 0110011001111000111101010101

C6 = 0011001010101011111111000011

D6 = 1001100111100011110101010101

C7 = 1100101010101111111100001100

D7 = 0110011110001111010101010110

C8 = 0010101010111111110000110011

D8 = 1001111000111101010101011001

C9 = 0101010101111111100001100110

D9 = 0011110001111010101010110011

C10 = 0101010111111110000110011001

D10 = 1111000111101010101011001100

C11 = 0101011111111000011001100101

D11 = 1100011110101010101100110011

C12 = 0101111111100001100110010101

D12 = 0001111010101010110011001111

C13 = 0111111110000110011001010101

D13 = 0111101010101011001100111100

C14 = 1111111000011001100101010101

D14 = 1110101010101100110011110001

C15 = 1111100001100110010101010111

D15 = 1010101010110011001111000111

C16 = 1111000011001100101010101111

D16 = 0101010101100110011110001111

حالا نوبت این است که جدول جایگذاری CP\_2 رو بر روی جفت کلیدهای بالا اعمال کنیم. جدول CP\_2 بردار زیر است:

CP\_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]

برای مثال بعد از اعمال جدول بالا بر روی C1,D1 عبارت زیر حاصل می شود:

C1D1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

CP\_2 باید بر روي تک تک 16 زیر کلید محاسبه شده به صورت مستقل اعمال شود .یعنی براي مثال در K1) با توجه

به جدول فوق) اولین بیت، معادل 14 امین بیت C1D1 است، دومین بیت K1 معادل 17 امین بیت C1D1 است و الی آخر.

در نهایت پس از اعمال 2-PC بر روي CnDn هاي محاسبه شده فوق داریم:

K1 = 000110 110000 001011 101111 111111 000111 000001 110010

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 010011 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

اگر بخواهیم تا اینجا قسمت تهیه subkey ها را خلاصه کنیم به صورت زیر می شود:

C[0]D[0] = PC1(key)

for 1 <= i <= 16

C[i] = LS[i](C[i-1])

D[i] = LS[i](D[i-1])

K[i] = PC2(C[i]D[i])

ازتابع زیر به منظور تولید کلیدها استفاده میشود:

def generatekeys(self):#الگوریتم تولید کلید

self.keys = []

key = string\_to\_bit\_array(self.password)

key = self.permut(key, CP\_1) #اعمال جایگشت اولیه بر روی کلید

g, d = nsplit(key, 28) # تقسیم کلید به دو بخش ۲۸ بیتی

for i in range(16):# اعمال ۱۶ راند بر روی کلید

g, d = self.shift(g, d, SHIFT[i]) # شیفت به چپ کلیدها

tmp = g + d # ادغام کلیدها

self.keys.append(self.permut(tmp, CP\_2))

حالا بعد از تولید کلیدها نیاز است که متن ورودی که به قطعات ۶۴ بیتی تقسیم شده است کدگذاری شود. جدول زیر را در نظر بگیرید:

PI = [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]

جدول IP در اینجا به معناي permutation initial است (جایگردانی اولیه) و بر روي پیغام اولیه ما یعنی M ،اعمال می

شود:

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

همانند روش هاي اعمال جداول قبل، 58 امین بیت M که در اینجا "1 " است اولین بیت IP می شود، سپس 50 امین بیت

M که در اینجا "1 " است ، دومین بیت IP می گردد و الی آخر.

سپس IP به دو نیمه ي 32 بیتی چپ و راست تقسیم می شود :

L0 = 1100 1100 0000 0000 1100 1100 1111 1111

R0 = 1111 0000 1010 1010 1111 0000 1010 1010

فرمول کلی این 16 راند به صورت زیر است :

L[n] = R[n-1]

R[n] = L[n-1] XOR f(R[n-1],K[n])

که در اینجا n بین ۱ و ۱۶ تغییر می کند. در مورد تابع f در ادامه توضیح داده خواهد شد.

