专业：信通 1006
姓名：卢俊
学号：3100102884
日期：2013.6.14
地点：\

浙江大学实验报告

课程名称：信息通信安全 指导老师：陈惠芳 成绩：_____
实验名称：C++实现DES算法 实验类型：验证型 同组学生姓名：单人

一、实验目的：

C/C++实现加解密算法及其应用：DES（或 AES）和基于 DES（或 AES）的 CMAC

(1). 复习 DES（或 AES）原理。

(2). 用 C/C++编写 DES（或 AES）算法并调试通过。

(3). 复习 CMAC 原理。

(4). 在实现 DES（或 AES）基础上，用 C/C++编写 CMAC 算法并调试通过。

(5). 回答下列思考题。

- DES（或 AES）解密算法和 DES（或 AES）的逆算法之间有什么不同？
- CMAC 与 HMAC 相比，有什么优点？

(6). 完成实验报告。

二、实验原理：

数据加密算法（Data Encryption Algorithm，DEA）的数据加密标准（Data Encryption Standard，DES）是规范的描述，它出自 IBM 的研究工作，并在 1997 年被美国政府正式采纳。它很可能是使用最广泛的秘钥系统，特别是在保护金融数据的安全中，最初开发的 DES 是嵌入硬 件中的。通常，自动取款机（Automated Teller Machine，ATM）都使用 DES。

DES 使用一个 56 位的密钥以及附加的 8 位奇偶校验位，产生最大 64 位的分组大小。这是一个迭代的分组密码，使用称为 Feistel 的技术，其中将加密的文本块分成两半。使用子密钥对其中一半应用循环功能，然后将输出与另一半进行“异或”运算；接着交换这两半，这一过程会继续下去，但最后一个循环不交换。DES 使用 16 个循环。

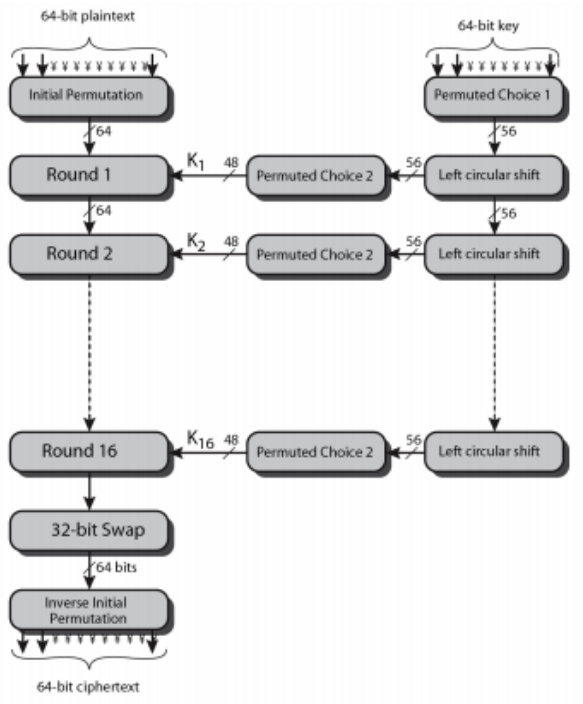
攻击 DES 的主要形式被称为蛮力的或彻底密钥搜索，即重复尝试各种密钥直到有一个符合为止。如果 DES 使用 56 位的密钥，则可能的密钥数量是 2 的 56 次方个。随着计算机系统能力的不断发展，DES 的安全性比它刚出现时会弱得多，然而从非关键性质的实际出发，仍可

以认为它是足够的。不过 ，DES 现在仅用于旧系统的鉴定，而更多地选择新的加密标准 — 高级加密标准（Advanced Encryption Standard，AES）。

DES 的常见变体是三重 DES，使用 168 位的密钥对资料进行三次加密的一种机制；它通常（但非始终）提供极其强大的安全性。如果三个 56 位的子元素都相同，则三重 DES 向后兼容 DES。

IBM 曾对 DES 拥有几年的专利权，但是在 1983 年已到期，并且处于公有范围中，允许在特定条件下可以免除专利使用费而使用。

DES 加密框图如下：



而由于 DES 是 Feistel 结构的，Feistel 密码的解密算法与加密算法是相同的，只是子密钥的使用次序相反。

详细实验原理：

1、DES 加密

DES 算法为密码体制中的对称密码体制，整个加密体质如图 3.5。对于任意加密方案，总有两个输入：明文和密钥。DES 的明文为 64 位，密钥长为 56 位。

从图中左半部分，可见明文的处理经过了三个阶段。首先，64 位的明文经过初始置换（IP）而被重新排列。然后进行 16 轮相同函数的作用，每轮作用都有置换和代替。最后一轮迭代的输出有 64 位，它是输入明文和密钥的函数。其左半部分和有伴部分互换产生预输出。最后预输出再被与初始置换（IP）互逆的置换（IP^{-1}）产生 64 位的秘闻。除了初始和末位的置换，DES 的结构与 Feistel 密码结构完全相同。

图 3.5 的右半部分给出了使用 56 位密钥的过程。首先，密钥经过一个置换后，在经过循环左移和一个置换分别得到各轮的子密钥 Ki 用于各轮的迭代。每轮的置换函数都一样，但是由于密钥的循环移位使得各轮子密钥互不相同。

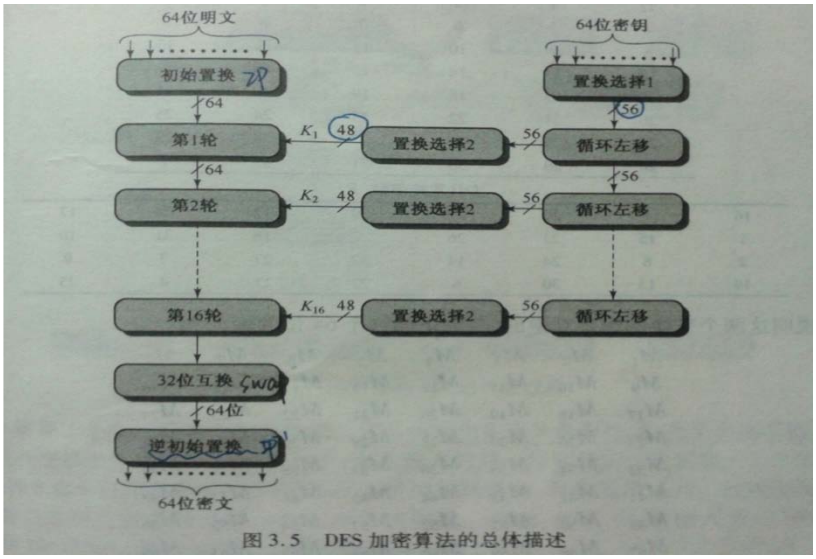


图 3.5　DES 加密算法的总体描述

（1）初始置换

表 3.2（a）和表 3.2（b）分别定义了初始置换及其逆置换，其解释如下。表的输入标记为从 1 到 64 位，共 64 位。置换表中 64 个元素代表从 1 到 64 这些数的一个置换。置换表中的每个元素表明了某个输入位在 64 位输出中的位置。

表3.2 DES 的置换表

（a） 初始置换（IP）

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

（b）逆初始置换（IP^{-1}）

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

（c） 扩展置换（E）

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

（d） 置换函数（P）

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

为了说明这两个变换的确是互逆的，考虑下面这个 64 位的输入 M：

$$\begin{matrix} M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & M_7 & M_8 \\ M_9 & M_{10} & M_{11} & M_{12} & M_{13} & M_{14} & M_{15} & M_{16} \\ M_{17} & M_{18} & M_{19} & M_{20} & M_{21} & M_{22} & M_{23} & M_{24} \\ M_{25} & M_{26} & M_{27} & M_{28} & M_{29} & M_{30} & M_{31} & M_{32} \\ M_{33} & M_{34} & M_{35} & M_{36} & M_{37} & M_{38} & M_{39} & M_{40} \\ M_{41} & M_{42} & M_{43} & M_{44} & M_{45} & M_{46} & M_{47} & M_{48} \\ M_{49} & M_{50} & M_{51} & M_{52} & M_{53} & M_{54} & M_{55} & M_{56} \\ M_{57} & M_{58} & M_{59} & M_{60} & M_{61} & M_{62} & M_{63} & M_{64} \end{matrix}$$

这里 Mi 都是二进制数。经过置换 X=IP（M）之后，得到的 X 是：

M_{58}	M_{50}	M_{42}	M_{34}	M_{26}	M_{18}	M_{10}	M_2
M_{60}	M_{52}	M_{44}	M_{36}	M_{28}	M_{20}	M_{12}	M_4
M_{62}	M_{54}	M_{46}	M_{38}	M_{30}	M_{22}	M_{14}	M_6
M_{64}	M_{56}	M_{48}	M_{40}	M_{32}	M_{24}	M_{16}	M_8
M_{57}	M_{49}	M_{41}	M_{33}	M_{25}	M_{17}	M_9	M_1
M_{59}	M_{51}	M_{43}	M_{35}	M_{27}	M_{19}	M_{11}	M_3
M_{61}	M_{53}	M_{45}	M_{37}	M_{29}	M_{21}	M_{13}	M_5
M_{63}	M_{55}	M_{47}	M_{39}	M_{31}	M_{23}	M_{15}	M_7

如果我们再对它作用逆置换 $Y=IP^{-1}（X）=IP^{-1}（IP（M））$，就会恢复出 M。

（2）每轮变换的详细过程

图 3.6 给出了一轮的内部结构。我们同样先看图的左半部分。64 位中间数据的左右两部分作为独立的 32 位数据，分别即为 L 和 R。在经典 Feistel 密码中，没轮变换的整个过程可以写为下面的公式：

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

轮密钥 Ki 长 48 位，R 是 32 位。首先将 R 用表 3.2（c）定义的置换扩展为 48 位，其中有 16 位是重复的。这 48 位于 Ki 异或，所得结果再用一个代替函数作用产生 32 位的输出，再用表 3.2（d）定义的置换进行作用后输出。

