

**信息安全概论实验报告**

实验1-2 对称密码算法DES

**学 院**  电子与信息学院

**班 级** 信息工程7班

**学生姓名**  李文剑

**指导教师** 向友君

**提交日期** 2016年 1月7 日

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。

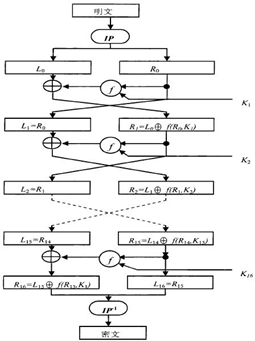
二、实验原理：

信息加密根据采用的密钥类型可以划分为对称密码算法和非对称密码算法。对称密码算法是指加密系统的加密密钥和解密密钥相同，或者虽然不同，但是可以从其中任意一个推导出另一个，更形象的说就是用同一把钥匙开锁和解锁。在对称密码算法的发展历史中曾出现过多种优秀的算法，包括DES、3DES、AES等。下面我们以DES算法为例介绍对称密码算法的实现机制。

DES算法是有美国IBM公司在20世纪70年代提出，并被美国政府、美国国家标准局和美国国家标准协会采纳和承认的一种标准加密算法。它属于分组加密算法，即明文加密和密文解密过程中，信息都是按照固定长度分组后进行处理的。混淆和扩散是它采用的两个最重要的安全特性，混淆是指通过密码算法使明文和密文以及密钥的关系非常复杂，无法从数学上描述或者统计。扩散是指明文和密钥中每一位信息的变动，都会影响到密文中许多位信息的变动，从而隐藏统计上的特性，增加密码安全。

DES将明文分成64比特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，则将其扩展为64位（例如补零等方法）。具体加密过程首先是将输入的数据进行初始换位（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施行在子密钥（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施行逆初始置换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。

DES全部16轮的加密过程如图1所示。



三、实验环境：

运行windows或linux操作系统的PC机，具有gcc（linux）、VC（windows）等C语言编译环境。

四、实验步骤：

1．使用光盘附录提供的程序对一个文件进行加密和解密，提交程序代码和执行结果。

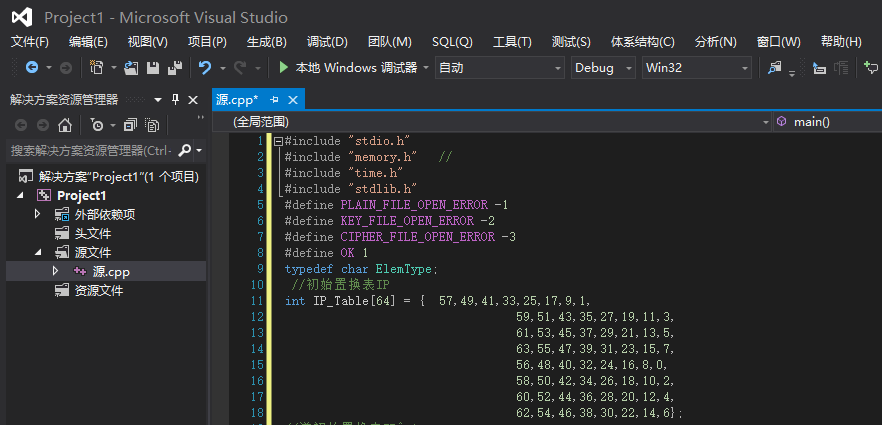
2．使用光盘附录提供的程序对输入的十六进数加密（把输入的字符转化成整数。比如输入两个字符1F，转化成二进制数00011111），比较输入和输出。当把输入的数改变一个比特时（如把1F变为1E），比较输出的变化，说明原因。