براي مثال براي 1=n داریم :

L[0] = 1100 1100 0000 0000 1100 1100 1111 1111

R[0] = 1111 0000 1010 1010 1111 0000 1010 1010

K[1] = 000110 110000 001011 101111 111111 000111 000001 110010

سپس:

L[1] = R[0] = 1111 0000 1010 1010 1111 0000 1010 1010

R[1] = L[0] + f(R[0],K[1])

براي محاسبه f ابتدا باید [1-n[R سی و دو بیتی به 48 بیت بسط داده شود. براي انجام این کار از یک جدول انتخاب selection table به نام E استفاده می شود.

E(R[n-1])

تابع فوق ورودي اش 32 بیتی بوده و خروجی اش 48 بیتی است. این جدول به صورت زیر است:

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]

در ادامه قسمت قبل فرض کنید میخواهیم E(R[0]) را از [0[R محاسبه نماییم:

R[0] = 1111 0000 1010 1010 1111 0000 1010 1010

E(R[0]) = 011110 100001 010101 010101 011110 100001 010101 010101

همانطور که ملاحظه می کنید با اعمال جدول E هر 4 بیت اولیه به 6 بیت بسط داده شده است.

در ادامه ي محاسبه f خروجی ([1-n[R(E با کلید [n[K ، XOR می شود، برای مثال :

K[1] = 000110 110000 001011 101111 111111 000111 000001 110010

E(R[0]) = 011110 100001 010101 010101 011110 100001 010101 010101

K1 XOR E(R[0]) = 011000 010001 011110 111010 100001 100110 010100 100111

هنوز محاسبه ي f تمام نشده است. تا اینجا ([n[R(E از 32 بیت به 48 بیت با استفاده از جدول E ،بسط یافته است. سپس

حاصل آن با [n[K، XOR شده است. اکنون 48 بیت و یا هشت گروه 6 بیتی حاصل کار است. به این 8 گروه نامهایی از

[1[B تا [8[B نسبت می دهیم، یعنی :

K[n] XOR E(R[n]-1) =B[1] B[2] B[3] B[4] B[5] B[6] B[7] B[8]

در ادامه نوبت عملیاتی دیگر بر روي این [i[B ھا است. در اینجا مفهومی دیگر به نام SBoxes ارائه می شود .این جداول

باید بر روي [i[B ها اعمال می گردند. یعنی

S1(B[1]) S2(B[2]) S3(B[3]) S4(B[4]) S5(B[5]) S6(B[6]) S7(B[7]) S8(B[8])

این جداول به شرح زیر هستند:

S\_BOX = [

[[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],

],

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

[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],

[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],

[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],

]

]

مرحله پنجم

نحوه ي استفاده از S-boxes و یا boxes Substitution :

فرض کنید عدد 48 بیتی زیر را داریم که می خواهیم Boxes-S را بر آن اعمال کنیم :

011101000101110101000111101000011100101101011101

همانطور که در قسمت قبل گفته شد ، 8 گروه 6 بیتی از آن استخراج می شود که از B1 تا B8 را تشکیل می دهند:

011101 000101 110101 000111 101000 011100 101101 011101

محاسبات زیر را در نظر بگیرید:

B[1] => S[1](01, 1110) = S[1][1][14] = 3 = 0011

B[2] => S[2](01, 0010) = S[2][1][2 ] = 4 = 0100

B[3] => S[3](11, 1010) = S[3][3][10] = 14 = 1110

B[4] => S[4](01, 0011) = S[4][1][3 ] = 5 = 0101

B[5] => S[5](10, 0100) = S[5][2][4 ] = 10 = 1010

B[6] => S[6](00, 1110) = S[6][0][14] = 5 = 0101

B[7] => S[7](11, 0110) = S[7][3][6 ] = 10 = 1010

B[8] => S[8](01, 1110) = S[8][1][14] = 9 = 1001

نحوه ي محاسبه :

B[n] => S[n][row][column]

n : دقیقا متناظر است با اندیس B .

