杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

《密码学实验》

实验报告

实验 4:SMS4 分组密码算法编程实验

姓　　名：易涛

学　　号：16031330

专　　业：信息安全

实验时间：2018.05.21

指导老师：吕秋云

一．实验目的

实验环境：Windows10 Visual Studio 2017

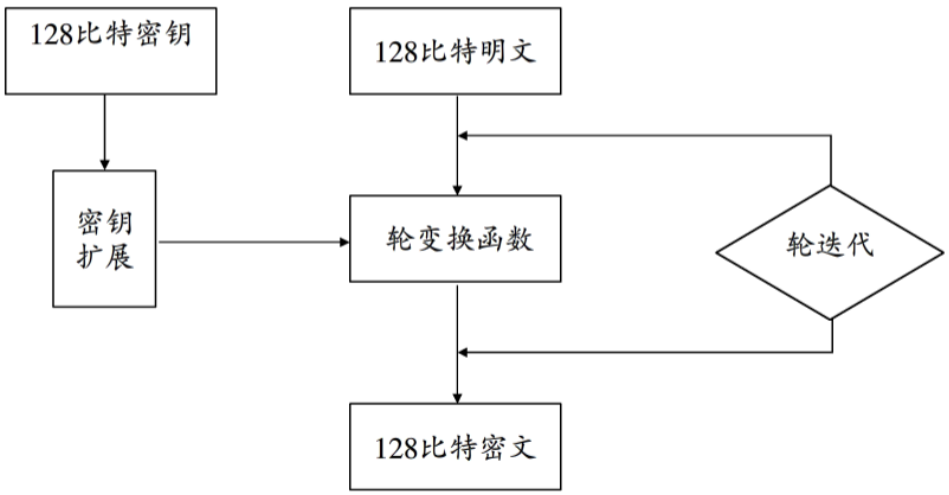
实验目的：

掌握 SMS4 密码加解密原理，并利用 Visual C++编程实现

加密原理：

SMS4 算法是一个对称分组算法，分组长度和密钥长度均为 128 比特。使用 32 轮的非线性迭代结构，在最后一轮非线性迭代之后加上一个反序变换，因此 SMS4 中职业解密密钥是加密密钥的逆序，它的解密算法和加密算法就可以保持一致、另外，SMS4 中的密钥扩展也是利用了 32 轮的非线性迭代结构。SMS4 结构图如下

