

实验二 分组密码算法DES

学号： 姓名： 专业

一、实验目的

通过用DES算法对实际的数据进行加密和解密来深刻了解DES的运行原理。

二、实验原理

1. DES算法将明文分成64位大小的众多数据块，即分组长度为64位。同时用56位密钥对64位明文信息加密，最终形成64位的密文。如果明文长度不足64位，即将其扩展为64位（如补零等方法）。
2. 具体加密过程首先是将输入的数据进行初始置换（IP），即将明文M中数据的排列顺序按一定的规则重新排列，生成新的数据序列，以打乱原来的次序。然后将变换后的数据平分成左右两部分，左边记为L0，右边记为R0，然后对R0实行在子密钥（由加密密钥产生）控制下的变换f，结果记为f(R0, K1)，再与L0做逐位异或运算，其结果记为R1，R0则作为下一轮的L1。如此循环16轮，最后得到L16、R16，再对L16、R16实行逆初始置换IP⁻¹，即可得到加密数据。解密过程与此类似，不同之处仅在于子密钥的使用顺序正好相反。

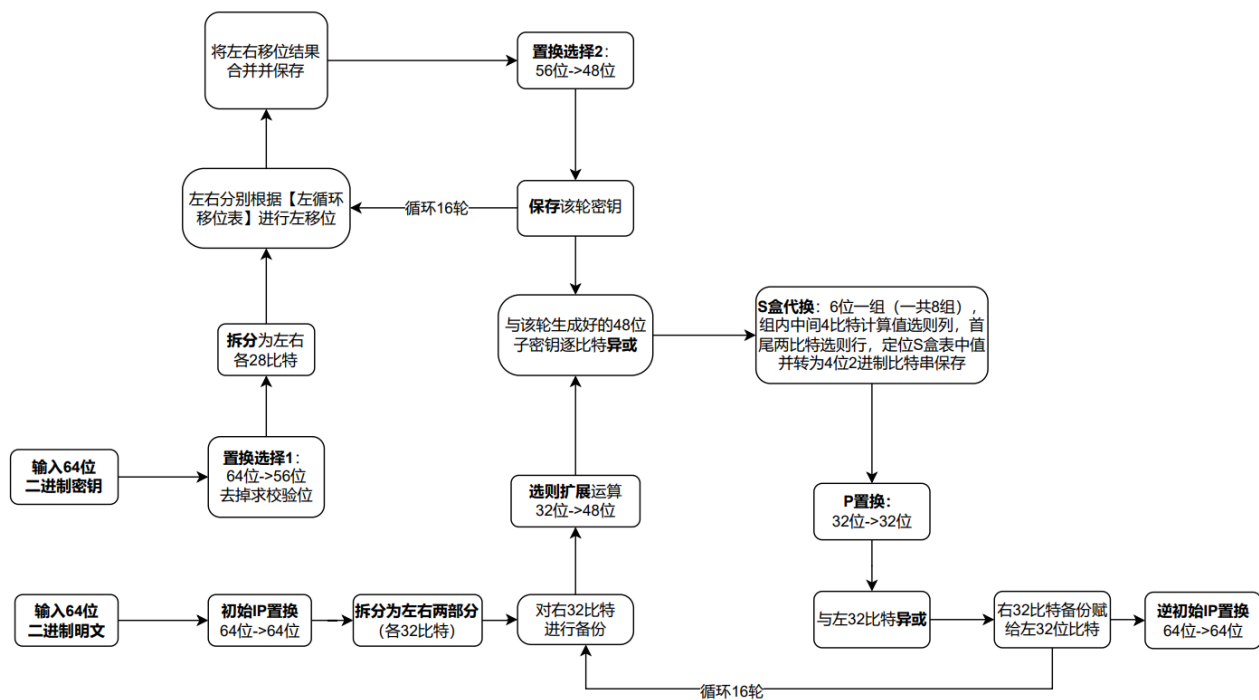
三、实验环境

运行Windows操作系统的PC机，具有VC等语言编译环境

四、实验内容和步骤

1. 算法分析：对课本中DES算法进行深入分析，对初始置换、E扩展置换，S盒代换、轮函数、密钥生成等环节要有清晰的了解，并考虑其每一个环节的实现过程。
2. DES实现程序的总体设计：在第一步的基础上，对整个DES加密函数的实现进行总体设计，考虑数据的存储格式，参数的传递格式，程序实现的总体层次等，画出程序实现的流程图。
3. 在总体设计完成后，开始具体的编码，在编码过程中，注意要尽量使用高效的编码方式。
4. 利用3中实现的程序，对DES的密文进行雪崩效应检验。即固定密钥，仅改变明文中的一位，统计密文改变的位数；固定明文，仅改变密钥中的一位，统计密文改变的位数。

