DES

## 1.1.DES的ECB模式实现

ECB模式的DES算法，输入8字节的明文/密文、8字节的密钥，可执行加密/解密操作得到密文/明文。

DES本身是一个对称算法，加密和解密使用同一组算法即可实现。

### 1.1.1.概述

算法原理见图1：

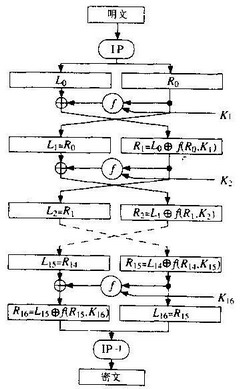


图1 DES算法原理图

详细描述如下：

**Algo. DES**

Input : unsigned char input[8];

Output : unsigned char output[8];

1. 将输入的8字节input内容，转换成64bit的表现形式后，通过IP\_table进行置换，得到全新序列的64bit数据；
2. 对step1中得到的数据，分组成为两个32bit的数据，表示为Li和Ri；
3. 对Li和Ri循环执行16轮变换，变换规则为：Li+1 = Ri; Ri+1 = Li xor f(Ri, Ki)；
4. 第16论的转换结束后，将left和right部分互换；
5. 将left和right拼接成64bit数据后，通过IP\_inv\_table进行反置换；
6. 将step4得到的64bit数据组合成8字节的形式保存到output中，并返回；

可见，算法中除step3外，其他步骤比较容易理解。对于每个步骤中使用到的表格，将在程序中给出定义，且这些表格是DES标准中已经定义好的，无需做任何改动。IP\_table定义如下：

const static unsigned char IP\_table[DES\_BLOCK\_BITS\_SIZE] = {

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

};

所谓的置换就是将原来的64bit数据位置打乱后重新放置，放置的规则就是上面的表格所规定的。例如原来的第58个bit，需要置换到第1个bit的位置上，等等。下面依据表格进行置换，都将按照这种规则进行。

IP\_inv\_table定义如下：

const static unsigned char ip\_inv\_table[DES\_BLOCK\_BITS\_SIZE] = {

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

};

step3中，Li+1无需赘述，主要的实现集中在Ri+1上。可以看到，要得到Ri+1，除了需要Li、Ri外，还需要一个Ki，该参数是由密钥扩展(1.1.2)得到的。

对于DES用户来说，无论是加密和解密，除去明文/密文外，自己肯定还要持有一个密钥，否则显然无法进行加解密的操作。这个密钥是64bit的数据。用户在进行加解密的时候，首先需要对这个密钥进行扩展，得到16个48bit的扩展密钥组合。这个扩展密钥集将在16轮循环中被依次使用，也就是上面step3中使用到的Ki。

除此之外，step3中对于还用到了一个函数：f()。该函数(1.1.3)将通过扩展置换、s盒代替、p盒置换三个步骤实现。

### 1.1.2.密钥扩展

如上所述，密钥扩展将8字节的用户手持密钥扩展得到[16][48bit]的密钥集，该密钥集将依次参与加密/解密过程。其流程图参见图2：

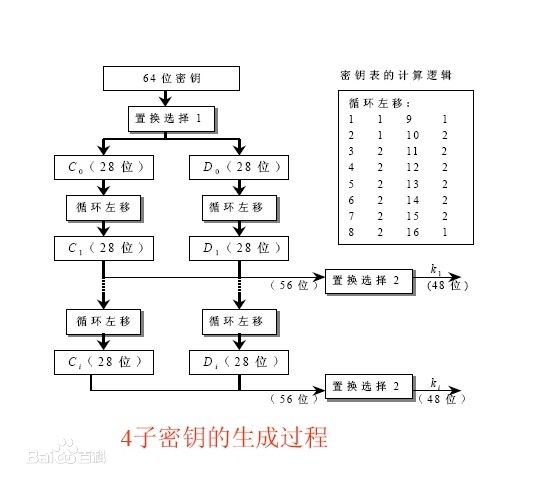


图2 密钥扩展(子密钥集合的生成)

其算法描述如下：

Input : key, 8byte data;

Output : subkey\_set, [16][ 48bit] data;

1. 对输入的8字节密钥，转换得到64bit形式；通过密钥置换表(key\_perm\_table)将其转换成56bit的数据；
2. 将step1中得到的56bit数据，分为Ci和Di两部分，长度都是28bit；
3. 对Ci和Di执行循环左移的操作，得到Ci+1和Di+1；循环左移的位数由表格rls\_table决定，16轮循环中每一轮对应各自的左移位数；
4. 得到的Ci+1和Di+1拼接成56bit的数据后，执行压缩置换；通过comp\_perm\_table(压缩置换表)压缩成为48bit的数据后保存；
5. 循环执行step3和step4，循环次数为16，最终得到16个48bit的数据集：subkey\_set[16][48bits]；

step1中用到了密钥置换表，该表的定义如下：

const static unsigned char key\_perm\_table[DES\_KEY\_PERM\_LEN] = {

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

};