SM4算法结构



实验要求：

1> 采用 MFC 编程实现简单页面编程

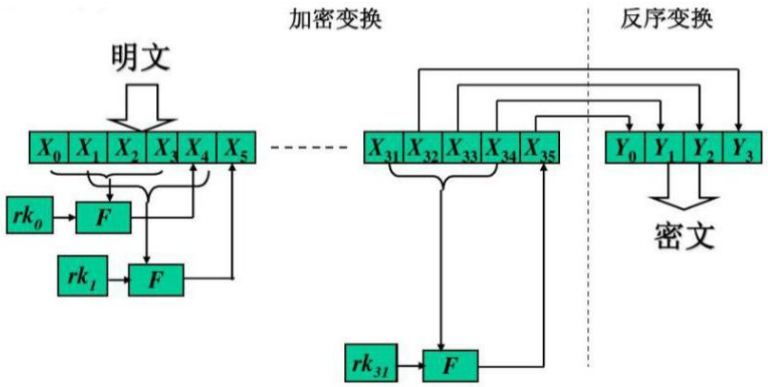
2> 模块化编程

3> 通过 SMS4 实现对不同文件格式的文件加解密

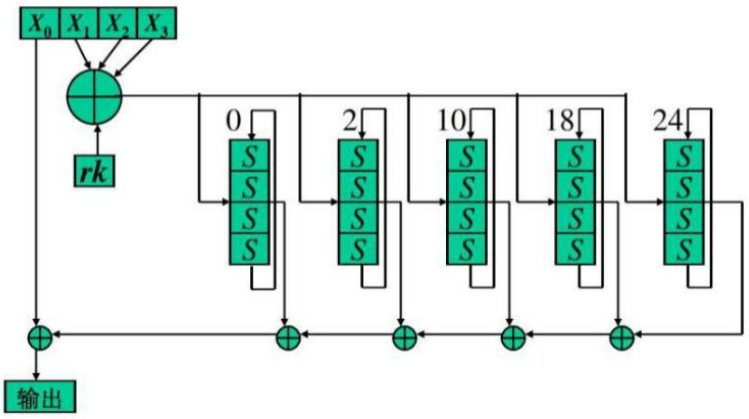
二．实验内容及实现过程步骤

1> SMS4 的加密步骤如图，因为采用模块化编程，故将主要步骤操作各编写成子函数，供上一层函数调用。

■ 加密算法



■ 轮函数F



2> 由上图分析可得，为实现模块化编程，以及使程序编写更严谨有逻辑，需定义下列子函数，因为存在底层调用，故先声明所有所需函数（并不编写函数代码），再由顶层向底层编写。

```
unsigned int* SMS4_Run(unsigned int msg[], bool flag);//主要加密函数
//unsigned int F_func(unsigned int M[4] ,unsigned int subkey);//轮函数
void SMS4_SetSubKey(unsigned int In[4]);//密钥扩展,生成全局变量rK[32],子密钥
unsigned int T_func(unsigned int t);//T置换
unsigned int S_func(unsigned int In);
unsigned int L_func(unsigned int In);
unsigned int leftmove(unsigned int In,int len);
unsigned int T1_func(unsigned int In);
unsigned int L1_func(unsigned int In);
void SMS4_File_En(char* msgFile, char* cipherFile);//需加密文件路径，加密生成路径
void SMS4_File_De(char* msgFile, char* cipherFile);
```

3> SMS4 顶层函数，SMS4_Run，SMS4_Run 通过对所有底层模块的调用实现 DES 加密和解密，代码如下图所示，参数由左至右分别表示：需加密的明文分组，因为 SMS4 分组长度为 128 比特，故使用 unsigned int 类型数组，4 长度表示 128 比特，模式选择（flag 为 1 则为加密，为 0 为解密）。

此函数流程为

1.SMS4 加密过程中共进行 32 轮，故定义 X[36]数组，用来存储暂时数据，且(M0,M1,M2,M3)=(X0,X1,X2,X3)

① 加密变换：对i=0, 1,…, 31,

$$X_{i+4}=F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i)=X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i)$$

② 反序变换$(Y_0, Y_1, Y_2, Y_3)=(X_{35}, X_{34}, X_{33}, X_{32})$

2.开始进行 32 轮加密变换，通过 flag 选择加密或是解密，前面已经提到，加解密过程除密钥逆序外，其他无区别，循环执行的操作如下图所示

① 加密变换：对i=0, 1,…, 31,

$$X_{i+4}=F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i)=X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i)$$

② 反序变换$(Y_0, Y_1, Y_2, Y_3)=(X_{35}, X_{34}, X_{33}, X_{32})$

3.经 32 轮变换之后，得到 X35,X34,X33,X32,为所需密文

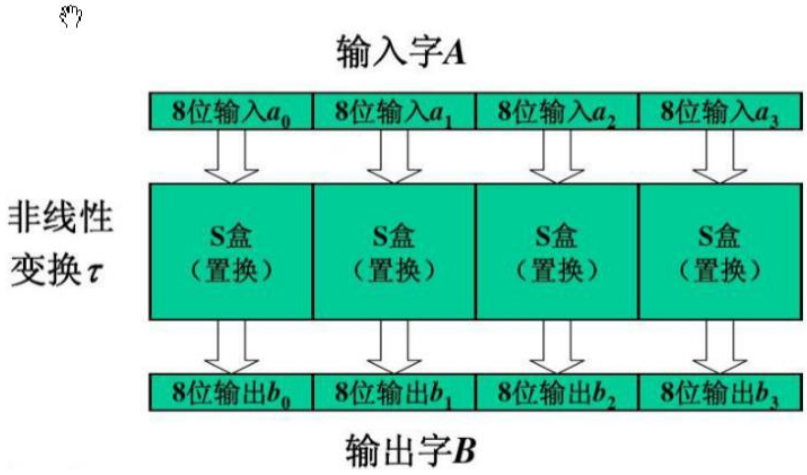
```
unsigned int* SMS4_Run(unsigned int msg[] , bool flag)
{
    unsigned int X[36] = { 0 };
    //初始化
    for (int i = 0; i < 4; i++)
    {
        X[i] = msg[i];
    }
    for (int i = 0; i < 32; i++)
    {
        if (flag)
            X[i + 4] = X[i] ^ (T_func(X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rK[i]));
        else
            X[i + 4] = X[i] ^ (T_func(X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rK[31 - i]));
    }
    unsigned int cipher[4] = { 0 };
    for (int i = 0; i < 4; i++)
        cipher[i] = X[35 - i];
    return cipher;
}
```