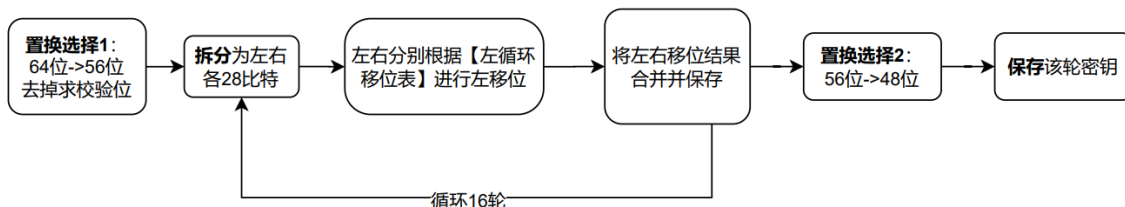
五. 整体流程图



六. DES加密与解密

1. 十六轮子密钥生成

。流程图



。代码

```

1 void Gen_Subkeys(bitset<64> key, bitset<48>* subKey)
2 {
3     bitset<28> left_key;    //子密钥左半部分
4     bitset<28> right_key;   //子密钥右半部分
5     bitset<56> new_key;     //保存更新的密钥
6
7     // 密钥完成置换选则1: 64位 -> 56位 (去掉奇偶标记位)
8     for (int i = 0; i < 56; ++i)
9         new_key[55 - i] = key[64 - PC_1[i]];
10
11    // 通过16轮迭代, 得到每一次迭代的48位用作和明文F函数加密的子密钥
12    for (int cycle = 0; cycle < 16; cycle++)
13    {
14        //把密钥分为左右两半
15        for (int i = 0; i < 28; ++i)
16            right_key[i] = new_key[i];
17        for (int i = 28; i < 56; ++i)
18            left_key[i - 28] = new_key[i];
19    }

```

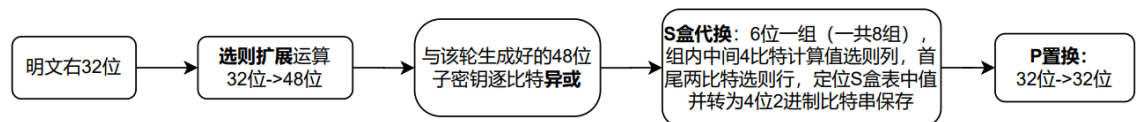
```

20 // 左右分别根据做循环移位表进行左移
21 left_key = leftShift(left_key, Left_Shift[cycle]);
22 right_key = leftShift(right_key, Left_Shift[cycle]);
23
24 // 密钥完成置换选则2: 56位->48位
25 for (int i = 28; i < 56; ++i)
26     new_key[i] = left_key[i - 28];
27 for (int i = 0; i < 28; ++i)
28     new_key[i] = right_key[i];
29 for (int i = 0; i < 48; ++i)
30     subkey[cycle][47 - i] = new_key[56 - PC_2[i]];
31 }
32 }

```

2. F函数

。流程图



。代码

```

1 bitset<32> F(bitset<32> R, bitset<48> K)
2 {
3     bitset<48> Right_Expand; //保存选则扩展运算E后的48位
4     bitset<32> res; //保存P置换之后结果
5
6     // 选则扩展运算E: 32位 -> 48位
7     for (int i = 0; i < 48; ++i)
8         Right_Expand[47 - i] = R[32 - Extend_Table[i]];
9
10    // 与子密钥进行异或运算
11    Right_Expand = Right_Expand ^ K;
12
13    // S盒代换: 48位 -> 32位
14    int index = 31;
15    for (int i = 0; i < 48; i = i + 6) //s1~s8 一组六个值
16    {
17        int row = Right_Expand[47 - i] * 2 + Right_Expand[47 - i - 5];
18        int col = 0;
19        for (int j = 1; j <= 4; j++)
20        {
21            col += Right_Expand[47 - i - j];
22            if (j != 4)
23                col *= 2;
24        }
25        int num = S_Box[i / 6][row][col];
26        bitset<4> binary(num);
27        for (int j = 0; j < 4; j++)
28            res[index--] = binary[3 - j];
29    }
30