五、实验过程：

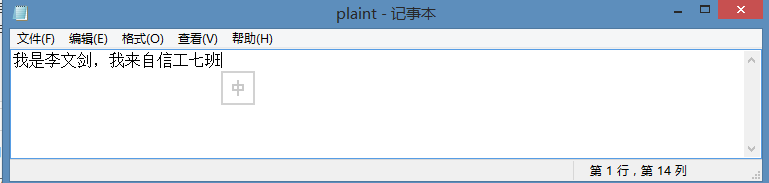
（一）对一个文件进行加密和解密，提交程序代码和执行结果。

1．完成了对一个文档的加密与解密。

采用在window8下的vs2012的开发编译环境：如下图所示：

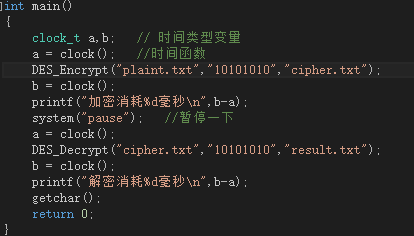


2. 待加密文件的输入。命名为plaint,保存至程序目录下。

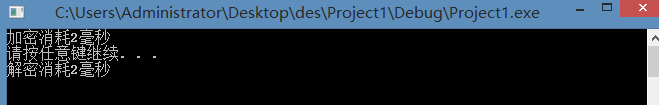


3. 运行加解密程序。

主函数的程序如下：key为[a;a;a;a]其中a=[10101010]

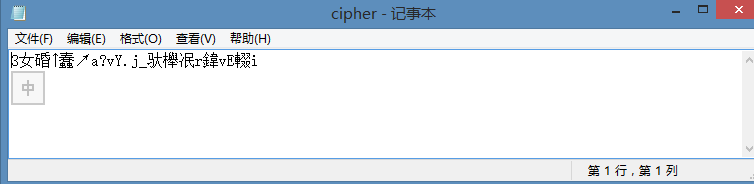


加密解密耗时

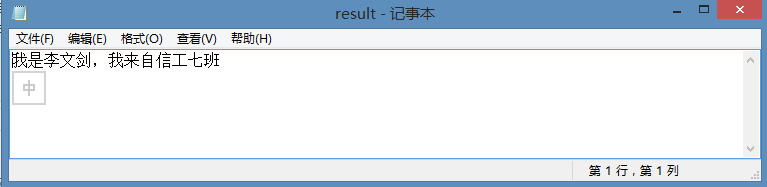


结果：

(1)文件加密的结果为cipher,如下图。

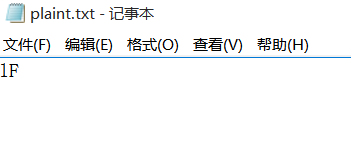


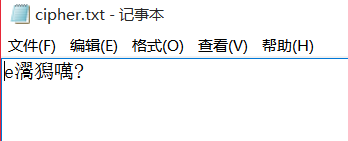
(2) 文件加密的结果为result,如下图，准确完整地恢复了原始的文件。

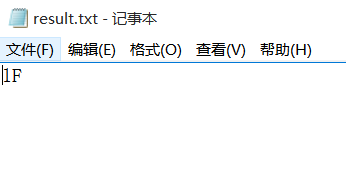


（二）．使用光盘附录提供的程序对输入的十六进数加密（把输入的字符转化成整数。比如输入两个字符1F，转化成二进制数00011111），比较输入和输出。

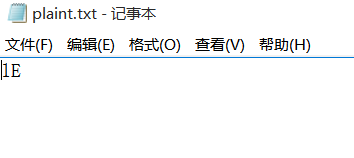
对1F：加密解密

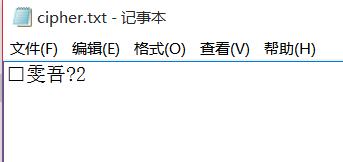


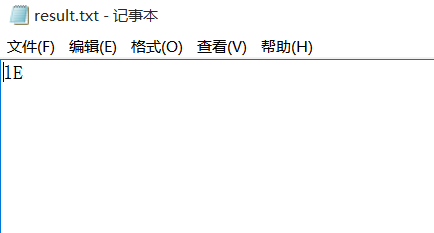
****

****

对1E:加密解密







**六、结论**

从实验结果可以看出，明文就算是相差一个位,但是他们的密文完全没有规律可循,这就保证了数据加密的可靠性,至于这样的原因,DES是由于采用了Feistel结构,在加密过程中经过多轮重复操作,但每一轮的子密钥不同,这样就使得即使原来很相近的两个数据串经过N轮后,这个数据串被分得很散,也就很不一样了.