4> SMS4 加密顶层模块在 3 中给出，SMS4_Run 中用到了 T_func 函数，于是实现 T_func 函数，如图示，T 置换由非线性变换 S 函数，L 函数组成，将 T 函数的输入依次进行 S 函数和 L 函数变换后得到输出。

```
unsigned int T_func(unsigned int In)//S函数和L函数，输入32，输出32
{
    unsigned int Out = 0;//养成良好习惯
    Out = S_func(In);
    Out = L_func(Out);
    return Out;
}
```

T 函数又调用了 S_func 和 L_func 函数，所以分别进入这两个函数

首先是 S_func 函数，S_func 作用是将输入的 32 比特拆成 4 组 8 比特，然后将每 8 比特的高 4 比特作为行号，低 4 比特作为列号，查询 S 盒进行置换替代，然后将替代后的 4 组每组 8 比特共 32 比特重新拼装起来作为输出。




```
unsigned int S_func(unsigned int In)
{
    unsigned int Out = 0;
    /*unsigned*/ char Temp = { 0 };
    for (int i = 0; i < 4; i++)
    {
        Temp = ((In >> (24 - 8 * i)) & 0xFF);//8 bit保存在Temp中,由高位到低位
        Out = Out + (S_Box[Temp] << (24 - 8 * i));
        //上一步是由高位到低位，因此此处右移扩大，迎合高位到低位，
        //S_Box index为什么不需减1，因为8bit 最高1111 1111，从0开始计数
    }
    return Out;
}
```

L_func 函数实现的是一个简单的线性变换，将 32 比特的输入 B 进行下图操作，得到 32 比特输出 C

$$C=L(B)=B\oplus(B<<2)\oplus(B<<10)\oplus(B<<18)\oplus(B<<24)$$

```
unsigned int L_func(unsigned int In)
{
    unsigned int Out = 0;
    Out = In^leftmove(In, 2) ^ leftmove(In, 10)
        ^ leftmove(In, 18) ^ leftmove(In, 24);
    return Out;
}
```

L_func 函数中用到 leftmove 左移函数，leftmove 函数两个输入 In 和 len，功能是将输入 32 比特 In 左移 len 位后返回。

```
unsigned int leftmove(unsigned int In, int len)
{
    return (In << len) | (In >> (32 - len));//循环移位
}
```

5> 加密所用到函数都已实现，接下来是编写生成 32 轮子密钥的函数。

SMS4_SetSubKey 函数，SMS4 算法的加密和密钥扩展过程类似，都是 32 轮循环，循环变换过程都是由一次线性变换和一次非线性变换构成。SMS4_SetSubKey 函数首先定义一个中间变量数组 K[36]。输入的 In[4]数组分别和系统参数 FK 异或后，赋值给 K0，K1,K2,K3。然后进行 32 轮密钥扩展，过程如下图所示。

$(K_0, K_1, K_2, K_3)=(MK_0\oplus FK_0, MK_1\oplus FK_1, MK_2\oplus FK_2, MK_3\oplus FK_3)$，计算轮密钥

$rk_i=K_{i+4}=K_i\oplus T(K_{i+1}, K_{i+2}, K_{i+3}, MK_3\oplus CK_i)$

置换T'由非线性变换S盒和变换L'复合而成，L'是一些循环移位的异或：$L'(B)=B\oplus(B<<13)\oplus(B<<23)$

```
void SMS4_SetSubKey(unsigned int In[4])
{
    unsigned int K[36];//中间变量
    for (int i = 0; i < 4; i++)
        K[i] = In[i] ^ FK[i];
    for (int i = 0; i < 32; i++)
    {
        K[i + 4] = K[i] ^ T1_func(K[i + 1] ^ K[i + 2] ^ K[i + 3] ^ CK[i]);
        rK[i] = K[i + 4];
    }
}
```

T1_func 函数，将输入 32 比特 In 依次做 S_func 函数非线性变换和 L1_func 线性变换

```
unsigned int T1_func(unsigned int In)
{
    unsigned int Out = 0;
    Out = S_func(In);
    Out = L1_func(Out);
    return Out;
}
```

