DES算法详细设计

一、DES算法原理概述

DES算法为密码体制中的对称密码体制，又被称为美国数据加密标准，是1972年美国IBM公司研制的对称密码体制加密算法。 明文按64位进行分组，密钥长64位，密钥事实上是56位参与DES运算（第8、16、24、32、40、48、56、64位是校验位， 使得每个密钥都有奇数个1）分组后的明文组和56位的密钥按位替代或交换的方法形成密文组的加密方法。DES是一种分组密码，是两种基本的加密组块替代和换位的细致而复杂的组合，它通过反复依次应用这两项技术来提高其强度，共经过16轮的替代和换位。使得密码分析者无法获得该算法一般特性以外更多的信息。
DES算法的实现大致可以分为以下几个步骤：**初始置换—密钥生成—16轮迭代变换—IP逆置换—密文。**
以下是一个DES的简单流程图：

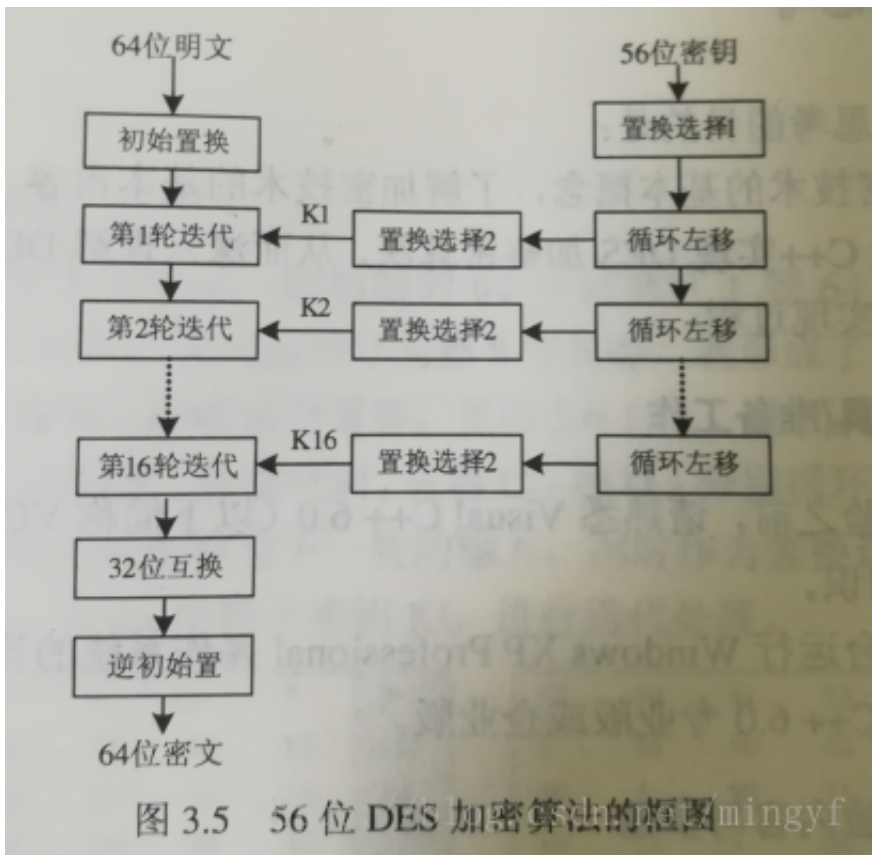

图 3.5　56 位 DES 加密算法的框图

二、分模块详细设计每一个模块

1、将明文与密文用二进制表示

2、初始置换

对输入的64位二进制明文按照2.1所示的置换表进行置换，打乱明文的顺序：

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

表2.1
将置换后的明文按顺序分为左右两组L0和R0，每一组都是32位的。IP初始置换的算 法设计如下图2.2所示：

```
//IP初始置换表
int IP_IniRep[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};
//IP初始置换的算法如下
void DES_IPinitial_Transform(elemtype data[64])
    elemtype temp[64];
    for (int i = 0; i < 64; i++)
        temp[i] = data[i];
    for (int i = 0; i < 64; i++)
        data[i] = temp[IP_IniRep[i]];   http://blog.csdn.net/mingyf
```

图2.2

3、密钥生成（密钥置换）

（1）56位密钥的生成过程：将初始的64位密钥从1到64位进行编号，形成一个矩阵， 如下表3.1所示，每一行的第八位将作为奇偶校验位被忽略，于是形成了初始的56位 的密钥输入：

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

表3.1

（2）接下来使用选择置换表（如下表3.2所示），进行置换操作

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

表3.2

密钥置换的算法如下图3.3所示：

```
//密钥置换
int KEY_Repl[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};
//密钥置换算法
void Key_Replace1(elemtype key[64], elemtype keyout1[56])
    for (int i = 0; i < 56; i++)
        keyout1[i] = key[KEY_Repl[i]];
```