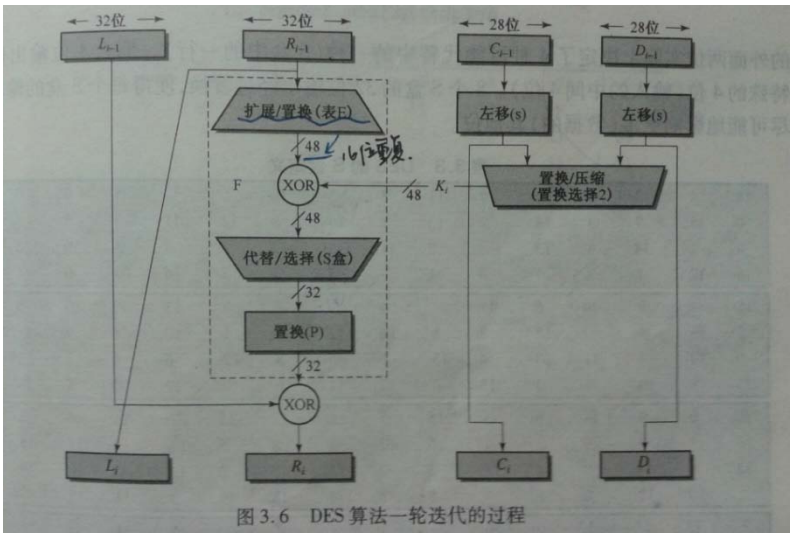


图 3.6　DES 算法一轮迭代的过程

图 3.7 解释了 S 盒在函数 F 中的作用。代替函数由 8 个 S 盒来组成，每个 S 盒都输入 6 位，

输出 4 位。这些变换参见表 3.3，其解释如下：盒 Si 输入的第一位和最后一位组成一个 2 位的二进制数，用来选择 S 盒 4 行代替值中的一行，中间 4 位用来选择 16 列中的某一列。行列交叉处的十进制值转换为二进制之后可能得到输出的 4 为二进制数。

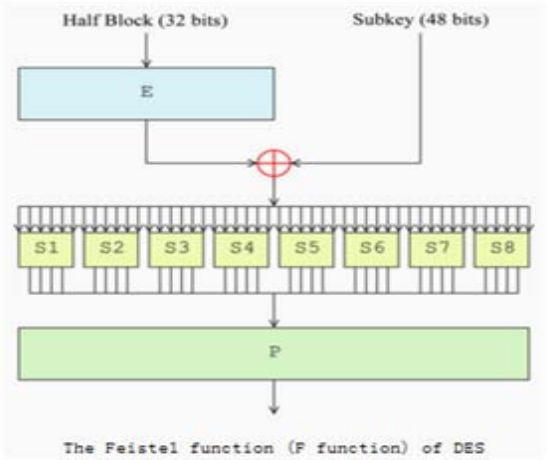


图 3.7

S 盒的每行都定义了一个普通的可逆代替。图 3.2 中显示了盒 S1 第 0 行的代替关系。

表3.3 DES 的 S 盒定义

S1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S2																
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3																
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S4																
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9

2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S5																
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6																
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7																
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S8																
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

（3）密钥产生

回到图 3.5 和图 3.6 中，我们看到算法输入了 64 位的密钥，密钥各位分别标记为 1 到 64。如表 3.4（a）中没有阴影的部分，也就是每行第 8 个位被忽略。首先用标记为置换选择 1 的表作用。所得 56 为密钥分别为两个 28 位数据 C_0 和 D_0。每轮迭代中，C_{i-1} 和 D_{i-1} 分别循环左移一位或两位，具体左移位数参见表 3.4（d）。移位后的值作为下一轮的输入。它们同时也作为置换选择 2 的输入，将产生一个 48 位的输出作为函数 F（R_{i-1}，K_i）的输入。

表 3.4 DES 密钥的使用

（a）输入密钥

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	53	54	55	56
57	58	59	60	61	62	63	64

（b）置换选择1（PC-1）

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

（c）置换选择 2（PC-2）

14	17	11	24	1	5	14
3	28	15	6	21	10	3
23	19	12	4	26	8	23
16	7	27	20	13	2	16
41	52	31	37	47	55	41
30	40	51	45	33	48	30
44	49	39	56	34	53	44
46	42	50	36	29	32	46

（d）左移次数的确定

迭代轮数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
移位次数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

2、DES 解密

Feistel 密码的解密算法与加密算法是相同的，只是子密钥的使用次序相反。

3、消息认证码

消息认证码（Cipher-based MAC），又称密码校验和或者 MAC，也是一种认证技术，它利用密钥来生成一个固定长度的短数据块，并将该数据块附加在消息之后。在这种方法中，假定通信双方比如 A 和 B，共享密钥 K。若 A 向 B 发送消息时，则 A 计算 MAC，它是消息和密钥的函数，即

MAC=C（K，M）

其中，M 是输入消息，C 是 MAC 函数，K 是共享的密钥，MAC 是消息认证码。

消息和 MAC 一起呗发送给接收方。接收方对收到的消息用相同的密钥 K 进行相同的计算，得出新的 MAC，并将接收到的 MAC 与其计算出的 MAC 进行比较，如果我们假定只有收发双方知道该密钥，那么若接收到的 MAC 与计算得出的 MAC 相等，则

（1）接收方可以相信消息未被修改。

（2）接收方可以相信消息来自于真正的发送方。

（3）如果消息中含有序列号，那么接收方可以相信消息顺序是正确的。

MAC 函数与加密类似。其区别之一是，MAC 算法不要求可逆性，而加密算法必须是可逆的。一般而言，MAC 函数是多对一函数，其定义域由任意长的消息组成，而值域由所有可能的 MAC 和密钥组成。

4、CMAC

基于密码分组的消息认证码可以用来处理任意长度的消息。首先，当消息长度是分组长度 b 的 n 倍时，对于 DES，b=64，这个消息被分为 n 组（M1，M2...Mn）。算法使用 64 位加密密钥 K 和 n 位的常数 K1。CMAC 按如下方式计算：

$$C_1 = E(K, M_1)$$

$$C_2 = E(K, [M_2 \oplus C_1])$$

$$C_3 = E(K, [M_3 \oplus C_2])$$

.

.

.

$$C_n = E(K, [M_n \oplus C_{n-1} \oplus K_1])$$

$$T = MSB_{Tlen}(C_n)$$

其中，T 为消息认证码，也称为 tag，Tlen 是 T 的位长度，MSB_s（X）是位串的 X 最左边的 s 位。

如果消息不是密文分组长度的整数倍，则最后分组的右边填充一个 1 和若干 0 使得最后的分组长度为 b。除了使用一个不同的 n 位密钥 K2 代替 K1 外，和前面所述的一样进行 CMAC 运算。

两个 n 位的密钥由 64 位的加密密钥按如下方式导出：

$$L = E(K, 0^n)$$

$$K_1 = L \bullet x$$

$$K_2 = L \bullet x^2 = (L \bullet x) \bullet x$$

其中乘法（ • ）在域 GF（2^n）内进行，x 和 x^2 是域 GF（2^n）的一次和二次多项式。因此 x 的二元表示为 n-2 个 0，后跟 10，而 x^2 的二元表示是 n-3 个 0，后跟 100。有限域由不可约多项式定义，该多项式是那些具有极小非零项的多项式集合里按字典排序第一的那个多项式。对于已获得批准的两个分组长度，多项式是 $x^{64}+x^4+x^3+x+1$ 以及 $x^{128}+x^7+x^2+x+1$。

为了生成 K1和 K2，分组密码应用到一个全0分组上。第一个子密钥从所得密文导出，即先左移一位，并且根据条件和一个常数进行异或运算得到，其中常数依赖于分组的大小。第二个子密钥采用相同的方式从第一个子密钥导出。

具体算法如下：

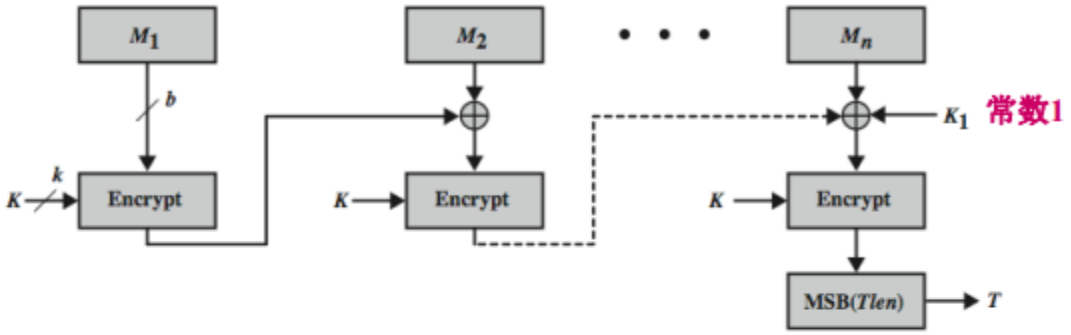
1．计算临时值 $K_0 = E_{K(0)}$。

2．如果 MSB（K_0）= 0，　则 K1 = $K_{0«1}$，否则 K 1 =（$K_{0«1}$）　⊕ C，其中 C 是有一定的常数，只取决于对 b。（具体而言，C 是领先的字典第一个不可约的数量降到最低的程度节点 b 的二元多项式系数。）

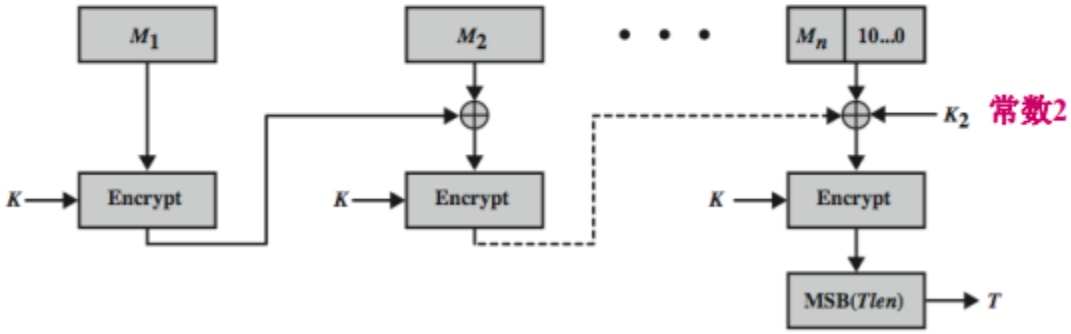
3．如果 MSB（K_1）= 0，　则 $K_2 = K_{1«1}$，否则 K 2 =（$K 1_{«1}$）⊕C。