L1_func 函数，与 L_func 函数相似，不再详述

```
unsigned int L1_func(unsigned int In)
{
    return In^leftmove(In, 13) ^ leftmove(In, 23);
}
```

6> 加密过程和生成密钥代码均已编写完成，接下来编写函数对文件进行加解密

SMS4_File_En，文件加密函数，输入明文文件路径和生成的加密文件路径，将明文文件分组读入加密，128 比特是 16 字节，故每次读取 16 字节，因为加密代码中是使用 unsigned int 类型，32 比特为单位，所以需要将读入的 16 字节以 4 字节一组转成 4 个 unsigned int 变量，然后进行加密，加密完成后再转成字符串输出。

```
void SMS4_File_En(char* msgFile, char* cipherFile)
{
    char msgBlock[16] = { 0 };
    char cipherBlock[16] = { 0 };
    unsigned int msgTemp[4] = { 0 };
    unsigned int cipherTemp[4] = { 0 };
    ifstream msg(msgFile, ios::binary);
    ofstream cipher(cipherFile, ios::binary);

    while (msg.peek() != EOF)
    {
        memset(msgBlock, 0, 16);//初始化置0
        msg.read(msgBlock, 16);//读完文件指针会后移
        //memcpy(msgTemp, StrToHex(msgBlock), 16);

        memset(msgTemp, 0, 16);     //出错，解密多了填充的0
        //转16进制
        for (int i = 0; i < 16; i++)
            msgTemp[i / 4] += (msgBlock[i] & 0xff) << (24 - (i % 4) * 8);
        memcpy(cipherTemp, SMS4_Run(msgTemp, 1), 16);//加密

        //密文转换字符串
        for (int i = 0; i < 16; i++)
            cipherBlock[i] = (cipherTemp[i / 4] >> (24 - (i % 4) * 8))&0xff;
        cipher.write(cipherBlock, 16);
    }
    msg.close();
    cipher.close();
}
```

SMS4_File_De 函数，与 SMS4_File_En 函数类似，不多叙述。

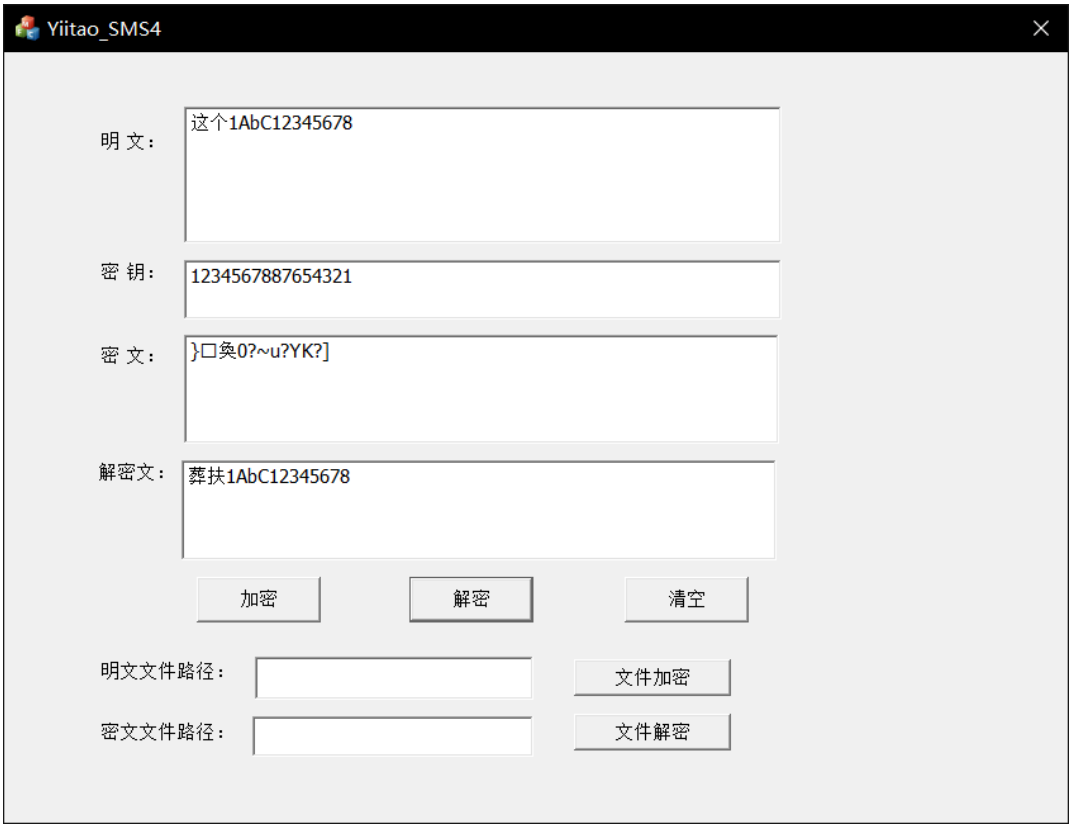
Yiitao_SMS4 (全局范围)

```
102  void SMS4_File_De(char* msgFile, char* cipherFile)
103  {
104      char msgBlock[16] = { 0 };
105      char cipherBlock[16] = { 0 };   //错误，解密之后文件有填充的0
106      unsigned int msgTemp[4] = { 0 };
107      unsigned int cipherTemp[4] = { 0 };
108      ifstream cipher(cipherFile, ios::binary);
109      ofstream msg(msgFile, ios::binary);
110
111      //int flag = 0;//尝试解决解密后多余0错误
112
113
114      while (cipher.peek() != EOF)
115      {
116          memset(cipherBlock, 0, 16);
117          cipher.read(cipherBlock, 16);//密文必为16整数
118          memset(cipherTemp, 0, 16);
119          for (int i = 0; i < 16; i++)//转16进制解密
120              cipherTemp[i / 4] += (cipherBlock[i] & 0xff) << (24 - 8 * (i % 4));
121          memcpy(msgTemp, SMS4_Run(cipherTemp, 0), 16);//解密
122          for (int i = 0; i < 16; i++)
123              msgBlock[i] = (msgTemp[i / 4] >> (24 - 8 * (i % 4))) & 0xff;
124          /* ... */
130          msg.write((char*)&msgBlock, 16);
131      }
132      msg.close();
133      cipher.close();
134  }
```