Row : از کنار هم قرار گرفتن بیت اول و بیت آخر یک گروه 6 بیتی فوق تشکیل می شود.

Column : مابقی بیت ها ، ستون را تشکیل می دهند (یعنی از بیت ھاي 2 تا 5 . (

براي مثال:

011101 را در نظر بگیرید.

چون اولین گروه 6 بیتی عدد 48 بیتی ما را تشکیل می دھد ، 1=n خواهد بود.

براي تشکیل row ،دو بیت اول و آخر کنار هم قرار می گیرد، یعنی 01=Row .

Column تشکیل شده از بیت 2 تا 5 عدد فوق است یعنی 1110=Column .

نتیجه ي حاصل:

اولین سطر محاسباتی است که در بالا ذکر شد یعنی:

S[n][row][column] = S[1](01,1110)

براي محاسبه ي این موقعیت در جدول S1 ، ابتدا اعداد درون پرانتز ها به معادل دسیمال خود تبدیل می شوند، داریم :[14][1][1[S

جدول S1 هم به صورت زیر است

[[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],

]

سپس این موقعیت در جدول S1 پیدا می شود. یعنی به جدول S1 مراجعه کرده و سپس عدد قرار گرفته در سطر 1 و ستون

14 را پیدا می کنیم. این عدد معادل 3 است. سپس آن را به باینری تبدیل می نماییم.

بنابراین به صورت خلاصه داریم :

B[1] => S[1](01, 1110) = S[1][1][14] = 3 = 0011

به همین ترتیب براي سایر گروه های 6 بیتی عمل می شود.

نتیجه این قسمت کنار هم قرار دادن نتایج حاصل فوق، که یک عدد 32 بیتی دودویی را تشکیل می دهد.

مرحله ششم

پس از اعمال جداول جانشینی و یا همان Boxes-S بر روي Bi ها، بر روي نتیجه ی حاصل یک جایگذاری دیگر صورت می

گیرد.

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]

یعنی به صورت خلاصه تابع f که در چند قسمت قبل راجع به آن بحث شد به صورت زیر محاسبه می شود:

f = P(S[1](B[1])...S[8](B[8]))

براي مثال :

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

==>

f = 0010 0011 0100 1010 1010 1001 1011 1011

و در ادامه :

R[1] = L[0] XOR f(R[0] , K[1] )=

1100 1100 0000 0000 1100 1100 1111 1111 XOR 0010 0011 0100 1010 1010 1001 1011 1011

= 1110 1111 0100 1010 0110 0101 0100 0100

و L1 هم که قبلا محاسبه شده بود:

L[1] = R[0] = 1111 0000 1010 1010 1111 0000 1010 1010

تا اینجا یک راند از 16 راند الگوریتم پایان پذیرفت.

**شروع راند دوم:**

مطابق فرمول کلی عنوان شده داریم :

L[2] = R[1]

R[2] = L[1] + f(R[1], K[2])

که محاسبهي آن با توجه به مطالب گفته شده تاکنون ساده است این رویه تا پایان 16 راند ادامه دارد.

در پایان راند 16 ، حاصل کار L16 و R16 است. در اینجا جاي این دو بلاك معکوس می شود یعنی :

R[16]L[16]

و بر روي آن آخرین جایگردانی مطابق جدول زیر صورت می گیرد:

PI\_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]

براي مثال با توجه به اعداد انتخاب شده در این برنامه داریم :

L16 = 0100 0011 0100 0010 0011 0010 0011 0100

R16 = 0000 1010 0100 1100 1101 1001 1001 0101

===>

R[16]L[16] = 00001010 01001100 11011001 10010101 01000011 01000010 00110010

00110100

===>

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

00000101 (bin)

= 85E813540F0AB405 (hex)

و یا به صورت خلاصه :

M = 0123456789ABCDEF

کلید بکار گرفته شده براي كدگذاري:

Key = 13 34 57 79 9B BC DF F1

خروجی كدگذاري شده:

C = 85E813540F0AB405