4．返回键（$K_{0,1}$）的 MAC 生成过程。

算法框图如下：



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

三、主要仪器设备

注意：我的程序运行环境是 Code::Blocks10.05，在 vc6.0 下可能会报错！！Code::Blocks 是一个开放源码的全功能的跨平台 C/C++集成开发环境。 Code::Blocks 是开放源码软件。Code::Blocks 由纯粹的 C++语言开发完成，它使用了著名的图形界面库 wxWidgets(2.6.2 unicode)版。

如果老师要调试我的程序，可以到http://code-blocks.softonic.cn/download#downloading下载 Code::Blocks10.05 版本。Code::Blocks的安装程序也不大，安装也不麻烦！！

四、代码清单及代码分析：

用 C++实现的 DES，代码清单如下：

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<iostream>
using namespace std;
void Decode();
void Encode();
void keyBuild(int *keychar);                                    //create key array
void keyCreate(int *midkey2,int movebit,int i);          //call by keyBuild
void F(int *rData,int *key);                                    //F function
void Expand(int *rData,int *rDataP);                        //Expand function
void ExchangeS(int *rDataP,int *rData);                  //S-diagram change
void ExchangeP(int *rData);                                    //P change
void FillBin(int *rData,int n,int s);                        // data to binary;call by
S-Diagram change function
void InitialPermutation(int *plaintext_64bit);
void print_byte(char *text_16byte);
void Hex2Bit(char *text_byte,int *text_bit,int n);      //change into bit,n为bit的位数
void Bit2Hex(char *text_byte,int* text_bit,int n);      //change into hex,n为bit的位数
void RoundEncode(int n);
int IP1[]={58, 50, 42, 34, 26, 18, 10, 2,
           60, 52, 44, 36, 28, 20, 12, 4,      //initial change
           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,
 };
int IP2[]={40, 8, 48, 16, 56, 24, 64, 32,
           39, 7, 47, 15, 55, 23, 63, 31,      //opp initial change
           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
 };
int
s[][4][16]={{
      //S-diagram array
                {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}
```

```
                    },
                    {
                     {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}
                    },
                    {
                     {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}
                    },
                    {
                     {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}
                    },
                    {
                     {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}
                    },
                    {
                     {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}
                    },
                    {
                     {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}
                    },
                    {
                     {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}
                    }
};
int Ex[48]={    32,1,2,3,4,5,
```

```
//Expand array
                    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
 };
int P[32]={16,7,20,21,29,12,28,17,
//P-change
                1,15,23,26,5,18,31,10,
                2,8,24,14,32,27,3,9,
                19,13,30,6,22,11,4,25
 };
int PC1[56]={57,49,41,33,25,17,9,
 //PC-1 in keyBuild
                    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
 };
int PC2[48]={14,17,11,24,1,5,3,28,
//PC-2 in keyBuild
                    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
 };

int LSi[16]={1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

int subkey_bit[16][48];
char subkey_byte[16][16];
char str[8];    //明文
char plaintext_16byte[16];
int plaintext_64bit[64];
char ciphertext_16byte[16];
int ciphertext_64bit[64];
char key_16byte[16];
int key_64bit[64];
```

```
int mode;   //来区分是加密还是解密
int main()                          //main function
{
	Encode();
	system("pause");
	Decode();
	system("pause");
	return 0;
}
void InitialPermutation(int *plaintext_64bit)
{
	int temptext[64];
	for(int i=0;i<64;i++)
		temptext[i]=plaintext_64bit[i];
	for(int i=0;i<64;i++)
		temptext[i]=plaintext_64bit[IP1[i]-1];
	for(int i=0;i<64;i++)
		ciphertext_64bit[i]=temptext[i];
	Bit2Hex(ciphertext_16byte,temptext,64);
	cout<<"初始置换后：  "<<endl;
	print_byte(ciphertext_16byte);
}

void print_byte(char *text_16byte)
{
	cout<<"       L[0]=";
	for(int i=0;i<16;i++)
	{
		cout<<text_16byte[i];
		if(i==7)
			cout<<"      R[0]=";
	}
	cout<<endl;
}

void Hex2Bit(char *text_byte,int *text_bit,int n)       //change into bit,n为bit的位数
{
	int length=n/4;
	int temptext[length];
	for(int i=0;i<length;i++)
	{
		if(text_byte[i]>='a'&&text_byte[i]<='f')
			temptext[i]=text_byte[i]-'a'+10;
		else if(text_byte[i]>='0'&&text_byte[i]<='9')
			temptext[i]=text_byte[i]-'0';
```

```
    }
    for(int i=0;i<length;i++)
    {
        for(int j=3;j>=0;j--)
        {
            text_bit[i*4+j]=temptext[i]%2;
            temptext[i]/=2;
        }
    }
}
void Bit2Hex(char *text_byte,int* text_bit,int n)       //change into hex,n为bit的位数
{
    int length=n/4;
    int temptext[length];
    for(int i=0;i<length;i++)
    {
        temptext[i]=text_bit[i*4+0]*8;
        temptext[i]+=text_bit[i*4+1]*4;
        temptext[i]+=text_bit[i*4+2]*2;
        temptext[i]+=text_bit[i*4+3]*1;
    }
    for(int i=0;i<length;i++)
    {
        if(temptext[i]>=0&&temptext[i]<=9)
            text_byte[i]=temptext[i]+'0';
        else if(temptext[i]>=10&&temptext[i]<=15)
            text_byte[i]=(temptext[i]-10)+'a';
    }
}

void keyBuild(int *keychar)     //create key array
{
    int i,j,k;
    int movebit[]={1,1,2,2,2,2,2,2,
                   1,2,2,2,2,2,2,1};
    int midkey2[56];
    int midkey[64];
    int key[56];
    for(k=0;k<56;k++)
    {
       midkey2[k]=keychar[PC1[k]-1];
    }
    printf("\n");
    for(i=0;i<16;i++)                       //密钥按照移位表循环左移,i代表轮数
    {
```

```
        keyCreate(midkey2,movebit[i],i);
    }
}
// midkey2是pc-1置换后的56bit密钥，movebit为左移的位数，n为第几轮
void keyCreate(int *midkey2,int movebit,int n)   //create key
{
    int i,temp[4];
    temp[0]=midkey2[0];
    temp[1]=midkey2[1];
    temp[2]=midkey2[28];
    temp[3]=midkey2[29];
   if(movebit==2)          //左移两位的算法
   {
        for(i=0;i<26;i++)
        {
          midkey2[i]=midkey2[i+2];
          midkey2[i+28]=midkey2[i+30];
        }
        midkey2[26]=temp[0];
        midkey2[27]=temp[1];
        midkey2[54]=temp[2];
        midkey2[55]=temp[3];
   }
     else                              //左移一位的算法
     { for(i=0;i<27;i++)
     {
         midkey2[i]=midkey2[i+1];
         midkey2[i+28]=midkey2[i+29];
        }
        midkey2[27]=temp[0];midkey2[55]=temp[2];
     }
     //pc-2置换后变成48位
     printf("第%d次pc-2置换子密钥结果(48bit): subkey[%d] = ",n+1,n+1);
    for(i=0;i<48;i++)
    {
      subkey_bit[n][i]=midkey2[PC2[i]-1];
    }
    Bit2Hex(subkey_byte[n],subkey_bit[n],48);
    cout<<subkey_byte[n];
    cout<<endl;
}


void RoundEncode(int n)   //第n轮加密
```

```
{
    int lData[32],rData[32];
    int rDataE[48];
    for(int i=0;i<32;i++)
        lData[i]=ciphertext_64bit[i];
    for(int i=0;i<32;i++)
        rData[i]=ciphertext_64bit[i+32];
    if(mode==1)
        cout<<"*******第"<<n+1<<"轮加密*******";
    else if(mode==2)
        cout<<"*******第"<<16-n<<"轮解密*******";
    Expand(rData,rDataE);
    for(int i=0;i<48;i++)        //取异或
        rDataE[i]=rDataE[i]^subkey_bit[n][i];
    ExchangeS(rDataE,rData);   //S盒变换
    ExchangeP(rData);            //P置换
    for(int i=0;i<32;i++)        //取异或
        rData[i]=rData[i]^lData[i];
    for(int i=0;i<32;i++)
        ciphertext_64bit[i]=ciphertext_64bit[i+32];
    for(int i=0;i<32;i++)
        ciphertext_64bit[i+32]=rData[i];
}
void F(int *rData,int *key)        //F function
{
    int i,rDataP[48];
    Expand(rData,rDataP);      //将明文右半部分进行E扩展，扩展成48位的A
    cout<<endl<<"E扩展后与48位密钥异或"<<endl;
    for(i=0;i<48;i++)
    {
      rDataP[i]=rDataP[i]^key[i];      //将进行完E扩展的A与key1异或
      cout<<rDataP[i];
    }

    ExchangeS(rDataP,rData);   //S盒变换
    ExchangeP(rData);
}
void Expand(int *rData,int *rDataP) //E扩展　将明文右半部分进行E扩展，扩展成48位
{
    int i;
    char temp[12];
    cout<<endl<<"E扩展"<<endl<<"　　二进制：";
    for(i=0;i<48;i++)
    {
      rDataP[i]=rData[Ex[i]-1];
```

```
        cout<<rDataP[i];
    }
    cout<<endl<<"      十六进制：";
    Bit2Hex(temp,rDataP,48);
    for(int i=0;i<12;i++)
        cout<<temp[i];
}
void ExchangeS(int *rDataP,int *rData){                //S盒变换
    int i,n,linex,liney;
    linex=liney=0;
    for(i=0;i<48;i+=6)
    {
        n=i/6; //printf("%10d\n",(rDataP[i]<<1));
        linex=(rDataP[i]<<1)+rDataP[i+5];
        liney=(rDataP[i+1]<<3)+(rDataP[i+2]<<2)+(rDataP[i+3]<<1)+rDataP[i+4];

        FillBin(rData,n,s[n][linex][liney]);