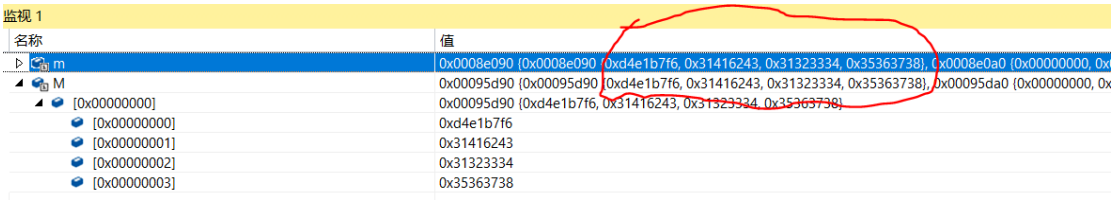
三．实验小结

1> 实验中遇到的错误

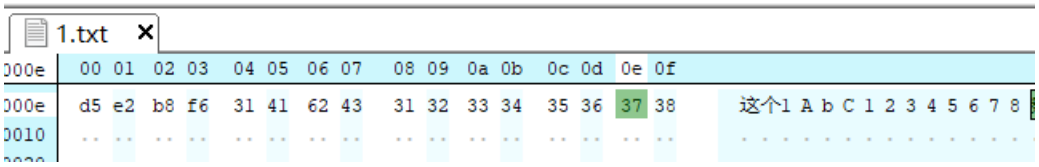
1.运行加密程序，发现对中文解密出现错误，但字符数字解密正确



解决方法：因为对字符和数字等解密正确，故可以确定加解密代码出错性不大，于是猜测可能是输入明文转成 unsigned int 时出错，于是下断点至明文转换之前，调试代码一步步执行至解密，对存储明文和解密文的 unsigned int 数组 m 和 M 添加监视，发现 m 和 M 数组内容一致，这表明加解密过程无错，能将密文 m 还原成解密文 M，证实了之前的猜想，即出错点在于明文由字符转 unsigned int 时出错

