DES 算法

17343023 董宸宇

1. 简介

   DES 算法是一种最常见也是安全性很高的加密算法，算法本身的流程并没有什么特别难以理解的地方，但是算法的步骤还是比较繁琐的

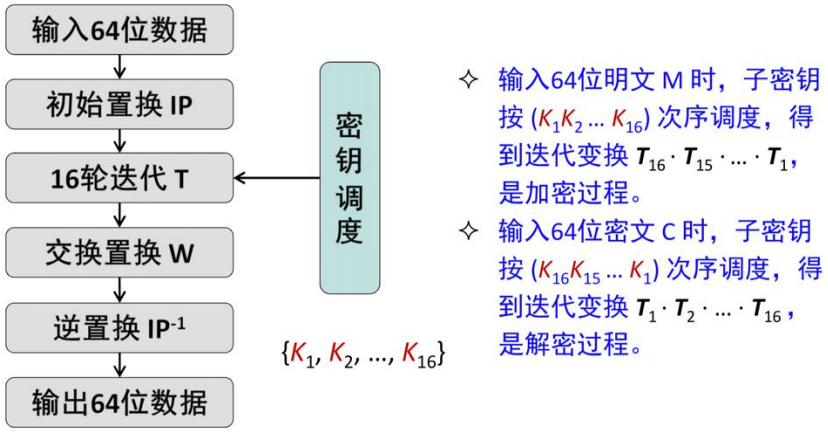
2. 算法原理与具体操作

   算法输入一个 16 位十六进制/64 位二进制数字(明文，也就是我们要传输的信息)，还需要一个同样是 16 位十六进制/64 位二进制数字作为密钥，密钥的作用是告诉我们明文怎么转换成其他人看不懂的密文，输出为一个 16 位十六进制/64 位二进制数字的密文，也就是将明文加密之后所得到的结果，同时不但要有加密的算法，还要有解密的算法，在收到密文并且有密钥的时候，能通过算法来求出明文，当然，整个算法也是要经过一些步骤的

   算法的过程与流程图大致如下

- **DES 算法概要**
  - 加密过程
    - $C = E_k(M) = IP^{-1} \cdot W \cdot T_{16} \cdot T_{15} \cdot \ldots \cdot T_1 \cdot IP(M)$.
      - M 为算法输入的64位明文块；
      - E_k 描述以 K 为密钥的加密函数，由连续的过程复合构成；
      - IP 为64位初始置换；
      - $T_1, T_2, \ldots, T_{16}$ 是一系列的迭代变换；
      - W 为64位置换，将输入的高32位和低32位交换后输出；
      - IP^{-1} 是 IP 的逆置换；
      - C 为算法输出的64位密文块。
  - 解密过程
    - $M = D_k(C) = IP^{-1} \cdot W \cdot T_1 \cdot T_2 \cdot \ldots \cdot T_{16} \cdot IP\ (C)$.

DES 算法的总体结构 — *Feistel* 结构



当然，光看上面两图的话肯定有些懵，不知道具体操作的方法，在下面详细说明
我们首先要把我们输入的 16 进制数字转换成二进制，因为后面进行的操作都是基于二进制进行，包括异或，移位等
关于 DES 的密钥，有一点是很重要的，就是密钥虽然是 64 位二进制数字，但是每第 8N 位都没有被用到，所以实际发挥作用的是其中的 56 位，而这个 64 位的密钥要进行一下变换，变换矩阵如下

PC-1 置换表						
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

简单说一下这个变换矩阵，它的意思是将我们 64 位密钥(第一位在图里面表示为 1，而不是 0)转换成一个 56 位的密钥
原密钥的第 57 位为新密钥的第 1 位
原密钥的第 49 位为新密钥的第 2 位
原密钥的第 41 位为新密钥的第 3 位
原密钥的第 33 位为新密钥的第 4 位
原密钥的第 25 位为新密钥的第 5 位

具体操作如下，先声明一个表示上面矩阵的数组(一维二维均可，一维方便一点)

```
int key1[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
};
```

接下来我们将这个 56 位密钥分为两串，一串为密钥的前一半 C0，另一半为密钥的后一半 D0,由于我是用字符串表示的所以直接使用 substr

```
string key_change(string key){
    int len=key.size();
    string result;
    for(int i=0;i<56;i++){
        char c=key[key1[i]-1];
        result.push_back(c);
    }
    return result;
}
```

上面这个函数就是将 64 位密钥变成 56 位的转换函数

接下来我们将这个 56 位密钥分为两串，一串为密钥的前一半 C0，另一半为密钥的后一半 D0,由于我是用字符串表示的所以直接使用 substr()函数

```
string key_1=key_change(bin_key);
string C0=key_1.substr(0,28);
string D0=key_1.substr(28,28);
C.push_back(C0);
D.push_back(D0);
```

在 DES 中需要的密钥不止 C0 和 D0,而我们接下来要基于 C0 和 D0 创建 C1—C16,D1—D16,而这一系列密钥都是根据 C0 和 D0 进行移位变换而来，变换规则是我们规定 C(n)/D(n)为在 C(m)/D(m)的基础上左移 m 位得到