    }
    cout<<endl<<"S-盒运算"<<endl;
    cout<<"      二进制:";
    for(i=0;i<32;i++)
    {
        printf("%d",rData[i]);
    }
    cout<<endl<<"      十六进制：";
    char temp[8];
    Bit2Hex(temp,rData,32);
    for(int i=0;i<8;i++)
        cout<<temp[i];
}
void FillBin(int *rData,int n,int s)     // data to binary;call by S-Diagram change function
{
    int temp[4],i;
    for(i=0;i<4;i++)
    {
        temp[i]=s%2;
        s=s/2;
     }
    for(i=0;i<4;i++)
    {
        rData[n*4+i]=temp[3-i];
    }
}
void ExchangeP(int *rData)
```

```
{                                       //P change
    int i,temp[32];
    for(i=0;i<32;i++)
       temp[i]=rData[i];
    cout<<endl<<"p-置换"<<endl<<"      二进制：";
    for(i=0;i<32;i++)
    {
       rData[i]=temp[P[i]-1];
       printf("%d",rData[i]);
    }
    cout<<endl<<"      十六进制：";
    char temp2[8];
    Bit2Hex(temp2,rData,32);
    for(int i=0;i<8;i++)
        cout<<temp2[i];

}
void Encode()
{
    mode=1;//表示加密
    cout<<"请输入16个要加密的字符："<<endl;
    for(int i=0;i<16;i++)
        cin>>plaintext_16byte[i];
    Hex2Bit(plaintext_16byte,plaintext_64bit,64);
    cout<<"输入明文的二进制为："<<endl;
    for(int i=0;i<64;i++)
        cout<<plaintext_64bit[i];
    cout<<endl;

    cout<<endl<<"请输入密钥(16个字符):"<<endl;
    for(int i=0;i<16;i++)
        cin>>key_16byte[i];
    Hex2Bit(key_16byte,key_64bit,64);
    cout<<"输入密钥二进制为："<<endl;
    for(int i=0;i<64;i++)
        cout<<key_64bit[i];
    cout<<endl;
    InitialPermutation(plaintext_64bit);

    Hex2Bit(key_16byte,key_64bit,64);
    keyBuild(key_64bit);

    for(int i=0;i<16;i++)
    {
        RoundEncode(i);
```

```
            cout<<endl;
            Bit2Hex(ciphertext_16byte,ciphertext_64bit,64);
            cout<<"轮输出"<<endl;
            int round=i+1;
            cout<<"       L["<<round<<"]=";
            for(int i=0;i<16;i++)
            {
                  if(i==8)
                        cout<<"       R["<<round<<"]=";
                  cout<<ciphertext_16byte[i];
            }
            cout<<endl;
      }
      int temp[64];     //逆初始置换
      for(int i=0;i<64;i++)
            temp[i]=ciphertext_64bit[i];
      for(int i=0;i<32;i++)
            ciphertext_64bit[i]=temp[i+32];
      for(int i=0;i<32;i++)
            ciphertext_64bit[i+32]=temp[i];

      for(int i=0;i<64;i++)
            temp[i]=ciphertext_64bit[i];
      for(int i=0;i<64;i++)
            ciphertext_64bit[i]=temp[ IP2[i]-1    ];
      Bit2Hex(ciphertext_16byte,ciphertext_64bit,64);

      cout<<"加密后密文为："<<ciphertext_16byte;
}
void Decode()
{
      mode=2;//表示解密
      cout<<endl<<"请输入解密密钥(16个字符):"<<endl;
      for(int i=0;i<16;i++)
            cin>>key_16byte[i];
      Hex2Bit(key_16byte,key_64bit,64);
      cout<<"输入密钥二进制为："<<endl;
      for(int i=0;i<64;i++)
            cout<<key_64bit[i];
      cout<<endl;
      InitialPermutation(ciphertext_64bit);
      for(int i=15;i>=0;i--)
      {
            RoundEncode(i);
            cout<<endl;
```

```
            Bit2Hex(ciphertext_16byte,ciphertext_64bit,64);
            cout<<"轮输出"<<endl;
            int round=16-i;
            cout<<"       L["<<round<<"]=";
            for(int i=0;i<16;i++)
            {
                  if(i==8)
                        cout<<"       R["<<round<<"]=";
                  cout<<ciphertext_16byte[i];
            }
            cout<<endl;
      }

      int temp[64];     //逆置换
      for(int i=0;i<64;i++)
            temp[i]=ciphertext_64bit[i];
      for(int i=0;i<32;i++)
            ciphertext_64bit[i]=temp[i+32];
      for(int i=0;i<32;i++)
            ciphertext_64bit[i+32]=temp[i];

      for(int i=0;i<64;i++)
            temp[i]=ciphertext_64bit[i];
      for(int i=0;i<64;i++)
            ciphertext_64bit[i]=temp[ IP2[i]-1    ];
      Bit2Hex(ciphertext_16byte,ciphertext_64bit,64);

      cout<<"解密后明文为："<<ciphertext_16byte;
}
```

为了验证上述代码的正确性，我从网上找了一组DES的数据：

输入数据：D=74 65 73 74 64 61 74 61，

输入密钥：K=6D 79 64 65 73 6B 65 79

生成的 16 组密钥如下：

sub-key[1]=F0 BE 6E B3 88 28
sub-key[2]=E0 BE F6 03 46 5E
sub-key[3]=F4 F6 76 9D 91 80
sub-key[4]=E6 D7 72 80 46 65
sub-key[5]=EE D3 77 5A AA 84
sub-key[6]=AF D3 5B B0 45 99
sub-key[7]=2F 53 FB 0B 32 03
sub-key[8]=BF 59 D9 F6 61 20
sub-key[9]=1F 59 DB 17 C8 07
sub-key[10]=3F 69 DD 46 05 D0
sub-key[11]=1F 6D 8D 89 A1 4D

sub-key[12]=5B 2D BD 62 D6 80
sub-key[13]=DD AC AD 58 05 2F
sub-key[14]=D3 AE AE 8E 58 88
sub-key[15]=F8 BE A6 40 73 71
sub-key[16]=F1 BE 26 EC C8 11

先进行初始置换，得到D'=FF 4D 5B A6 00 FF 00 04，分成左右两组，得到**初始值** L[0]=FF 4D 5B A6 和 R[0]=00 FF 00 04。

右半部分的数据经过扩展，数据如下：

R'[1]=00 17 FE 80 00 08
R'[2]=0F 02 0F DA 40 08
R'[3]=15 97 F0 2A AB A0
R'[4]=C5 01 F9 7F 3D F7
R'[5]=E0 94 02 B5 7D 53
R'[6]=AA 95 56 90 D5 5A
R'[7]=D5 15 0E A0 40 0B
R'[8]=5A 7C FA AA 7C A1
R'[9]=3F 2A 0C 0F 3E A8
R'[10]=EF 7C FA 95 97 5B
R'[11]=1F 15 02 A0 16 FC
R'[12]=80 03 FD 60 C2 F2
R'[13]=DF 97 F7 EA EB A3
R'[14]=95 56 00 3F BC F2
R'[15]=25 82 59 75 3D A8
R'[16]=85 E9 0B E5 56 FA

R'与 sub-key 异或操作后，经过 S 盒选择，生成数据如下：

RS[1]=5B 91 B1 17
RS[2]=05 FC 50 2E
RS[3]=3D E3 2E C7
RS[4]=2E 4F 37 E9
RS[5]=F7 F5 48 6B
RS[6]=02 B0 77 0F
RS[7]=07 B5 D7 F6
RS[8]=A7 CF AA 81
RS[9]=27 AC 15 AD
RS[10]=93 C6 C6 03
RS[11]=E7 63 AB 1F
RS[12]=78 B7 24 E6
RS[13]=E8 8C 73 5B
RS[14]=A2 6D 7B E3
RS[15]=E0 FE D3 20
RS[16]=34 2D C0 2A

RS 通过 P 盒置换，生成数据如下：

RP[1]=E7 0A E9 A2
RP[2]=2C 07 55 74
RP[3]=90 7B 4B BF

RP[4]=E8 79 3E D9
RP[5]=DC 97 DF 16
RP[6]=6C 26 29 AC
RP[7]=E7 36 75 3D
RP[8]=D1 E1 5B D1
RP[9]=68 00 7F 7D
RP[10]=41 E7 59 0A
RP[11]=DB E3 EA B4
RP[12]=84 5A 97 AF
RP[13]=2A BE BB C0
RP[14]=F0 B7 3E C5
RP[15]=25 E5 B7 44
RP[16]=89 06 16 56

每次循环，左右两部分数据分别为：

L[1]=00 FF 00 04 R[1]=18 47 B2 04
L[2]=18 47 B2 04 R[2]=2C F8 55 70
L[3]=2C F8 55 70 R[3]=88 3C F9 BB
L[4]=88 3C F9 BB R[4]=C4 81 6B A9
L[5]=C4 81 6B A9 R[5]=54 AB 26 AD
L[6]=54 AB 26 AD R[6]=A8 A7 42 05
L[7]=A8 A7 42 05 R[7]=B3 9D 53 90
L[8]=B3 9D 53 90 R[8]=79 46 19 D4
L[9]=79 46 19 D4 R[9]=DB 9D 2C ED
L[10]=DB 9D 2C ED R[10]=38 A1 40 DE
L[11]=38 A1 40 DE R[11]=00 7E C6 59
L[12]=00 7E C6 59 R[12]=BC FB D7 71
L[13]=BC FB D7 71 R[13]=2A C0 7D 99
L[14]=2A C0 7D 99 R[14]=4C 4C E9 B4
L[15]=4C 4C E9 B4 R[15]=0F 25 CA DD
L[16]=0F 25 CA DD R[16]=C5 4A FF E2

最后一次置换前的数据 D-1=C5 4A FF E2 0F 25 CA DD，置换后生成最终数据为 DE=E6 9D E6 9E 06 25 5F 4F。

运行程序结果如下：

```
请输入16个要加密的字符：
7465737464617461
输入明文的二进制为：
0111010001100101011100110111010001100100011000010111010001100001

请输入密钥<16个字符>：
6d796465736b6579
输入密钥二进制为：
0110110101111001011001000110010101110011011010110110010101111001
初始置换后：
    L[0]=ff4d5ba6    R[0]=00ff0004
```