```

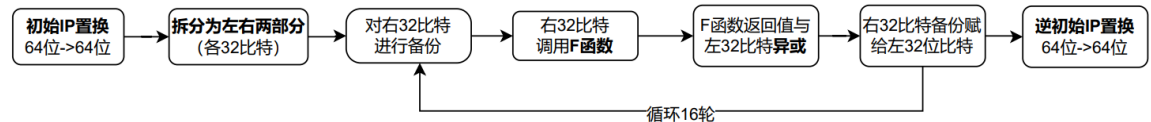
```

31 // P置换: 32位 -> 32位
32 bitset<32> tmp = res;
33 for (int i = 0; i < 32; ++i)
34     res[31 - i] = tmp[32 - P_Table[i]];
35
36 return res;
37 }

```

3. 加密总函数

。流程图



。代码

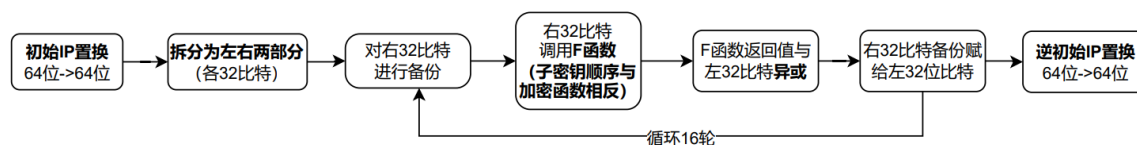
```

1 bitset<64> encrypt(bitset<64>& initial, bitset<48>* subkey)
2 {
3     bitset<64> result;           //保存最终结果
4     bitset<64> combine;         //保存初始置换后的64位输入/左右合并的加密结果
5     bitset<32> left;            //左边的32位
6     bitset<32> right;           //右边的32位
7     bitset<32> temp;            //对右明文进行备份
8
9     // 初始置换IP过程
10    for (int i = 0; i < 64; ++i)
11        combine[63 - i] = initial[64 - IP_Table[i]];
12
13    // 分为左右两部分
14    for (int i = 32; i < 64; ++i)
15        left[i - 32] = combine[i];
16    for (int i = 0; i < 32; ++i)
17        right[i] = combine[i];
18
19    // 进行迭代 (共16轮)
20    for (int cycle = 0; cycle < 16; cycle++)
21    {
22        temp = right;
23        right = left ^ F(right, subkey[cycle]);
24        left = temp;
25    }
26
27    // 左右交换
28    for (int i = 0; i < 32; ++i)
29        combine[i] = left[i];           //把left的放在高位
30    for (int i = 32; i < 64; ++i)
31        combine[i] = right[i - 32];    //把right的放在低位
32
33    // 逆初始置换
34    for (int i = 0; i < 64; ++i)
35        result[63 - i] = combine[64 - Rev_IP[i]];
36    return result;
37 }

```

4. 解密总函数

流程图



代码

唯一与加密总函数不同的地方在迭代16轮的时候，子密钥和加密时使用的顺序相反

```
1 bitset<64> decrypt(bitset<64>& cipher, bitset<48>* subKey)
2 {
3     bitset<64> result;    //保存最终结果
4     bitset<64> combine;   //保存初始置换后的64位输入/左右合并的解密结果
5     bitset<32> left;      //左边的32位
6     bitset<32> right;     //右边的32位
7     bitset<32> temp;      //对右明文进行备份
8     // 初始置换IP
9     for (int i = 0; i < 64; ++i)
10         combine[63 - i] = cipher[64 - IP_Table[i]];
11
12     // 拆分为左右两部分
13     for (int i = 0; i < 32; ++i)
14         right[i] = combine[i];
15     for (int i = 32; i < 64; ++i)
16         left[i - 32] = combine[i];
17
18     for (int cycle = 0; cycle < 16; cycle++){
19         temp = right;
20         right = left ^ F(right, subKey[15 - cycle]);
21         left = temp;
22     }
23
24     // 左右交换
25     for (int i = 0; i < 32; ++i)
26         combine[i] = left[i];
27     for (int i = 32; i < 64; ++i)
28         combine[i] = right[i - 32];
29
30     // 逆初始置换
31     for (int i = 0; i < 64; ++i)
32         result[63 - i] = combine[64 - Rev_IP[i]];
33     // 返回明文
34     return result;
35 }
```

七. 运行结果

1. 加密

十六进制密钥: 0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A

十六进制明文: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

十六进制密文: 0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02

```
D:\算法\密码学\2_分组密码算法DES\DES\Debug\DES.exe
===== 1——加密 2——解密 =====
请输入: 1
=====加密=====
请输入十六进制密钥:
10 31 6E 02 8C 8F 3B 4A
请输入十六进制明文:
00 00 00 00 00 00 00 00
=====
二进制初始密钥为:
0001000000110001011011100000001010001100100011110011101101001010
二进制明文为:
0000000000000000000000000000000000000000000000000000000000000000
二进制密文为:
1000001011011100101110101111011110111101110101010110110011000000010
十六进制密文为:
0x82 0xDC 0xBA 0xFB 0xDE 0xAB 0x66 0x02
=====
```