代码附录：

#include "stdio.h"

#include "memory.h" //

#include "time.h"

#include "stdlib.h"

#define PLAIN\_FILE\_OPEN\_ERROR -1

#define KEY\_FILE\_OPEN\_ERROR -2

#define CIPHER\_FILE\_OPEN\_ERROR -3

#define OK 1

typedef char ElemType;

//初始置换表IP

int IP\_Table[64] = { 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,

56,48,40,32,24,16,8,0,

58,50,42,34,26,18,10,2,

60,52,44,36,28,20,12,4,

62,54,46,38,30,22,14,6};

//逆初始置换表IP^-1

int IP\_1\_Table[64] = {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,

32,0,40,8,48,16,56,24};

//扩充置换表E 32--->48

int E\_Table[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};

//置换函数P

int P\_Table[32] = {15,6,19,20,28,11,27,16,

0,14,22,25,4,17,30,9,

1,7,23,13,31,26,2,8,

18,12,29,5,21,10,3,24};

//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}}};

//置换选择1

int PC\_1[56] = {56,48,40,32,24,16,8,

0,57,49,41,33,25,17,

9,1,58,50,42,34,26,

18,10,2,59,51,43,35,

62,54,46,38,30,22,14,

6,61,53,45,37,29,21,

13,5,60,52,44,36,28,

20,12,4,27,19,11,3};

//置换选择2

int PC\_2[48] = {13,16,10,23,0,4,2,27,

14,5,20,9,22,18,11,3,

25,7,15,6,26,19,12,1,

40,51,30,36,46,54,29,39,

50,44,32,46,43,48,38,55,

33,52,45,41,49,35,28,31};

//对左移次数的规定

int MOVE\_TIMES[16] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

int ByteToBit(ElemType ch,ElemType bit[8]);

int BitToByte(ElemType bit[8],ElemType \*ch);

int Char8ToBit64(ElemType ch[8],ElemType bit[64]);

int Bit64ToChar8(ElemType bit[64],ElemType ch[8]);

int DES\_MakeSubKeys(ElemType key[64],ElemType subKeys[16][48]);

int DES\_PC1\_Transform(ElemType key[64], ElemType tempbts[56]);

int DES\_PC2\_Transform(ElemType key[56], ElemType tempbts[48]);

int DES\_ROL(ElemType data[56], int time);

int DES\_IP\_Transform(ElemType data[64]);

int DES\_IP\_1\_Transform(ElemType data[64]);

int DES\_E\_Transform(ElemType data[48]);

int DES\_P\_Transform(ElemType data[32]);

int DES\_SBOX(ElemType data[48]);

int DES\_XOR(ElemType R[48], ElemType L[48],int count); //异或操作

int DES\_Swap(ElemType left[32],ElemType right[32]); //交换 L分组与R分组交换.

int DES\_EncryptBlock(ElemType plainBlock[8], ElemType subKeys[16][48], ElemType cipherBlock[8]);

int DES\_DecryptBlock(ElemType cipherBlock[8], ElemType subKeys[16][48], ElemType plainBlock[8]);

int DES\_Encrypt(char \*plainFile, char \*keyStr,char \*cipherFile);

int DES\_Decrypt(char \*cipherFile, char \*keyStr,char \*plainFile);

//字节转换成二进制

int ByteToBit(ElemType ch, ElemType bit[8]){

int cnt;

for(cnt = 0;cnt < 8; cnt++){

\*(bit+cnt) = (ch>>cnt)&1; //右移cnt位,与1相与.将字节装入8位二进制数组中.倒序装入

} //7=00000111,bit={11100000}

//printf("%s",bit);

return 0;