```
第1次pc-2置换子密钥结果(48bit): subkey[1] = f0be6eb38828
第2次pc-2置换子密钥结果(48bit): subkey[2] = e0bef603465e
第3次pc-2置换子密钥结果(48bit): subkey[3] = f4f6769d9180
第4次pc-2置换子密钥结果(48bit): subkey[4] = e6d772804665
第5次pc-2置换子密钥结果(48bit): subkey[5] = eed3775aaa84
第6次pc-2置换子密钥结果(48bit): subkey[6] = afd35bb04599
第7次pc-2置换子密钥结果(48bit): subkey[7] = 2f53fb0b3203
第8次pc-2置换子密钥结果(48bit): subkey[8] = bf59d9f66120
第9次pc-2置换子密钥结果(48bit): subkey[9] = 1f59db17c807
第10次pc-2置换子密钥结果(48bit): subkey[10] = 3f69dd4605d0
第11次pc-2置换子密钥结果(48bit): subkey[11] = 1f6d8d89a14d
第12次pc-2置换子密钥结果(48bit): subkey[12] = 5b2dbd62d680
第13次pc-2置换子密钥结果(48bit): subkey[13] = ddacad58052f
第14次pc-2置换子密钥结果(48bit): subkey[14] = d3aeae8e5888
第15次pc-2置换子密钥结果(48bit): subkey[15] = f8bea6407371
第16次pc-2置换子密钥结果(48bit): subkey[16] = f1be26ecc811
```

```
*******第1轮加密*******
E扩展
    二进制：000000000000101111111111010000000000000000001000
    十六进制：0017fe800008
S-盒运算
    二进制:01011011100100011011000100010111
    十六进制：5b91b117
p-置换
    二进制：11100111000010101111010011010010
    十六进制：e70ae9a2
轮输出
    L[1]=00ff0004    R[1]=1847b204
```

```
*******第2轮加密*******
E扩展
    二进制：000011111000000100000111111011010010000000001000
    十六进制：0f020fda4008
S-盒运算
    二进制:000000101111111000101000000101110
    十六进制：05fc502e
p-置换
    二进制：00101100000001110101010101110100
    十六进制：2c075574
轮输出
    L[2]=1847b204    R[2]=2cf85570
```

```
*******第3轮加密*******
E扩展
    二进制：000101011001011111110000001010101010111010000
    十六进制：1597f02aaba0
S-盒运算
    二进制:001111011110001100101111011000111
    十六进制：3de32ec7
p-置换
    二进制：10010000011110110100101110111111
    十六进制：907b4bbf
轮输出
    L[3]=2cf85570    R[3]=883cf9bb
```

```
*******第4轮加密*******
E扩展
    二进制：110001010000000111111001011111110011111011111 0111
    十六进制：c501f97f3df7
S-盒运算
    二进制:001011100100111100110111111101001
    十六进制：2e4f37e9
p-置换
    二进制：11101000011110010011111011011001
    十六进制：e8793ed9
轮输出
    L[4]=883cf9bb    R[4]=c4816ba9
```

```
*******第5轮加密*******
E扩展
    二进制：111000001001010000000010101101010111110101010011
    十六进制：e09402b57d53
S-盒运算
    二进制:111101111111010101001000011010 11
    十六进制：f7f5486b
p-置换
    二进制：11011100100101111101111100010110
    十六进制：dc97df16
轮输出
    L[5]=c4816ba9    R[5]=54ab26ad
```

```
*******第6轮加密*******
E扩展
    二进制：101010101001010101010110100100001101010101011010
    十六进制：aa955690d55a
S-盒运算
    二进制:000000101011000001110111100001111
    十六进制：02b0770f
p-置换
    二进制：01101100001001100010100110101100
    十六进制：6c2629ac
轮输出
    L[6]=54ab26ad    R[6]=a8a74205
```

```
*******第7轮加密*******
E扩展
    二进制：110101010001010100001110101000000100000000001011
    十六进制：d5150ea0400b
S-盒运算
    二进制:000001111011010111010111111110110
    十六进制：07b5d7f6
p-置换
    二进制：11100111001101100111010100111101
    十六进制：e736753d
轮输出
    L[7]=a8a74205    R[7]=b39d5390
```

```
*******第8轮加密*******
E扩展
    二进制：010110100111111001111101010101010011111100101000001
    十六进制：5a7cfaaa7ca1
S-盒运算
    二进制:101001111100111111010101010000001
    十六进制：a7cfaa81
p-置换
    二进制：11010001111000010101101111010001
    十六进制：d1e15bd1
轮输出
    L[8]=b39d5390    R[8]=794619d4
```

```
*******第9轮加密*******
E扩展
    二进制：001111110010101000001100000011110011111010101000
    十六进制：3f2a0c0f3ea8
S-盒运算
    二进制:001001111010110000010101101101
    十六进制：27ac15ad
p-置换
    二进制：01101000000000000111111101111101
    十六进制：68007f7d
轮输出
    L[9]=794619d4    R[9]=db9d2ced
```

```
*******第10轮加密*******
E扩展
    二进制：111011110111110011111010100101011001011101011011
    十六进制：ef7cfa95975b
S-盒运算
    二进制:100100111100011011000110000000011
    十六进制：93c6c603
p-置换
    二进制：01000001111001110101100100001010
    十六进制：41e7590a
轮输出
    L[10]=db9d2ced    R[10]=38a140de
```

```
*******第11轮加密*******
E扩展
    二进制：000111110001010100000010101000000000101101111110 0
    十六进制：1f1502a016fc
S-盒运算
    二进制:111001110110001110101011000111111
    十六进制：e763ab1f
p-置换
    二进制：11011011111000111111010101011010 0
    十六进制：dbe3eab4
轮输出
    L[11]=38a140de    R[11]=007ec659
```

```
*******第12轮加密*******
E扩展
    二进制：100000000000001111111101011000001100001011110010
    十六进制：8003fd60c2f2
S-盒运算
    二进制:01111000101101110010010011100110
    十六进制：78b724e6
p-置换
    二进制：10000100010110101001011110101111
    十六进制：845a97af
轮输出
    L[12]=002ec659    R[12]=bcfbd771
```

```
*******第13轮加密*******
E扩展
    二进制：110111111001011111111011111101010111010111010100011
    十六进制：df97f7eaeba3
S-盒运算
    二进制:11101000100011000111001101011011
    十六进制：e88c735b
p-置换
    二进制：00101010101111101011101111000000
    十六进制：2abebbc0
轮输出
    L[13]=bcfbd771    R[13]=2ac07d99
```

```
*******第14轮加密*******
E扩展
    二进制：100101010101011000000000001111111011110011110010
    十六进制：9556003fbcf2
S-盒运算
    二进制:10100010011011010111101111100011
    十六进制：a26d7be3
p-置换
    二进制：11110000101101110011111011000101
    十六进制：f0b73ec5
轮输出
    L[14]=2ac07d99    R[14]=4c4ce9b4
```

```
*******第15轮加密*******
E扩展
    二进制：001001011000001001011001011101010011111011010100
    十六进制：258259753da8
S-盒运算
    二进制:11100000111111101101001100100000
    十六进制：e0fed320
p-置换
    二进制：00100101111001011011011101000100
    十六进制：25e5b744
轮输出
    L[15]=4c4ce9b4    R[15]=0f25cadd
```

```
*******第16轮加密*******
E扩展
    二进制：10000101111010010000101111100101010101011011111010
    十六进制：85e90be556fa
S-盒运算
    二进制:00110100001011011100000000101010
    十六进制：342dc02a
p-置换
    二进制：10001001000001100001011001010110
    十六进制：89061656
轮输出
    L[16]=0f25cadd    R[16]=c54affe2
加密后密文为：e69de69e06255f4f请按任意键继续. . .
```

可以看到加密后为 DE=E6 9D E6 9E 06 25 5F 4F，验证加密正确。接下来验证解密：

```
请输入解密密钥(16个字符):
6d796465736b6579
输入密钥二进制为:
0110110101111001011001000110010101110011011010110110010101111001
初始置换后:
    L[0]=c54affe2    R[0]=0f25cadd
```

```
*******第1轮解密*******
E扩展
    二进制：10000101111010010000101111100101010101011011111010
    十六进制：85e90be556fa
S-盒运算
    二进制:00110100001011011100000000101010
    十六进制：342dc02a
p-置换
    二进制：10001001000001100001011001010110
    十六进制：89061656
轮输出
    L[1]=0f25cadd    R[1]=4c4ce9b4
```

```
*******第2轮解密*******
E扩展
    二进制：001001011000001001011001011101010011111011010100
    十六进制：258259753da8
S-盒运算
    二进制:11100000111111101101001100100000
    十六进制：e0fed320
p-置换
    二进制：00100101111001011011011101000100
    十六进制：25e5b744
轮输出
    L[2]=4c4ce9b4    R[2]=2ac07d99
```

```
*******第3轮解密*******
E扩展
    二进制：100101010101011000000000000111111101111001111001O
    十六进制：9556003fbcf2
S-盒运算
    二进制:10100010011011010111101111100011
    十六进制：a26d7be3
p-置换
    二进制：11110000101101110011111011000101
    十六进制：f0b73ec5
轮输出
    L[3]=2ac07d99    R[3]=bcfbd771
```

```
*******第4轮解密*******
E扩展
    二进制：110111111001011111111011111101010111010111010100011
    十六进制：df97f7eaeba3
S-盒运算
    二进制:11101000100011000111001101011011
    十六进制：e88c735b
p-置换
    二进制：00101010101111101011101111000000
    十六进制：2abebbc0
轮输出
    L[4]=bcfbd771    R[4]=007ec659
```

```
*******第5轮解密*******
E扩展
    二进制：100000000000001111111101011000001100001011110010
    十六进制：8003fd60c2f2
S-盒运算
    二进制:01111000101101110010010011100110
    十六进制：78b724e6
p-置换
    二进制：10000100010110101001011110101111
    十六进制：845a97af
轮输出
    L[5]=007ec659    R[5]=38a140de
```

```
*******第6轮解密*******
E扩展
    二进制：000111111000101010000001010100000000010110111111100
    十六进制：1f1502a016fc
S-盒运算
    二进制:11100111011000111010101100011111
    十六进制：e763ab1f
p-置换
    二进制：11011011111000111111010101011010O
    十六进制：dbe3eab4
轮输出
    L[6]=38a140de    R[6]=db9d2ced
```

```
*******第7轮解密*******
E扩展
    二进制：111011110111110011111010100101011001011101011011
    十六进制：ef7cfa95975b
S-盒运算
    二进制:10010011111000110110001100000011
    十六进制：93c6c603
p-置换
    二进制：01000001111001110101100100001010
    十六进制：41e7590a
轮输出
    L[7]=db9d2ced    R[7]=794619d4
```