2. 解密

十六进制密钥: 0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A

十六进制密文: 0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02

十六进制明文: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

```
D:\算法\密码学\2_分组密码算法DES\DES\Debug\DES.exe
===== 1——加密 2——解密 =====
请输入: 2
=====解密=====
请输入密钥 (hex):
10 31 6E 02 8C 8F 3B 4A
请输入密文 (hex):
82 DC BA FB DE AB 66 02
=====
二进制初始密钥为:
0001000000110001011011100000001010001100100011110011101101001010
二进制密文为:
1000001011011100101110101111011110111101110101010110110011000000010
下面输出64位明文:
0000000000000000000000000000000000000000000000000000000000000000
明文的十六进制形式为:
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
=====
```

八. 检验雪崩效应

1. 明文雪崩效应

。核心主函数代码

```
1  cout << "===== 2--检验明文雪崩效应 =====" << endl;
2  // 输入十六进制密钥并保存为64位二进制
3  cout << "请输入十六进制密钥: " << endl;
4  for (int i = 0; i <= 7; i++)
5      cin >> (hex) >> hex_key[i];
6  bin_key_original = Hex_Bin(hex_key);
7
8  // 输入十六进制明文并保存为64位二进制
9  cout << "请输入十六进制明文: " << endl;
10 for (int i = 0; i <= 7; i++)
11     cin >> (hex) >> hex_text[i];
12 bin_text_original = Hex_Bin(hex_text);
13 // 得到十六轮密钥
14 Gen_Subkeys(bin_key_original, subkey);
15
16 // 加密过程
17 encrypt_text_original = encrypt(bin_text_original, subkey);
18
19 cout << "===== " << endl;
20
21 cout << "二进制初始密钥为: " << endl;
22 cout << bin_key_original << endl;
23 cout << "二进制初始明文为: " << endl;
24 cout << bin_text_original << endl;
25 cout << "二进制初始密文为" << endl;
26 for (int i = 0; i <= 63; i++)
27     cout << encrypt_text_original[63 - i];
28 cout << endl;
29 cout << "===== " << endl;
30
31 // 分别改变密钥每一位检验雪崩效应
32 for (int i = 0; i < 64; i++)
33 {
34     bitset<64> encrypt_text; //保存获得的密文
35     bitset<64> bin_key = bin_key_original;
36     bitset<64> bin_text = bin_text_original;
37     bin_text[63 - i] = 1 - bin_text[63 - i];
38
39     // 得到十六轮密钥
40     Gen_Subkeys(bin_key, subkey);
41
42     // 加密过程
43     encrypt_text = encrypt(bin_text, subkey);
44
45     //统计密文不同位数
46     int num = 0;
47     for (int m = 0; m < 64; m++)
48         if (encrypt_text_original[m] != encrypt_text[m])
49             num++;
50     if (i % 5 == 0)
51     {
```

```

52         cout << "改变第" << i + 1 << "位明文-密文改变位数为：" << num
    << endl;
53         cout << "新密文为：";
54         for (int i = 0; i <= 63; i++)
55             cout << encrypt_text[63 - i];
56         cout << endl;
57     }
58     total += num;
59 }