左移 m 位：将二进制数字前 m 位移到最后，其他位就自然而然的向左移了 m 位

而关于 m 的确定，则由下面这个映射关系来确定

1—1

2—1

3—2

4—2

5—2

6—2
7—2
8—2
9—1
10—2
11—2
12—2
13—2
14—2
15—2
16—1
解释一下上面这组数据的意思，C1/D1 由 C0/D0 左移一位得到，C3/D3 由 C2/D2 左移两位得到
具体操作如下

```
int left_shift[16]={1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};
string move(string s,int shift){
    string result;
    int len=s.size();
    string temp=s.substr(0,shift);
    for(int i=shift;i<len;i++){
        result.push_back(s[i]);
    }
    result+=temp;
    return result;
}
```

上面这个函数代表将字符串 s 左移 shift 位
接下来建立两个 vector 用来存储 C 和 D 两个系列的密钥

```
C.push_back(C0);
D.push_back(D0);
for(int i=0;i<16;i++){
    string c,d;
    c=move(C[i],left_shift[i]);
    d=move(D[i],left_shift[i]);
    C.push_back(c);
    D.push_back(d);
}
```

这样 C 和 D 里面分别存储了 17 个元素，分别是 C0—C16，D0—D16
然后我们要把每对 Cn 和 Dn 拼接起来(1<=n<=16)
在这里把拼接之后的结果用一个 vector K0 来存储

```
for(int i=0;i<17;i++){
    string temp=C[i]+D[i];
    K0.push_back(temp);
}
```

这时我们的每个拼接之后的密钥都有 56 位，我们还需要对它们进行变换，把他们变成 48 位，变换矩阵如下

PC-2 压缩置换表					
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

变换原理跟上面所提及的类似
具体操作如下

```
int key2[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
};
```

```
string key_change2(string key){
    int len=key.size();
    string result;
    for(int i=0;i<48;i++){
        char c=key[key2[i]-1];
        result.push_back(c);
    }
    return result;
}
```

经过上面的函数，就可以将 56 位的密钥转换成 48 位
然后我们将 C1—C16 与 D1—D16 依次配对
并将拼接而成的密钥由 56 位转变为 48 位

```
for(int i=0;i<17;i++){
    string temp=key_change2(K0[i]);
    K.push_back(temp);
}
```

在这里我们完成了对密钥的处理，接下来我们要处理明文
首先也是对明文进行一个变换，不过变换之后的明文还是 64 位，只不过数字的顺序跟原来有很大区别
变换矩阵如下

IP 置换表（64位）							
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

体现在具体操作中

```
int IP1[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
};
```

```
string IP_change(string IP){
    string result;
    int len=IP.size();
    for(int i=0;i<len;i++){
        char c=IP[IP1[i]-1];
        result.push_back(c);
    }
    return result;
}
```

接下来我们要把变换的来的字符串分为左右两边，分别为 L0 和 R0，其中 L0 位前 32 位，R0 为后 32 位

```
string IP=IP_change(bin_input);
string L0=IP.substr(0,32);
string R0=IP.substr(32,32);
```

在这里我们依旧使用两个 vector 来存储 Ln 与 Rn
接下来我们要求的是 Ln 与 Rn，公式如下

L(n)=R(n−1)

R(n)=L(n−1)^f(Rn−1,Kn)(其中^位按位异或操作)

具体操作如下

```
string XOR_(string a,string b){
    string result;
    int len=a.size();
    for(int i=0;i<len;i++){
        if(a[i]==b[i]){
            result.push_back('0');
        }
        else{
            result.push_back('1');
        }
    }
    return result;
}
```

这个是对字符串进行按位异或操作

接下来还有一个问题就是上面的 f 函数如何工作

首先要把每个 Rn 从 32 位拓展为 48 位

拓展矩阵如下

E-扩展规则 (比特-选择表)					
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

具体操作如下

```
int E1[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,
};
```

```
string E(string R){
    string result;
    int len=R.size();
    for(int i=0;i<48;i++){
        char c=R[E1[i]-1];
        result.push_back(c);
    }
    return result;
}
```

这样的话我们就把每个 Rn 由 32 位拓展到 48 位

然后把我们拓展而成的结果与相应的 Kn 做异或运算，不过这里面相互之间进行配对的是 Kn 和 R(n−1)

这样我们得到的结果是一个 48 位的二进制数，我们还要对其进行操作，使其从 48 位变成 32 位，不过这个变换有点绕

在原 48 位数字中，我们每 6 位分成一组，这样会分成 8 组，分别命名为 S1,S2,S3,S4,S5,S6,S7,S8,我们把每一组数字从 6 位变成 4 位,同样是通过变换矩阵

S1−S8 所对应的变换矩阵分别如下

S_1-BOX															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	15	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

S_2-BOX															
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

S_3-BOX															
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

S_4-BOX															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
12	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