```
*******第8轮解密*******
E扩展
    二进制：001111110010101000001100000011110011111010101000
    十六进制：3f2a0c0f3ea8
S-盒运算
    二进制:001001111010110000010101101101
    十六进制：27ac15ad
p-置换
    二进制：01101000000000000111111101111101
    十六进制：68007f7d
轮输出
    L[8]=794619d4    R[8]=b39d5390
```

```
*******第9轮解密*******
E扩展
    二进制：010110100111111001111110101010101001111100101000001
    十六进制：5a7cfaaa7ca1
S-盒运算
    二进制:101001111100111111010101010000001
    十六进制：a7cfaa81
p-置换
    二进制：11010001111100001010110111101000l
    十六进制：d1e15bd1
轮输出
    L[9]=b39d5390    R[9]=a8a74205
```

```
*******第10轮解密*******
E扩展
    二进制：11010101000101010000111010100000010000000000001011
    十六进制：d5150ea0400b
S-盒运算
    二进制:000001111011010111010111111110110
    十六进制：07b5d7f6
p-置换
    二进制：11100111001101100111010100111101
    十六进制：e736753d
轮输出
    L[10]=a8a74205    R[10]=54ab26ad
```

```
*******第11轮解密*******
E扩展
    二进制：101010101001010101010110100100001101010101011010
    十六进制：aa955690d55a
S-盒运算
    二进制:00000010101100000111011100001111
    十六进制：02b0770f
p-置换
    二进制：01101100001001100010100110101100
    十六进制：6c2629ac
轮输出
    L[11]=54ab26ad    R[11]=c4816ba9
```

```
*******第12轮解密*******
E扩展
    二进制：111000001001010000000010101101010111110101010011
    十六进制：e09402b57d53
S-盒运算
    二进制:11110111111110101010010000l101011
    十六进制：f7f5486b
p-置换
    二进制：11011100100101111101111100010110
    十六进制：dc97df16
轮输出
    L[12]=c4816ba9    R[12]=883cf9bb
```

```
*******第13轮解密*******
E扩展
    二进制：110001010000000111111001011111110011110111110111
    十六进制：c501f97f3df7
S-盒运算
    二进制:00101110010011110011011111101001
    十六进制：2e4f37e9
p-置换
    二进制：11101000011110010011111011011001
    十六进制：e8793ed9
轮输出
    L[13]=883cf9bb    R[13]=2cf85570
```

```
*******第14轮解密*******
E扩展
    二进制：000101011001011111111000000101010101011110100000
    十六进制：1597f02aaba0
S-盒运算
    二进制:001111011110001100101110110001111
    十六进制：3de32ec7
p-置换
    二进制：10010000011110110100101110111111
    十六进制：907b4bbf
轮输出
    L[14]=2cf85570    R[14]=1847b204
```

```
*******第15轮解密*******
E扩展
    二进制：000011110000001000001111111011010010000000001000
    十六进制：0f020fda4008
S-盒运算
    二进制:000001011111111000101000000101110
    十六进制：05fc502e
p-置换
    二进制：00101100000001110101010101110100
    十六进制：2c075574
轮输出
    L[15]=1847b204    R[15]=00ff0004
```

```
*******第16轮解密*******
E扩展
    二进制：000000000000101111111111010000000000000000001000
    十六进制：0017fe800008
S-盒运算
    二进制:010110111001000110110001000010111
    十六进制：5b91b117
p-置换
    二进制：11100111000010101110100110100010
    十六进制：e70ae9a2
轮输出
    L[16]=00ff0004    R[16]=ff4d5ba6
解密后明文为：7465737464617461请按任意键继续. . .
```

可以看到解密后输出为 74 65 73 74 64 61 74 61，验证解密正确！

检验雪崩效应：

输入名为改为：746573746461746**2**（其中下划线处为改变的明文）

输入密钥不变：6d796465736b6579
此时加密的第二轮输出如下：

```
*******第2轮加密*******
E扩展
    二进制：100011110100001100001111111001010110101010001011010
    十六进制：8f430fcad45a
S-盒运算
    二进制:01011001001111100100100011111000
    十六进制：593c90f8
p-置换
    二进制：00101111000110001101010001000111
    十六进制：2f18d447
轮输出
    L[2]=1a67968d    R[2]=2fe7d4c3
```

而原先输入明文为：7465737464617461

输入密钥不变：6d796465736b6579
此时加密的第二轮输出如下：

```
*******第2轮加密*******
E扩展
    二进制：000011110000001000001111111011010010000000000001000
    十六进制：0f020fda4008
S-盒运算
    二进制:00000101111111000101000000101110
    十六进制：05fc502e
p-置换
    二进制：00101100000000111010101010111010 0
    十六进制：2c075574
轮输出
    L[2]=1847b204    R[2]=2cf85570
```

可以看到由于输入明文改变了一位，而加密的第二轮已经有很大的雪崩效应了，说明 des 的可用性。

基于 des 的 cmac

思路表述：由于要输入不定长的明文，所以我用到了 c++中 string 的知识，非常适合于这里不定长明文的要求，并且用到求 string 长度的 string：：size()函数来求输入明文长度。

代码清单如下（虽然与第一个 des 的程序只做了一点变化，但为了表达清晰，我还是将 des+cmac 的代码贴出来了！）：

主要多了void cmac()、void Encode_cmac()和void k1_k2_create()三个函数，并且其它一些函数也经过了简化！其中文字加深阴影处就是与cmac的主要代码。

其中void k1_k2_create()主要利用密钥产生了k1和k2的值，用于CMAC最后一组的异或。

```
#include<stdio.h>
#include<string.h>
#include<string>
#include<stdlib.h>
#include<iostream>
using namespace std;
void keyBuild(int *keychar);                                    //create key array
```

```
void keyCreate(int *midkey2,int movebit,int i);              //call by keyBuild
void F(int *rData,int *key);                                          //F function
void Expand(int *rData,int *rDataP);                          //Expand function
void ExchangeS(int *rDataP,int *rData);                       //S-diagram change
void ExchangeP(int *rData);                                           //P change
void FillBin(int *rData,int n,int s);                         // data to binary;call by
S-Diagram change function
void InitialPermutation(int *plaintext_64bit);
void Hex2Bit(char *text_byte,int *text_bit,int n);      //change into bit,n为bit的位数
void Bit2Hex(char *text_byte,int* text_bit,int n);      //change into hex,n为bit的位数
void RoundEncode(int n);
void cmac();
void Encode_cmac();
void k1_k2_create();
int IP1[]={58, 50, 42, 34, 26, 18, 10, 2,
                60, 52, 44, 36, 28, 20, 12, 4,      //initial change
                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,
 };
int IP2[]={40, 8, 48, 16, 56, 24, 64, 32,
                39, 7, 47, 15, 55, 23, 63, 31,      //opp initial change
                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
 };
int
s[][4][16]={{
        //S-diagram array
                        {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}
                        },
                       {
                        {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}
```

```
                    },
                    {
                      {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}
                    },
                    {
                      {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}
                    },
                    {
                      {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}
                    },
                    {
                      {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}
                    },
                    {
                      {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}
                    },
                    {
                      {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}
                    }
};
int Ex[48]={    32,1,2,3,4,5,
//Expand array
                    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
 };
int P[32]={16,7,20,21,29,12,28,17,
//P-change
                1,15,23,26,5,18,31,10,
                2,8,24,14,32,27,3,9,
                19,13,30,6,22,11,4,25
 };
int PC1[56]={57,49,41,33,25,17,9,                                    //PC-1 in keyBuild
                  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
 };
int PC2[48]={14,17,11,24,1,5,3,28,
//PC-2 in keyBuild
                  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
 };

int LSi[16]={1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

int subkey_bit[16][48];
char subkey_byte[16][16];
char plaintext_16byte[16];
int plaintext_64bit[64];
char ciphertext_16byte[16];
int ciphertext_64bit[64];
char key_16byte[16];
int key_64bit[64];
int mode;   //来区分是加密还是解密
string plaintext_cmac;
int k1[64];
char k1_byte[16];
int k1_temp[65];
int k2[64];
char k2_byte[16];
int k2_temp[66];
```

```
int main()                          //main function
{
	cmac();
	system("pause");
	return 0;
}
void cmac()
{
	cout<<"请输入密钥(16个字符):"<<endl;
	for(int i=0;i<16;i++)
		cin>>key_16byte[i];
	Hex2Bit(key_16byte,key_64bit,64);

	k1_k2_create();

	cout<<"请输入明文(可以大于16个字符)："<<endl;
	cin>>plaintext_cmac;
	int length=plaintext_cmac.size();
	cout<<"输入字符的长度为："<<length<<endl;
	for(int i=0;i<64;i++)
		//ciphertext_64bit初始化都为0！以便第一次异或ciphertext_64bit的时候没有取反！！
		ciphertext_64bit[i]=0;
	if(length%16==0)//如果输入的字符串长度正好是整数倍的分组长度，则如下处理
	{
		int i;
		for(i=0;i<length/16-1;i++)	//注意这里要-1
		{
			for(int j=0;j<16;j++)
				plaintext_16byte[j]=plaintext_cmac[j+i*16];
			Hex2Bit(plaintext_16byte,plaintext_64bit,64);
			for(int j=0;j<64;j++)
				plaintext_64bit[j]=plaintext_64bit[j]^ciphertext_64bit[j];
			Encode_cmac();
			cout<<"第"<<i+1<<"组加密后密文为："<<ciphertext_16byte<<endl;
		}
		for(int j=0;j<16;j++)
			plaintext_16byte[j]=plaintext_cmac[j+i*16];
		Hex2Bit(plaintext_16byte,plaintext_64bit,64);
		for(int j=0;j<64;j++)
			plaintext_64bit[j]=plaintext_64bit[j]^ciphertext_64bit[j];


		for(int j=0;j<64;j++)
			plaintext_64bit[j]=plaintext_64bit[j]^k1[j];//相当于与K1异或了！！！
```