```

- 。64位明文逐一改变，下图中每改变5次输出一次，计算得出平均密文改变的位数位31.77

```

D:\算法\密码学\2_分组密码算法\DES\Avalanche\Debug\Avalanche.exe
===== 1——检验密钥雪崩效应 2——检验明文雪崩效应 =====
请输入：2
===== 2——检验明文雪崩效应 =====
请输入十六进制密钥：
01 01 01 01 01 01 01 01
请输入十六进制明文：
DD 7F 12 1C A5 01 56 19
=====
二进制初始密钥为：
0000000010000000010000000100000001000000010000000100000001
二进制初始明文为：
1101110101111111000100100001110010100101000000010101011000011001
二进制初始密文为
0100000000000000000000000000000000000000000000000000000000000000
=====
改变第1位明文—密文改变位数为： 33
新密文为： 10100100001111001010101001001001101010011101001010101010101011
改变第6位明文—密文改变位数为： 34
新密文为： 11110101100111100101000110011100111110010010000110001000111111011
改变第11位明文—密文改变位数为： 29
新密文为： 1110111001100111001000111100111101000010100101000000000101110001
改变第16位明文—密文改变位数为： 22
新密文为： 01100001000010000100001000100111011000100100011000100001101110001001
改变第21位明文—密文改变位数为： 33
新密文为： 0000011010100110101111110110111100100101010011001111100000110000
改变第26位明文—密文改变位数为： 40
新密文为： 1000100101111110101010011101100001100111100101100111111111011011
改变第31位明文—密文改变位数为： 31
新密文为： 0011010110110011101011001010001100111001100101010001010110100000
改变第36位明文—密文改变位数为： 30
新密文为： 1010001011000111001100110001000001111001101000110011000010011011
改变第41位明文—密文改变位数为： 35
新密文为： 101101010101000001110110100011101101100000111110001111100010011
改变第46位明文—密文改变位数为： 32
新密文为： 0001100111111110111110000000001110110110100110110100100000000110
改变第51位明文—密文改变位数为： 32
新密文为： 1001011110001010100001010101111111111100000011000010100101110000
改变第56位明文—密文改变位数为： 25
新密文为： 000101010100000000000000000110110100011011010100111000111100000110
改变第61位明文—密文改变位数为： 33
新密文为： 1001010011001000010011001001110111100010111100110011011000011101
31.77

```

2. 密钥雪崩效应

除去奇偶校验位之外的密钥逐一改变，下图中每改变5次输出一次，计算出平均密文改变位数为31.66


```

===== 1——检验密钥雪崩效应 2——检验明文雪崩效应 =====
请输入: 1
===== 1——检验密钥雪崩效应 =====
请输入十六进制密钥:
01 01 01 01 01 01 01 01
请输入十六进制明文:
DD 7F 12 1C A5 01 56 19
=====
二进制初始密钥为:
000000001000000010000000100000001000000010000000100000001
二进制初始明文为:
1101110101111111000100100001110010100101000000010101011000011001
二进制初始密文为:
0100000000000000000000000000000000000000000000000000000000000000
=====
改变第1位密钥—密文改变位数为: 30
新密文为: 0011000101101100001010001101100010100110011000001111000100101111
改变第6位密钥—密文改变位数为: 38
新密文为: 101010010010111001110111101010011011110110111001010101000011
改变第11位密钥—密文改变位数为: 31
新密文为: 00100000001000101110010110111101110000110110010001000011101
第16位密钥为奇偶校验位, 无需改变
改变第21位密钥—密文改变位数为: 29
新密文为: 0111011010000011111011001100001010000010100011011011110010000101
改变第26位密钥—密文改变位数为: 30
新密文为: 1101110101000111110101100010010101100111010100010001000001110010
改变第31位密钥—密文改变位数为: 26
新密文为: 1101101011011001010100101001001011010100010010000100011000000011
改变第36位密钥—密文改变位数为: 32
新密文为: 0010011001100010111110101101001111111011100000000010111000000
改变第41位密钥—密文改变位数为: 37
新密文为: 100101001011010111111000010001001100101101111111011010100110101
改变第46位密钥—密文改变位数为: 29
新密文为: 0101111101010001101101010110111000100010000010001100010100101011
改变第51位密钥—密文改变位数为: 34
新密文为: 0010000100011011110010100110110111100011001010111101010000111110
第56位密钥为奇偶校验位, 无需改变
改变第61位密钥—密文改变位数为: 30
新密文为: 1111011100001111010100010010111001100101000110100010100010100011
31.68

```

九. 实验中遇到的bug

在检验密钥雪崩效应的时候, 初始的实验结果发现有的地方改变密钥之后密文并没有改变, 当时思考之后发现是因为我把奇偶校验位也做了改变, 但其实奇偶校验位在密钥置换选择一的时候会被扔掉, 所以就算改变改位置也不会对最终密文产生影响。

```

新密文为: 000000001100001100101011010111111000110110001000011011000101111
改变第6位密钥—密文改变位数为: 38
新密文为: 101010010010111001110111101010011011110110111001010101000011
改变第7位密钥—密文改变位数为: 34
新密文为: 0000000010011111110011101111111001100100110100000010111010110101
改变第8位密钥—密文改变位数为: 0
新密文为: 0100000000000000000000000000000000000000000000000000000000000000
改变第9位密钥—密文改变位数为: 32
新密文为: 0011001000100001100001010100011111100011110001111011110010110100
改变第10位密钥—密文改变位数为: 30
新密文为: 100101001111011011110100000011001000000111010000111001001001100
改变第11位密钥—密文改变位数为: 31

```