执行该操作后，将有选择性的将64bit的数据，置换成56bit的数据并投入使用。step3中使用到的rls\_table定义如下：

const static unsigned char rls\_table[DES\_ROTATE\_ROUNDS] = {

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

};

该表格定义了每一轮循环执行时，Ci和Di将要循环左移的位数。step4中定义的压缩置换表定义如下：

const static unsigned char comp\_perm\_table[DES\_EXTENSION\_LEN] = {

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

};

该表格将56bit的数据，有选择性的进行压缩置换，置换后得到的数据将是48bit。

### 1.1.3.一轮DES详述

有了扩展密钥集后，将使用该密钥集和经过处理的明文/密文(1.1.1中step1)，执行16轮DES循环操作。每一轮操作的算法都是相同的，只不过输入参数是上一轮的输出，我们单独描述一轮DES的循环。

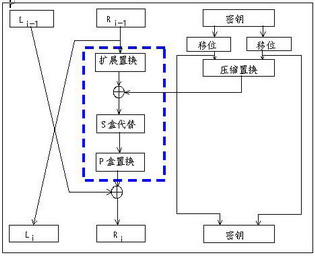


图3 一轮DES

详细描述如下：

Input : Li-1, Ri-1, Ki-1

Output : Li, Ri

1. 将Ri-1赋值给Li；
2. 通过扩展置换表ext\_perm\_table，将Ri-1从32bit扩展到48bit；
3. 将step2中得到的48bit数据，与对应的Ki-1做异或操作；
4. 将step3中得到的48bit数据，执行s盒代替，得到新的32bit数据；
5. 将step4中得到的32bit数据做p盒置换，得到32bit数据；
6. 将step5中得到的32bit数据，与32bit的Li-1进行异或操作后，结果赋值到Ri中；

扩展置换表将32bit的数据，扩展得到48bit的数据，其原理如图4所示：

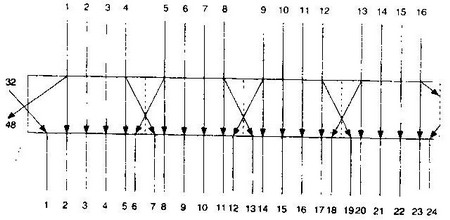


图4 扩展置换

其定义如下：

const static unsigned char ext\_perm\_table[DES\_EXTENSION\_LEN] = {

32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,

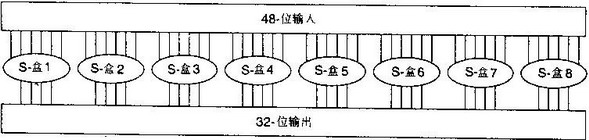
8, 9,10,11,12,13,12,13,14,15,16,17,

16,17,18,19,20,21,20,21,22,23,24,25,

24,25,26,27,28,29,28,29,30,31,32, 1

};

s盒代替是DES算法中的关键操作，因为整个算法中只有该操作不是线性的，也是算法安全性的保证。s盒代替将48bit的数据转换得到32bit的数据，其执行流程如图5所示：



详细描述如下：

Input : 48bit data;

Output : 32bit data;

1. 48bit数据分成8个6bit的数据；
2. 对每个6bit的数据，第1个bit和第6个bit组合，得到0—3之间的数字，标记为row\_num；中间4个bit组合得到0—f之间的数字，标记为column\_num；
3. 根据row\_num和column\_num，在s\_box中寻找对应的数值(1字节)，并将该值分解成4bit的形式，替换之前的6bit数据的位置；
4. 循环执行step2和step3两个操作，直到所有的8个6bit数据都替换成了4bit数据；

s\_box\_table的定义如下所示：

const static unsigned char s\_box\_table[DES\_BLOCK\_BYTES\_SIZE][DES\_BLOCK\_BITS\_SIZE] = {

{//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, 0, 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, 0, 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

}

};

最后执行p\_box的置换操作，将32位数据进行再一次的置换，该表格的定义如下：

const static unsigned char p\_box\_table[DES\_HALF\_PART\_LEN] = {

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

};

### 1.1.4.加密、解密

解密与加密的不同之处是：将subkey\_set按照15->0的顺序参与循环运算；