```
            Encode_cmac();
            cout<<"第"<<i+1<<"组加密后密文为："<<ciphertext_16byte<<endl;
        }
        else if(length%16!=0)//如果输入的字符串长度不是整数倍的分组长度，则如下处理
        {
            int i;
            for(i=0;i<length/16;i++)    //注意这里不-1，与第一种情况不同。
            {
                for(int j=0;j<16;j++)
                    plaintext_16byte[j]=plaintext_cmac[j+i*16];
                Hex2Bit(plaintext_16byte,plaintext_64bit,64);
                for(int j=0;j<64;j++)
                    plaintext_64bit[j]=plaintext_64bit[j]^ciphertext_64bit[j];
                Encode_cmac();
                cout<<"第"<<i+1<<"组加密后密文为："<<ciphertext_16byte<<endl;
            }

            //下面几行与第一种情况主要不同的地方！！即处理100...000的情况
            for(int j=0;j<length%16;j++)
                plaintext_16byte[j]=plaintext_cmac[j+i*16];
            plaintext_16byte[length%16]='8';//即8=1000
            for(int j=length%16+1;j<16;j++)
                plaintext_16byte[j]='0';

            Hex2Bit(plaintext_16byte,plaintext_64bit,64);
            for(int j=0;j<64;j++)
                plaintext_64bit[j]=plaintext_64bit[j]^ciphertext_64bit[j];

            for(int j=0;j<64;j++)
                plaintext_64bit[j]=plaintext_64bit[j]^k2[j];;//相当于与K2异或了！！！

            Encode_cmac();
            cout<<"第"<<i+1<<"组加密后密文为："<<ciphertext_16byte<<endl;
        }
}
void Encode_cmac()
{
    InitialPermutation(plaintext_64bit);

    Hex2Bit(key_16byte,key_64bit,64);
    keyBuild(key_64bit);

    for(int i=0;i<16;i++)
    {
        RoundEncode(i);
```

```
            Bit2Hex(ciphertext_16byte,ciphertext_64bit,64);
        }
        int temp[64];     //逆初始置换
        for(int i=0;i<64;i++)
            temp[i]=ciphertext_64bit[i];
        for(int i=0;i<32;i++)
            ciphertext_64bit[i]=temp[i+32];
        for(int i=0;i<32;i++)
            ciphertext_64bit[i+32]=temp[i];

        for(int i=0;i<64;i++)
            temp[i]=ciphertext_64bit[i];
        for(int i=0;i<64;i++)
            ciphertext_64bit[i]=temp[ IP2[i]-1    ];
        Bit2Hex(ciphertext_16byte,ciphertext_64bit,64);
}
void k1_k2_create()     //用于产生k1和k2
{
        for(int i=0;i<64;i++)
            plaintext_64bit[i]=0;
        Encode_cmac();//对E(K,0^n)加密
        cout<<"L=E(K,0^n)="<<ciphertext_16byte<<endl;
        Hex2Bit(ciphertext_16byte,ciphertext_64bit,64);
        cout<<"L=E(K,0^n)二进制为： ";
        for(int i=0;i<64;i++)
            cout<<ciphertext_64bit[i];
        cout<<endl;

        for(int i=0;i<64;i++)//给k1，k2赋初值，为L的值
        {
            k1[i]=ciphertext_64bit[i];
            k2[i]=ciphertext_64bit[i];
        }

        for(int i=0;i<64;i++)
            k1_temp[i]=k1[i];
        k1_temp[64]=0;//k1_temp为k1乘以x后的值。
        //if(k1_temp[0]==0)
        //由于要在有限域中取x^64+x^4+x^3+1 取余，当最高位即x^64为0的时候，k1不变，所以不
//用处理
        if(k1_temp[0]==1)
        {//取x^64+x^4+x^3+1 取余
            k1_temp[0]=k1_temp[0]^1;
            k1_temp[60]=k1_temp[60]^1;
            k1_temp[61]=k1_temp[61]^1;
```

```
            k1_temp[63]=k1_temp[63]^1;
            k1_temp[64]=k1_temp[64]^1;
        }
        for(int i=0;i<64;i++)
            k1[i]=k1_temp[i+1];//此时k1=L*x
        Bit2Hex(k1_byte,k1,64);
        cout<<"K1="<<k1_byte<<endl;

        for(int i=0;i<64;i++)
            k2_temp[i]=k2[i];
        k2_temp[64]=0;
        k2_temp[65]=0;//k2_temp为k2乘以x^2后的值。
        if(k2_temp[0]==1)
        {
            k2_temp[0]=k2_temp[0]^1;
            k2_temp[60]=k2_temp[60]^1;
            k2_temp[61]=k2_temp[61]^1;
            k2_temp[63]=k2_temp[63]^1;
            k2_temp[64]=k2_temp[64]^1;
        }
        if(k2_temp[1]==1)
        {
            k2_temp[1]=k2_temp[1]^1;
            k2_temp[61]=k2_temp[61]^1;
            k2_temp[62]=k2_temp[62]^1;
            k2_temp[64]=k2_temp[64]^1;
            k2_temp[65]=k2_temp[65]^1;
        }
        for(int i=0;i<64;i++)
            k2[i]=k2_temp[i+2];//此时k2=L*x^2
        Bit2Hex(k2_byte,k2,64);
        cout<<"K2="<<k2_byte<<endl;
}
void InitialPermutation(int *plaintext_64bit)
{
        int temptext[64];
        for(int i=0;i<64;i++)
            temptext[i]=plaintext_64bit[i];
        for(int i=0;i<64;i++)
            temptext[i]=plaintext_64bit[IP1[i]-1];
        for(int i=0;i<64;i++)
            ciphertext_64bit[i]=temptext[i];
        Bit2Hex(ciphertext_16byte,temptext,64);
}
void Hex2Bit(char *text_byte,int *text_bit,int n)       //change into bit,n为bit的位数
```

```
{
	int length=n/4;
	int temptext[length];
	for(int i=0;i<length;i++)
	{
		if(text_byte[i]>='a'&&text_byte[i]<='f')
			temptext[i]=text_byte[i]-'a'+10;
		else if(text_byte[i]>='0'&&text_byte[i]<='9')
			temptext[i]=text_byte[i]-'0';
	}
	for(int i=0;i<length;i++)
	{
		for(int j=3;j>=0;j--)
		{
			text_bit[i*4+j]=temptext[i]%2;
			temptext[i]/=2;
		}
	}
}
void Bit2Hex(char *text_byte,int* text_bit,int n)	//change into hex,n为bit的位数
{
	int length=n/4;
	int temptext[length];
	for(int i=0;i<length;i++)
	{
		temptext[i]=text_bit[i*4+0]*8;
		temptext[i]+=text_bit[i*4+1]*4;
		temptext[i]+=text_bit[i*4+2]*2;
		temptext[i]+=text_bit[i*4+3]*1;
	}
	for(int i=0;i<length;i++)
	{
		if(temptext[i]>=0&&temptext[i]<=9)
			text_byte[i]=temptext[i]+'0';
		else if(temptext[i]>=10&&temptext[i]<=15)
			text_byte[i]=(temptext[i]-10)+'a';
	}
}

void keyBuild(int *keychar)	//create key array
{
	int i,j,k;
	int movebit[]={1,1,2,2,2,2,2,2,
				1,2,2,2,2,2,2,1};
	int midkey2[56];
```

```
    int midkey[64];
    int key[56];
    for(k=0;k<56;k++)
    {
       midkey2[k]=keychar[PC1[k]-1];
    }
    for(i=0;i<16;i++)                          //密钥按照移位表循环左移,i代表轮数
    {

       keyCreate(midkey2,movebit[i],i);
    }
}
// midkey2是pc-1置换后的56bit密钥，movebit为左移的位数，n为第几轮
void keyCreate(int *midkey2,int movebit,int n)    //create key
{
    int i,temp[4];
    temp[0]=midkey2[0];
    temp[1]=midkey2[1];
    temp[2]=midkey2[28];
    temp[3]=midkey2[29];
   if(movebit==2)          //左移两位的算法
   {
        for(i=0;i<26;i++)
        {
           midkey2[i]=midkey2[i+2];
           midkey2[i+28]=midkey2[i+30];
        }
        midkey2[26]=temp[0];
        midkey2[27]=temp[1];
        midkey2[54]=temp[2];
        midkey2[55]=temp[3];
   }
     else                                  //左移一位的算法
     { for(i=0;i<27;i++)
     {
         midkey2[i]=midkey2[i+1];
         midkey2[i+28]=midkey2[i+29];
        }
        midkey2[27]=temp[0];midkey2[55]=temp[2];
     }
     //pc-2置换后变成48位
    for(i=0;i<48;i++)
    {
       subkey_bit[n][i]=midkey2[PC2[i]-1];
    }
```

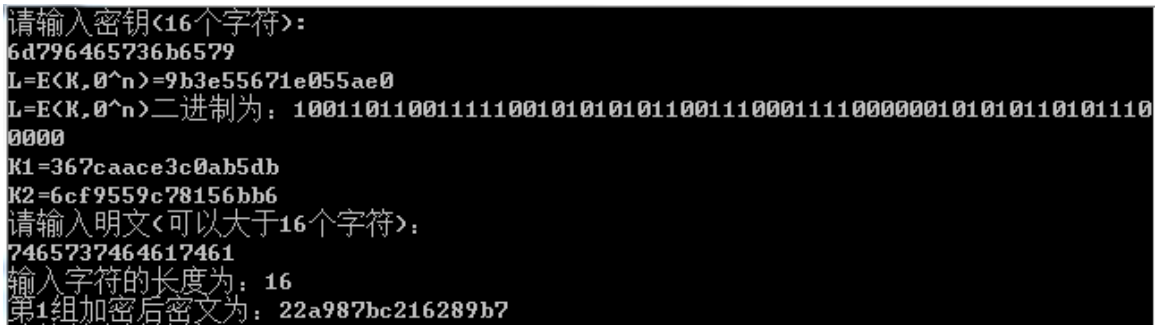
```
    Bit2Hex(subkey_byte[n],subkey_bit[n],48);
}