图3.3

这个置换操作就是将第57位放到第1位，第49位放到第2位，将第4位放到最后一位，由于这里除去了奇偶校验位，所以不会包含6,16,24,32,40,48,56,64这八个数。

（3）将第（2）步得到的输出按顺序分为左右两个部分，每个部分有28位，分别用L0和R0来表示，根据轮迭代的轮数，在第1,2,9,16轮时分别将L0和R

0循环左移一位，其余轮数时循环左移两位，得到56位的输出。

轮数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
位数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

移位算法如下图3.4所示：

```
//迭代过程中密钥循环左移位数
int KEY_Mov[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};
void Move_left(elemtype C0[28], elemtype D0[28], int m)
    elemtype temp;
    elemtype temp1;
    elemtype temp2;
    elemtype temp3;
    if (m == 1 || m  == 2 || m == 9 || m == 16)
        temp = C0[0];
        temp1 = D0[0];
        for (int i = 0; i < 27; i++)
            C0[i] = C0[i + 1];
            D0[i] = D0[i + 1];
        C0[27] = temp;
        D0[27] = temp1;
    else
        temp = C0[0];
        temp1 = C0[1];
        temp2 = D0[0];
        temp3 = D0[1];
        for (int i = 0; i < 26; i++)
            C0[i] = C0[i + 2];
            D0[i] = D0[i + 2];
        C0[26] = temp;
        C0[27] = temp1;
        D0[26] = temp2;
        D0[27] = temp3;
```


图3.4

（4）再将第（3）得到的输出按照标3.5所示的置换表，进行密钥压缩置换，得到的就是在T迭代里面输入的密钥Ki：

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

表3.5

置换算法如下图3.6所示：

```
//密钥压缩置换表
int KEY_Comp[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};
void KEY_Compress(elemtype key[56], elemtype finalkey[48])
    for (int i = 0; i < 48; i++)
        finalkey[i] = key[KEY_Comp[i]];
```


图3.6

4、轮迭代

（1）总体过程：首先关于轮迭代的过程如下图4.1所示，Li–1和Ri–1是上一轮的迭代的输出，上一轮的左边的输出与经过轮函数之后的输出结果进行异或运算得到当前轮的右边输出，左边的结果是上一轮的右边的输出：

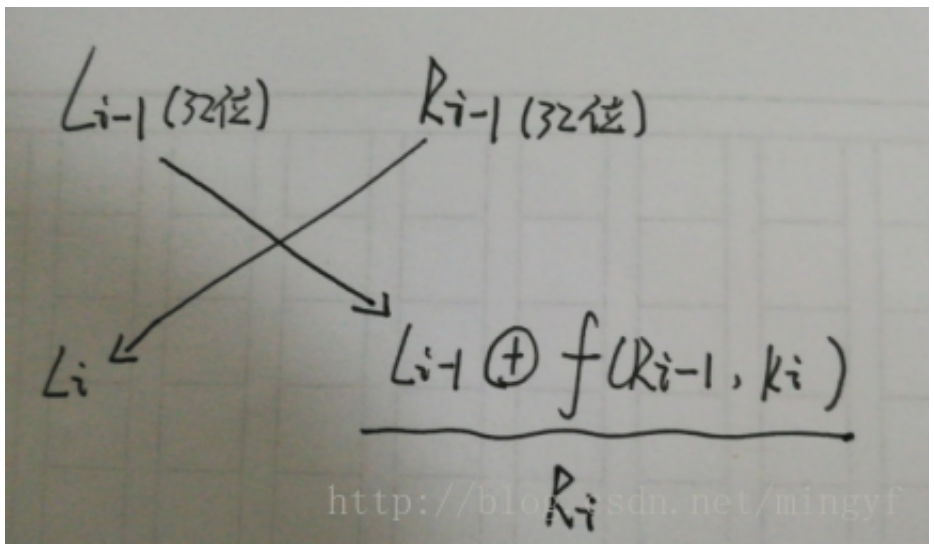

图4.1

（2）轮函数详细讨论

E扩展：首先将原来的明文数据的右半部分R从32位扩展成为48位，扩展置换表如4.2所示：

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

表4.2

上图中空白部分表示原来的32位，然后两边阴影部分的数据是扩展加上去的 数据，将第32位放置到第一位，将第5位数据放置到第6位的位置，由此就将32 位的数据扩展成为了48位的数据。

E扩展算法如下图4.3所示：

```
//E扩展置换
int E_Extend[48] = {31, 0, 1, 2, 3, 4,
                    3,  4, 5, 6, 7, 8,
                    7,  8,9,10,11,12,
                    11,12,13,14,15,16,
                    15,16,17,18,19,20,
                    19,20,21,22,23,24,
                    23,24,25,26,27,28,
                    27,28,29,30,31, 0};
void E_ExtendFunc(elemtype data[32], elemtype data1[48])
    for (int i = 0; i < 48; i++)
        data1[i] = data[E_Extend[i]];
```