}

//二进制转换成字节

int BitToByte(ElemType bit[8],ElemType \*ch){

int cnt;

for(cnt = 0;cnt < 8; cnt++){

\*ch |= \*(bit + cnt)<<cnt; // 为\*(1)+\*(10)+\*(100)+0000+00000+0000000...=7

}

return 0;

}

//将长度为8的字符串转为二进制位串

int Char8ToBit64(ElemType ch[8],ElemType bit[64]){

int cnt;

for(cnt = 0; cnt < 8; cnt++){

ByteToBit(\*(ch+cnt),bit+(cnt<<3)); //cnt<<3即为cnt\*8

}

return 0;

}

//将二进制位串转为长度为8的字符串

int Bit64ToChar8(ElemType bit[64],ElemType ch[8]){

int cnt;

memset(ch,0,8); //以ch为起点的8位空间数组置0.

for(cnt = 0; cnt < 8; cnt++){

BitToByte(bit+(cnt<<3),ch+cnt);

}

return 0;

}

//IP置换

int DES\_IP\_Transform(ElemType data[64]){

int cnt;

ElemType temp[64];

for(cnt = 0; cnt < 64; cnt++){

temp[cnt] = data[IP\_Table[cnt]];

}

memcpy(data,temp,64); //由temp所指内存区域复制64个字节到data所指内存区域,与strcpy 同.

return 0;

}

//IP逆置换

int DES\_IP\_1\_Transform(ElemType data[64]){

int cnt;

ElemType temp[64];

for(cnt = 0; cnt < 64; cnt++){

temp[cnt] = data[IP\_1\_Table[cnt]];

}

memcpy(data,temp,64); //由temp所指内存区域复制64个字节到data所指内存区域,与strcpy同

return 0;

}

//密钥置换1 pc-1

int DES\_PC1\_Transform(ElemType key[64], ElemType tempbts[56]){

int cnt;

for(cnt = 0; cnt < 56; cnt++){

tempbts[cnt] = key[PC\_1[cnt]];

}

return 0;

}

//密钥置换2 pc-2

int DES\_PC2\_Transform(ElemType key[56], ElemType tempbts[48]){

int cnt;

for(cnt = 0; cnt < 48; cnt++){

tempbts[cnt] = key[PC\_2[cnt]];

}

return 0;

}

//E扩展置换

int DES\_E\_Transform(ElemType data[48]){ //这里data两种用处,故为48位极人臣

int cnt;

ElemType temp[48];

for(cnt = 0; cnt < 48; cnt++){

temp[cnt] = data[E\_Table[cnt]]; //这里data下标最高为32.

}

memcpy(data,temp,48);

return 0;

}

//P置换

int DES\_P\_Transform(ElemType data[32]){

int cnt;

ElemType temp[32];

for(cnt = 0; cnt < 32; cnt++){

temp[cnt] = data[P\_Table[cnt]];

}

memcpy(data,temp,32);

return 0;

}

//循环左移

//左右两边同时循环左移相同的位数.

//10010循环左移1位为00101,左移2位为01010

int DES\_ROL(ElemType data[56], int times){

ElemType temp[56]; //temp只保存将要循环到右边的位.

//保存将要循环移动到右边的位

memcpy(temp,data,times); //保存前28位的移位

memcpy(temp+times,data+28,times); //保存后28位的移位极人臣

//前28位移动

memcpy(data,data+times,28-times); //一般移位

memcpy(data+28-times,temp,times); //循环位移动

//后28位移动

memcpy(data+28,data+28+times,28-times);

memcpy(data+56-times,temp+times,times);

return 0;

}

//生成子密钥

int DES\_MakeSubKeys(ElemType key[64],ElemType subKeys[16][48]){

ElemType temp[56];

int cnt;

DES\_PC1\_Transform(key,temp);//PC1置换

for(cnt = 0; cnt < 16; cnt++){//16轮跌代，产生16个子密钥

DES\_ROL(temp,MOVE\_TIMES[cnt]);//Ci Di 左右两边,循环左移

DES\_PC2\_Transform(temp,subKeys[cnt]);//PC2置换，产生子密钥

}

return 0;