void RoundEncode(int n)   //第n轮加密
{
    int lData[32],rData[32];
    int rDataE[48];
    for(int i=0;i<32;i++)
        lData[i]=ciphertext_64bit[i];
    for(int i=0;i<32;i++)
        rData[i]=ciphertext_64bit[i+32];
    Expand(rData,rDataE);
    for(int i=0;i<48;i++)        //取异或
        rDataE[i]=rDataE[i]^subkey_bit[n][i];
    ExchangeS(rDataE,rData);   //S盒变换
    ExchangeP(rData);             //P置换
    for(int i=0;i<32;i++)         //取异或
        rData[i]=rData[i]^lData[i];
    for(int i=0;i<32;i++)
        ciphertext_64bit[i]=ciphertext_64bit[i+32];
    for(int i=0;i<32;i++)
        ciphertext_64bit[i+32]=rData[i];
}
void F(int *rData,int *key)        //F function
{
    int i,rDataP[48];
    Expand(rData,rDataP);      //将明文右半部分进行E扩展，扩展成48位的A
    for(i=0;i<48;i++)
    {
      rDataP[i]=rDataP[i]^key[i];      //将进行完E扩展的A与key1异或
    }
    ExchangeS(rDataP,rData);   //S盒变换
    ExchangeP(rData);
}
void Expand(int *rData,int *rDataP) //E扩展   将明文右半部分进行E扩展，扩展成48位
{
    int i;
    char temp[12];
    for(i=0;i<48;i++)
    {
      rDataP[i]=rData[Ex[i]-1];
    }
    Bit2Hex(temp,rDataP,48);
}
```

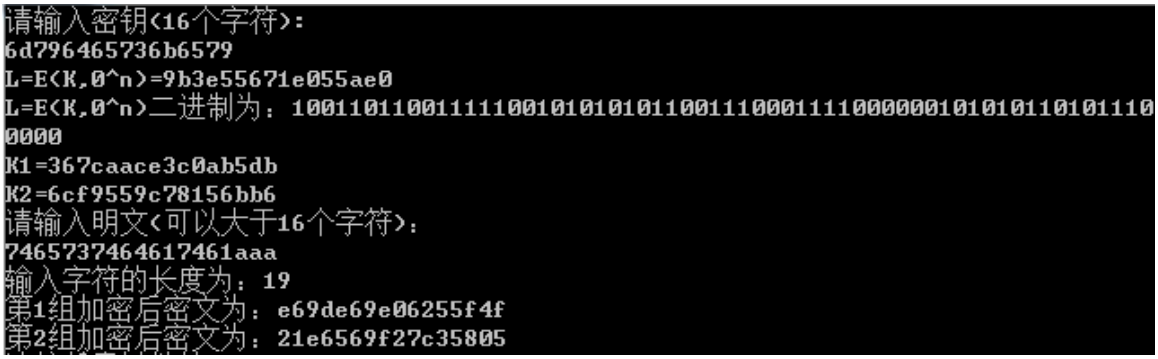
```
void ExchangeS(int *rDataP,int *rData){                    //S盒变换
    int i,n,linex,liney;
    linex=liney=0;
    for(i=0;i<48;i+=6)
    {
        n=i/6; //printf("%10d\n",(rDataP[i]<<1));
        linex=(rDataP[i]<<1)+rDataP[i+5];
        liney=(rDataP[i+1]<<3)+(rDataP[i+2]<<2)+(rDataP[i+3]<<1)+rDataP[i+4];
        FillBin(rData,n,s[n][linex][liney]);

    }
    char temp[8];
    Bit2Hex(temp,rData,32);
}
void FillBin(int *rData,int n,int s)     // data to binary;call by S-Diagram change function
{
    int temp[4],i;
    for(i=0;i<4;i++)
    {
        temp[i]=s%2;
        s=s/2;
     }
    for(i=0;i<4;i++)
    {
        rData[n*4+i]=temp[3-i];
    }
}
void ExchangeP(int *rData)
{                                          //P change
    int i,temp[32];
    for(i=0;i<32;i++)
       temp[i]=rData[i];
    for(i=0;i<32;i++)
       rData[i]=temp[P[i]-1];
    char temp2[8];
    Bit2Hex(temp2,rData,32);
}
```

最后输出的 64 位数据作为 CMAC，一些调试的例子如下：



上面这个例子中输入的明文长度为 16，输入明文和密钥与上面讲的 des 例子一样，但是输出却不一样了。这是因为这种情况下异或了一个 K1 数组，输出的与原来应该输出的密文完全不同。

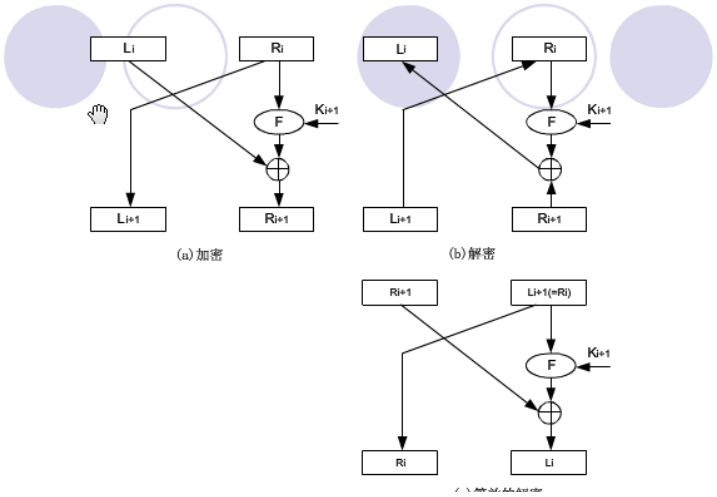


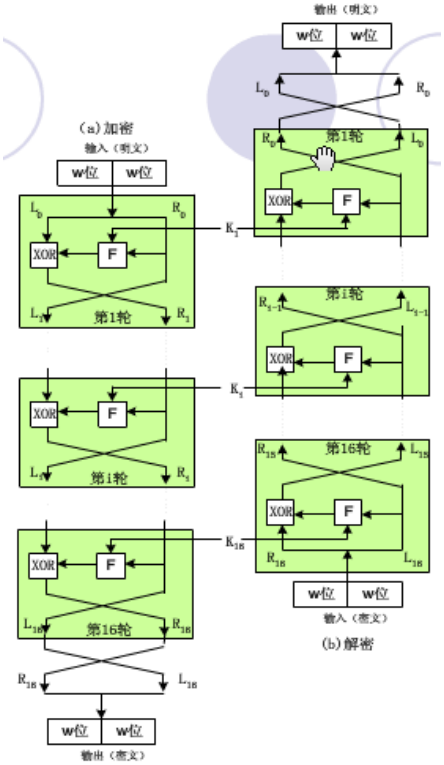
上面这个例子输出的明文多了3个a，输入明文长度为19，但是可以看到第1组加密后输出为E6 9D E6 9E 06 25 5F 4F，与输入明文：D=74 65 73 74 64 61 74 61，输入密钥：K=6D 79 64 65 73 6B 65 79 情况输出的密文相同。可以看到输出结果与实际物理意义符合。此时，最后一组输出的密文作文CMAC，也就是这个例子中的21 E6 56 9F 27 C3 58 05。**（即我的程序是用64位作为CMAC，当然也可以选择高位作为CMAC）**。

五、思考题

- **DES 解密算法和 DES 的逆算法之间有什么不同？**

DES 加解密是基于 Feistel 结构的，右下图单论的 Feistel 加密与解密结构可以清楚的看到 DES 加密与解密算法的区别与共同之处：





理论上，加密算法每一轮的输出，与解密算法相对应的那一轮输入左右的数据交换之后是相同的。

还有一点，加密算法先要经过初始置换，最后要经过逆初始置换，而解密算法也要先经过初始置换，最后再经过逆初始置换，因为初始置换与逆初始置换之间有

一个相逆的过程。

⊕ CMAC 与 HMAC 相比，有什么优点？

首先 CMAC 的结构比 HMAC 的结构简单，CMAC 中每一组中做的事情是相同的，只有最后一组需要有 K1 或 K2 的影响。因此构造起来简单。观察其框图，可以看到 CMAC 的构造方法很类似于密文分组链接模式（CBC），只是最后多了 K1 和 K2 两个数组异或。

其次来看一下 HMAC 和 CMAC 的优缺点：

HMAC 的优点是：这种构造方法具备很多优点，和同类型的 MAC 算法相比，它给出了安全性证明——将 MAC 的安全性归结到所使用的 Hash 函数上。在软件实现上，它要比使用分组密码构造的 MAC 快，并且它的效率特别高。从它的构造上可以看出，它以一种非常简单的方式使用带密钥的 Hash 函数，同底层的 Hash 函数相比，性能并没有降低多少。另外两个值得称道的优点是免费和黑盒。免费是指使用 Hash 函数不受法律限制，可以免费使用。黑盒是指可以将底层的 Hash 函数看成一个模块，可根据需要方便地进行更换。

HMAC 缺点是：安全性依赖于底层的 Hash 函数，而所使用的 Hash 函数有些是没有安全性证明的，所以不能保证这种方法的安全性。其次由于压缩函数是串行的，该构造方法不支持并行。

CMAC 优点是：这种方法出现得较早，是一种经典的构造方法，其构造方法简单，底层加密算法具备黑盒的性质，可以方便地进行替换。后来的很多 MAC 算法都是对它的改进。

CMAC 缺点是：CMAC 仅适用于对相同长度的消息进行认证，在消息长度变化的情况下是不安全的。这些在文献[5]中都已经给出了证明，另外，它的构造方法决定了该算法不支持并行计算。

由以上分析，可以知道 CMAC 相对来说可以做的比 HMAC 安全。并支持并行。

六、心得体会

DES 算法还是比较简单的，每一轮的 F 函数里面做的步骤比较多，刚开始写程序的时候想要“一步登天”直接把 F 函数写好。结果出了很多错误。后来就改成把 F 函数里面的扩展/置换、代替/选择、置换（P）等都分步骤写。结果就对了。

在本次编写程序过程中，深有对 C 语言和 C++语言输入方式的不同的感触。在 C 语言中的输入用 scanf 输入，结果经常把 enter 键也输入了，而这是不必要的。反而，用 C++的 cin 输入，则没有这个问题。C++中采用的是输入流。但是从底层的速度来说 C 语言中的 scanf 比 C++重的 cin 快些。可以说各有优缺点。个人建议如果对速度没有过分要求的话还是采用 cin 输入比较合适。

CMAC 算法是在 DES 加解密算法的基础上完成的算法，在编写的过程中遇到了很多问题，有些是对 C++语言的错误写法，有些则是对 CMAC 算法的理解错误（如 K1、K2 在最初的设计中被忽略了）。经过不断地改进，最终完成 CMAC 算法。