S_5-BOX															
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

S_6-BOX															
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

S_7-BOX															
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

S_8-BOX															
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

我们可以看到每个矩阵都是 4*16 的矩阵

这次的变换规则跟我们前面有些不一样，我们把六位数拆成两组，第一位和最后一位一组，称为 A，剩下 4 位一组，称为 B

举个例子：S1=110010

在这里面 A=10(第一位和最后一位)，B=1001(2-5 位组合)

转换成 10 进制数的话 A=2，B=9

那么其对应的数字就是 S1 矩阵里面的第 3 行第 10 列(因为是从 0 开始)，12

转换成 2 进制数就是 1100，所以在这里面 110010 转换得到 1100

具体操作如下

```
int S1[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
};
int S2[4][16]={
    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
};
int S3[4][16]={
    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
};
int S4[4][16]={
    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
};
```

```
int S5[4][16]={
    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,
};
int S6[4][16]={
    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
};
int S7[4][16]={
    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
};
int S8[4][16]={
    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
};
```

先声明好 8 个矩阵，这里更适合使用二维数组

接下来就是进行变换

```
string S(string s){
    string result;
    for(int i=0;i<8;i++){
        string temp=s.substr(i*6,(i+1)*6);
        int row=(temp[0]-'0')*2+(temp[5]-'0');
        int column=(temp[1]-'0')*8+(temp[2]-'0')*4+(temp[3]-'0')*2+(temp[4]-'0');
        int target;
        if(i==0){
            target=S1[row][column];
        }
        if(i==1){
            target=S2[row][column];
        }
        if(i==2){
            target=S3[row][column];
        }
        if(i==3){
            target=S4[row][column];
        }
        if(i==4){
            target=S5[row][column];
        }
        if(i==5){
            target=S6[row][column];
        }
        if(i==6){
            target=S7[row][column];
        }
        if(i==7){
            target=S8[row][column];
        }
        result+=hex_[target];
    }
    return result;
}
```

上面的 hex_是 0—15 每个数字对应的二进制字符串表示

```
string hex_[16]={"0000","0001","0010","0011","0100",
"0101","0110","0111","1000","1001","1010","1011","1100","1101","1110","1111"};
```

接下来我们再对我们所得到的 32 位数字进行一个矩阵映射的变换

```
int P[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
};
```

```
string P_(string s){
    string result;
    for(int i=0;i<32;i++){
        char c=s[P[i]-1];
        result.push_back(c);
    }
    return result;
}
```

这样我们就完成了 f 函数的所有操作，然后来看一下 f 函数的总体操作

```
string f(string R0,string K1){
    string temp=E(R0);
    string temp1=XOR_(K1,temp);
    string temp2=S(temp1);
    string result=P_(temp2);
    return result;
}
```

接下来我们就要进行迭代，将 L1—L16/R1—R16 分别求出来

```
for(int i=1;i<=16;i++){
    string templ=R[i-1];
    L.push_back(templ);
    string tempr=XOR_(L[i-1],f(R[i-1],K[i]));
    R.push_back(tempr);
}
```

接下来将 R16 与 L16 拼接起来
然后进行一个变换，变换矩阵如下

IP^{-1} 置换表（64位）							
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

具体操作和之前比较类似

```
int IP2[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,
};
```

```
string IP_2(string str){
    string result;
    for(int i=0;i<64;i++){
        result.push_back(str[IP2[i]-1]);
    }
    return result;
}
```

然后得到的结果就是我们加密之后的密文了

我们不但要进行加密，还要进行解密，即根据已有的密钥与收到的密文，将其翻译为明文

解密算法是加密算法的逆过程，二者在大体上很相似，只不过在某些细节上面略有不同

解密算法是将输入的密文进行第一次处理之后，将其一样按照前 32 位与后 32 位分为两组，只不过将前 32 位命名为 R16，后 32 位命名为 L16，然后在 16 次迭代的时候运算规则如下

R(n−1)=L(n)

L(n−1)=R(n)^f(L(n),K(n))(^代表异或操作)(n 从 16 开始取，一次递减，直到 1)

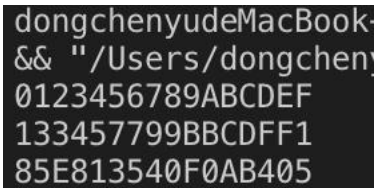
然后最后拼接的时候使用 L0 与 R0 进行拼接，拼接之后进行转换，即可得到结果

整个算法以及其每个模块的实现也就到此为止了

回顾整个算法，其实用到最多的是根据矩阵对数字进行变换，以及对二进制数字进行移位操作

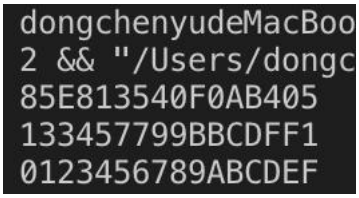
3. 代码测试结果

在加密算法中输入明文 123456789ABCDEF 以及密钥 133457799BBCDFF1



得到密文 85E813540F0AB405

接着测试解密算法输入密文 85E813540F0AB405 以及密钥 133457799BBCDFF1



得到明文 123456789ABCDEF