}

//异或

int DES\_XOR(ElemType R[48], ElemType L[48] ,int count){

int cnt;

for(cnt = 0; cnt < count; cnt++){

R[cnt] ^= L[cnt]; //异或即为模2加法

}

return 0;

}

//S盒置换 48=8\*6 32=8\*4

int DES\_SBOX(ElemType data[48]){

int cnt;

int line,row,output;

int cur1,cur2;

for(cnt = 0; cnt < 8; cnt++){

cur1 = cnt\*6;

cur2 = cnt<<2;

//计算在S盒中的行与列 b1b2b3b4b5b6 b1b2 //b3b4b5b6

line = (data[cur1]<<1) + data[cur1+5];

row = (data[cur1+1]<<3) + (data[cur1+2]<<2)

+ (data[cur1+3]<<1) + data[cur1+4];

output = S[cnt][line][row];

//化为2进制

data[cur2] = (output&0X08)>>3; //取得output二进制倒数第4位为0还是1

data[cur2+1] = (output&0X04)>>2; //output=7=0111 oxo4=0100

data[cur2+2] = (output&0X02)>>1; //相&得到0100 右移2位得到1

data[cur2+3] = output&0x01;

}

return 0;

}

//交换 L分组与R分组交换.

int DES\_Swap(ElemType left[32], ElemType right[32]){

ElemType temp[32];

memcpy(temp,left,32);

memcpy(left,right,32);

memcpy(right,temp,32);

return 0;

}

//加密单个分组

int DES\_EncryptBlock(ElemType plainBlock[8], ElemType subKeys[16][48], ElemType cipherBlock[8]){

ElemType plainBits[64]; //明文比特

ElemType copyRight[48]; //

int cnt;

Char8ToBit64(plainBlock,plainBits);

//初始置换（IP置换）

DES\_IP\_Transform(plainBits);

//16轮迭代

for(cnt = 0; cnt < 16; cnt++){

//将右半部分拷贝到数组中.

memcpy(copyRight,plainBits+32,32);

//将右半部分进行扩展置换，从32位扩展到48位

DES\_E\_Transform(copyRight);

//将右半部分与子密钥进行异或操作

DES\_XOR(copyRight,subKeys[cnt],48);

//异或结果进入S盒，输出32位结果

DES\_SBOX(copyRight);

//P置换

DES\_P\_Transform(copyRight);

//将明文左半部分与右半部分进行异或

DES\_XOR(plainBits,copyRight,32);

if(cnt != 15){

//最终完成左右部的交换

DES\_Swap(plainBits,plainBits+32);

}

}

//逆初始置换（IP^1置换）

DES\_IP\_1\_Transform(plainBits);

Bit64ToChar8(plainBits,cipherBlock);

return 0;

}

//解密单个分组

int DES\_DecryptBlock(ElemType cipherBlock[8], ElemType subKeys[16][48],ElemType plainBlock[8]){

ElemType cipherBits[64];

ElemType copyRight[48];

int cnt;

Char8ToBit64(cipherBlock,cipherBits);

//初始置换（IP置换）

DES\_IP\_Transform(cipherBits);

//16轮迭代

for(cnt = 15; cnt >= 0; cnt--){

memcpy(copyRight,cipherBits+32,32); //这里为什么还是右半部分?