图4.3

异或：将E扩展得到的48位的数据与密钥输入Ki进行异或运算，得到的48位的 输出数据作为”S盒”的输入。

异或操作算法如下图4.4所示：

```
//密钥与输入取异或运算
void XOR_Compute(elemtype data[48], elemtype key[48])
    for (int i = 0; i < 48; i++)
        data[i] ^= key[i];
```


图4.4

“S盒替换”：将48位数据按照每6位分为一组，一共分为8组，并分别输入S1， S2，S3，S4，S5，S6，S7，S8这8个盒子中，每个盒子产生4位的输出，将每个S 盒的输出拼接成32位。

S盒操作算法如下图4.5所示：

```
//S盒
int S[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}};
```

```
//S盒置换算法
void S_Replacement(elemtype data[48], elemtype data1[32])
    int temp1 = 0;
    int temp2 = 0;
    int row = 0;
    int col = 0;
    for (int i = 0; i < 8; i++)
        temp1 = i * 6;
        temp2 = i * 2;
        row = (data[temp1] << 1) + data[temp1 + 5];
        col = (data[temp1 + 1] << 3 + data[temp1 + 2] << 2
            + data[temp1 + 3] << 1 + data[temp1 + 4]);
        int out = s[i][row][col];
        data1[temp2] = out % 2;
        out /= 2;
        data1[temp2 + 1] = out % 2;
        out /= 2;
        data1[temp2 + 2] = out % 2;
        out /= 2;
        data1[temp + 3] = out % 2;
```


图4.5

P盒置换：将S盒的输出拼接得到32位的数据，对这些数据再一次进行置换，置 换表如下图4.6所示，置换之后的结果就是轮函数的输出结果：

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

图4.6

P盒置换算法如下图4.7所示：

```
//p盒置换
int P_Repl[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, 24};
void P_box_Repl(elemtype data[32])
    elemtype temp[32];
    for (int i = 0; i < 32; i++)
        temp[i] = data[i];
    for (int i = 0; i < 32; i++)
        data[i] = temp[P_Repl[i]];
```


图4.7

（3）S盒详细讨论

S盒是一个六进四出的处理盒子，也是DES算法中唯一没有公开技术细节的最 核心的部分。

S盒的使用方法：设S盒的输入为6位的二进制数b1b2b3b4b5b6,把b1b6两位 二进制数转换成十进制数，并作为S盒的行号i，把b2b3b4b5四位二进制数转 换成十进制数，并作为S盒的列号j,则对应S盒的(i, j)元素为S盒的十进制输出， 再将该十进制数转换为二进制数，就得到了S盒的4位二进制数输出。

S盒的内容 如下表4.8所示：

S盒1:

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

S盒2:

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盒3:

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盒4：

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	19
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盒5：

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

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盒7：

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盒8：

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

表4.8

（4）将轮函数的输出结果与上一轮得到的输出结果Li-1进行异或运算，得到的就是这 一轮的输出结果Ri。

算法如下图4.9所示：

```
//左边输入与轮函数输出结果取异或运算
void XOR_Compute(elemtype data[32], elemtype key[32])
    for (int i = 0; i < 32; i++)
        data[i] ^= key[i];
```


图4.9

按照这样的方法不停的继续下去直到第16轮。在第16轮的时候要将L16和R16互换位 置。

5、逆初始置换

经过16轮的迭代之后，将输出的L16和R16合并起来形成64位的二进制数，最后 按照表5.1进行逆初始置换，就可以得到64位的密文。

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

表5.1

逆初始置换算法实现如下图5.2所示：

```
//IP逆初始置换表
int IP_RevRep[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};
//IP逆初始置换算法
void DES_IPRevinitial_Transform(elemtype data[64])
    elemtype temp[64];
    for (int i = 0; i < 64; i++)
        temp[i] = data[i];
    for (int i = 0; i < 64; i++)
        data[i] = temp[IP_RevRep[i]];   http://blog.csdn.net/mingyf
```

图5.2

到此DES算法的详细设计的过程就结束了，从设计的过程可以看出它确实是一个相当复杂的过程。

6、DES解密过程

DES的解密过程在原理上是与加密过程一样的，只有极少的地方不一样：

（1）如果加密的时候子密钥的使用顺序是K1，K2……K16，那么解密的时候子密钥 的使用顺序是K16，K15……K1；

（2）输入是64位密文，输出是64位明文；

（3）在生成密钥的时候加密算法是向左循环移动，但是解密算法是向右循环移动。

三、数据结构

在设计的过程中为了方便使用将所有的数据类型都使用整数类型来表示，使用整数类型方便操作而且还不容易出错，只是在设计数组边界的地方需要小心一点，不然很容易出错。

四、小结

关于DES算法的设计首先要将大问题模块化，先实现每一个小的模块，然后再将小模块组合成为一个大模块，跟着老师上课所讲的思路走，按着那个思路来实现的话逻辑比较清晰也不容易出错。