1. 不考虑填充方式
2. 不考虑加密模式
3. 不考虑iv向量
4. 输入明文为8个字节，密钥为8个字节，输出结果也为8个字节

明文：0123456789ABCDEF

密钥：133457799BBCDFF1

**第1步：子密钥的生成**

把密钥转二进制

00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

根据PC1表，对密钥进行重新排列

int PC1\_Table[56] = {

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};

密钥编排后的结果

1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

接下来将这56比特长的密钥分成左右两部分，命名为C0，D0

C0 = 1111000 0110011 0010101 0101111

D0 = 0101010 1011001 1001111 0001111

循环左移规定的位数，得到C1，D1到C16，D16的十六个数据

key\_shift = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1];

C1 = 1110000 1100110 0101010 1011111

D1 = 1010101 0110011 0011110 0011110

C2D2可以对C1D1分别左移1位来得到

C2D2也可以对C0D0分别左移2位来得到

将C1D1拼接起来，继续后续的处理

C1D1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

根据PC2表进行重排，56bit长的CnDn只留下48比特

int PC2\_Table[48] = {

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};

重排后CnDn就是子密钥，命名为Kn

后续DES的十六轮运算中，第n轮就是用Kn，16个子密钥为

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

**第2步：明文的编排**

把明文转二进制

00000001 00100011 01000101 01100111 10001001 10101011 11001101 11101111

根据IP表，对明文进行初始置换

int IP\_Table[64] = {

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};

明文重新排列后的结果

11001100 00000000 11001100 11111111 11110000 10101010 11110000 10101010

**第3步：明文的运算**

首先将明文分成左右两半，像上面的密钥一样

L0 和 R0

L0：11001100 00000000 11001100 11111111

R0：11110000 10101010 11110000 10101010

16轮迭代的基本套路

Ln = Rn-1

Rn = Ln-1 + f(Rn-1, Kn)

以n = 1为例:

L1 = R0 = 11110000 10101010 11110000 10101010

R1 = L0 xor f(R0, K1)

f函数传入了R0与K1，即第1个子密钥

R0 = 11110000 10101010 11110000 10101010

K1 = 000110 110000 001011 101111 111111 000111 000001 110010

R0为32bit，K1为48bit，长度不同，没法按位异或

将32bit的R0扩展成48bit，使用到E表

E(R0) = 011110 100001 010101 010101 011110 100001 010101 010101

int E\_Table[48] = {

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};

进行密钥混合 E(R0) ^ K1

E(R0) = 011110 100001 010101 010101 011110 100001 010101 010101

K1 = 000110 110000 001011 101111 111111 000111 000001 110010

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

DES算法核心

将上一步的结果分成8块6bit

B1 = 011000

B2 = 010001

B3 = 011110

B4 = 111010

B5 = 100001

B6 = 100110

B7 = 010100

B8 = 100111

把B1-B8的值当成索引，在S1-S8盒中取值，规则如下：

以B1为例，值为011000，将它分成0 1100 0，得到

i = 00 即 0

j = 1100 即 12

在S1中查找第0行第12列的值(这里从0开始算)

int S\_Box[8][4][16] = {

// S1

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,

// S2

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,

// S3

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,

// S4

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,

// S5

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,

// S6

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,

// S7

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,

// S8

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};

第0行第12列的值为5

B1 = 5 = 0101

再以B2为例

B2 = 0 1000 1

i = 01 = 1

j = 1000 = 8

B2 = 12 = 1100

B1在S1中找，B2在S2中找，以此类推

最后8个6比特的值在S盒作用下变成8个4比特的值

S(K1+E(R0)) = 0101 1100 1000 0010 1011 0101 1001 0111

得到的结果进行P表的重排

int P\_Table[32] = {

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};

P(S(K1+E(R0))) = 0010 0011 0100 1010 1010 1001 1011 1011

F函数完事，即F = P(S(K1+E(R0)))

F的结果与L0异或

L1 = R0 = 11110000 10101010 11110000 10101010

R1 = Ln-1 xor f(Rn-1,Kn) = 11101111 01001010 01100101 01000100

一轮运算完毕，16轮运算的操作完全一致

L1: 11110000 10101010 11110000 10101010

R1: 11101111 01001010 01100101 01000100

L2: 11101111 01001010 01100101 01000100

R2: 11001100 00000001 01110111 00001001

L3: 11001100 00000001 01110111 00001001

R3: 10100010 01011100 00001011 11110100

L4: 10100010 01011100 00001011 11110100

R4: 01110111 00100010 00000000 01000101

L5: 01110111 00100010 00000000 01000101

R5: 10001010 01001111 10100110 00110111

L6: 10001010 01001111 10100110 00110111

R6: 11101001 01100111 11001101 01101001

L7: 11101001 01100111 11001101 01101001

R7: 00000110 01001010 10111010 00010000

L8: 00000110 01001010 10111010 00010000

R8: 11010101 01101001 01001011 10010000

L9: 11010101 01101001 01001011 10010000

R9: 00100100 01111100 11000110 01111010

L10:00100100 01111100 11000110 01111010

R10:10110111 11010101 11010111 10110010

L11:10110111 11010101 11010111 10110010

R11:11000101 01111000 00111100 01111000

L12:11000101 01111000 00111100 01111000

R12:01110101 10111101 00011000 01011000

L13:01110101 10111101 00011000 01011000

R13:00011000 11000011 00010101 01011010

L14:00011000 11000011 00010101 01011010

R14:11000010 10001100 10010110 00001101

L15:11000010 10001100 10010110 00001101

R15:01000011 01000010 00110010 00110100

L16:01000011 01000010 00110010 00110100

R16:00001010 01001100 11011001 10010101

L16和R16倒换，左右调一下顺序，结果为R16L16

R16L16 = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100

最后一步，根据FP表进行末置换

10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

int FP\_Table[64] = {

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};

上述数据的十六进制形式为

85 e8 13 54 0f 0a b4 05

末置换就是初始置换的逆运算

明文经过初始置换，不做任何处理再进行末置换，值等于它自身