//将右半部分进行扩展置换，从32位扩展到48位

DES\_E\_Transform(copyRight);

//将右半部分与子密钥进行异或操作

DES\_XOR(copyRight,subKeys[cnt],48);

//异或结果进入S盒，输出32位结果

DES\_SBOX(copyRight);

//P置换

DES\_P\_Transform(copyRight);

//将明文左半部分与右半部分进行异或

DES\_XOR(cipherBits,copyRight,32);

if(cnt != 0){

//最终完成左右部的交换

DES\_Swap(cipherBits,cipherBits+32);

}

}

//逆初始置换（IP^1置换）

DES\_IP\_1\_Transform(cipherBits);

Bit64ToChar8(cipherBits,plainBlock);

return 0;

}

//加密文件

int DES\_Encrypt(char \*plainFile, char \*keyStr,char \*cipherFile){

FILE \*plain,\*cipher;

int count;

ElemType plainBlock[8],cipherBlock[8],keyBlock[8];

ElemType bKey[64];

ElemType subKeys[16][48];

if((plain = fopen(plainFile,"rb")) == NULL){

return PLAIN\_FILE\_OPEN\_ERROR;

}

if((cipher = fopen(cipherFile,"wb")) == NULL){

return CIPHER\_FILE\_OPEN\_ERROR;

}

//设置密钥

memcpy(keyBlock,keyStr,8);

//将密钥转换为二进制流

//printf("%s\n",keyBlock);

//int i;

//for(i=0;i<8;i++){

// printf(" %s \n",keyBlock[i]);

//}

Char8ToBit64(keyBlock,bKey);

//生成子密钥

DES\_MakeSubKeys(bKey,subKeys);

while(!feof(plain)){

//每次读8个字节，即一个分组读八次.并返回成功读取的字节数

if((count = fread(plainBlock,sizeof(char),8,plain)) == 8){

DES\_EncryptBlock(plainBlock,subKeys,cipherBlock);

fwrite(cipherBlock,sizeof(char),8,cipher);

}

}

if(count){

memset(plainBlock + count,'\0',7 - count);

plainBlock[7] = 8 - count;

DES\_EncryptBlock(plainBlock,subKeys,cipherBlock);

fwrite(cipherBlock,sizeof(char),8,cipher);

}

fclose(plain);

fclose(cipher);

return OK;

}

//解密文件

int DES\_Decrypt(char \*cipherFile, char \*keyStr,char \*plainFile){

FILE \*plain, \*cipher;

int count,times = 0;

long fileLen;

ElemType plainBlock[8],cipherBlock[8],keyBlock[8];

ElemType bKey[64];

ElemType subKeys[16][48];

if((cipher = fopen(cipherFile,"rb")) == NULL){

return CIPHER\_FILE\_OPEN\_ERROR;

}

if((plain = fopen(plainFile,"wb")) == NULL){

return PLAIN\_FILE\_OPEN\_ERROR;

}

//设置密钥

memcpy(keyBlock,keyStr,8);

//将密钥转换为二进制流

Char8ToBit64(keyBlock,bKey);

//生成子密钥

DES\_MakeSubKeys(bKey,subKeys);

//取文件长度

fseek(cipher,0,SEEK\_END); //将文件指针置尾,距SEEK\_END 0的位置,

fileLen = ftell(cipher); //取文件指针当前位置,相对于文件头,即可取文件长度.

rewind(cipher); //将文件指针重指向文件头

while(1){

fread(cipherBlock,sizeof(char),8,cipher);

DES\_DecryptBlock(cipherBlock,subKeys,plainBlock);

times += 8;

if(times < fileLen){

fwrite(plainBlock,sizeof(char),8,plain);

}

else{

break;

}

}

if(plainBlock[7] < 8){

for(count = 8 - plainBlock[7]; count < 7; count++){

if(plainBlock[count] != '\0'){

break;

}

}

}

if(count == 7){//有填充

fwrite(plainBlock,sizeof(char),8 - plainBlock[7],plain);

}

else{//无填充

fwrite(plainBlock,sizeof(char),8,plain);

}

fclose(plain);

fclose(cipher);

return OK;

}

int main()

{

clock\_t a,b; // 时间类型变量

a = clock(); //时间函数

DES\_Encrypt("plaint.txt","10101010","cipher.txt");

b = clock();

printf("加密消耗%d毫秒\n",b-a);

system("pause"); //暂停一下

a = clock();

DES\_Decrypt("cipher.txt","10101010","result.txt");

b = clock();

printf("解密消耗%d毫秒\n",b-a);

getchar();

return 0;

}