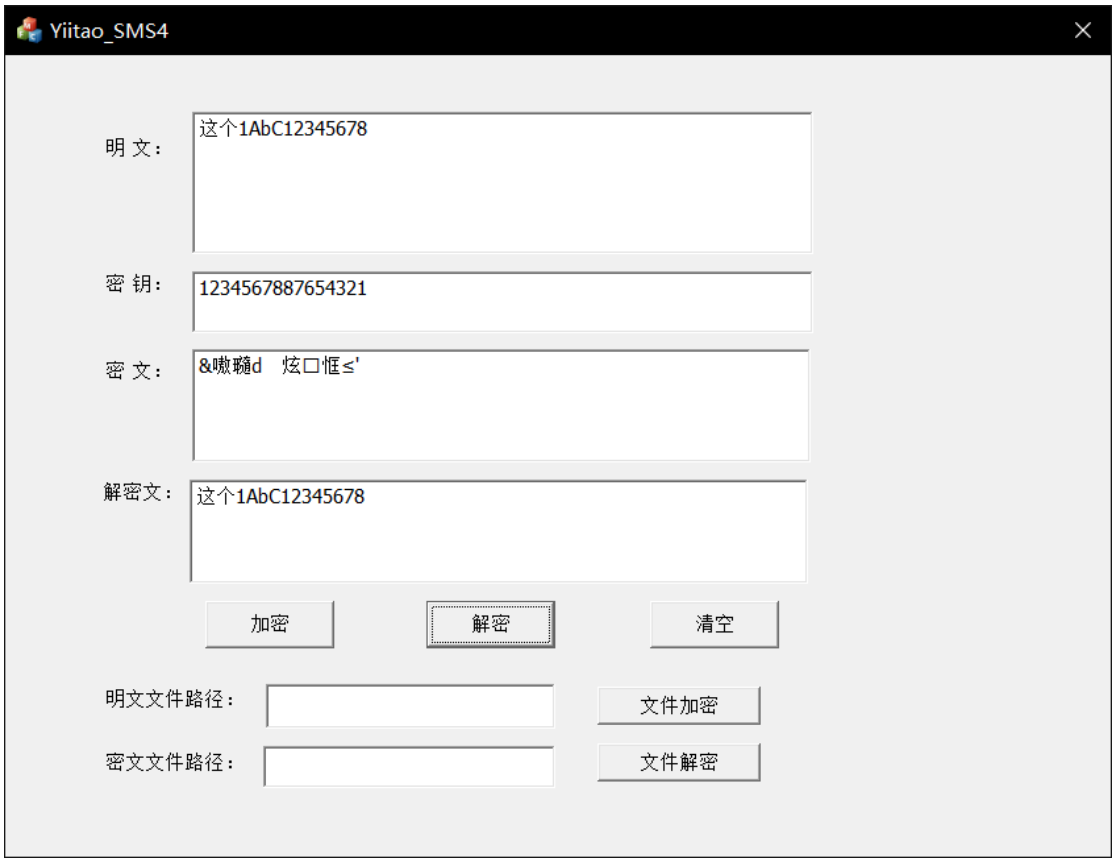


确定出错点之后，将明文通过其他方式转成十六进制与程序中对比，发现明文中汉字“这个”的十六进制对应是 0xd5e2 0xb8f6，而程序中转换成了 0xd4e1 0xb8f6。



于是进入程序对应位置代码，发现明文如图所示错误，输入的明文开始是 char 型，将 char 型转成 unsigned int 型过程中，应该将 char 转成 unsigned char 然后再进行移位，但是这里却直接将 char 移位，并没有考虑符号位的处理。于是改进代码，将 char_m 先&0xff 之后再进行移位，验证问题得到解决。

```
403     /* ... */
409
410     for (int i = 0; i < block - 1; i++)
411     {
412         for (int j = 0; j < 4; j++)
413             for (int t = 0; t < 4; t++)
414                 m[i][j] += (char_m[16 * i + 4 * j + t]/* & 0xff*/) << (24 - 8 * t);//本来中文加密错误，加了&0xff改正
415                                                                     //未加&之前，‘这个’的十六进制是d5e2 b8f6,未加的话，程序算成d4e2 b7f6
416     }
417     //最后一组处理，不需填充，/*m数组中初始化0，*/ 只需控制char_m不越界//出错导致16整数倍加解密错误
418     //memset(char_m + len, 0, 16 * block - len);  //出错，
419     for (int j = 0; j < 4; j++)
420     {
421         for (int t = 0; t < 4; t++)
422         {
423             if (16 * (block - 1) + 4 * j + t < len)//自动完成填充
424                 m[block - 1][j] += (char_m[16 * (block - 1) + 4 * j + t] /*& 0xff*/) << (24 - 8 * t);//本来中文加密错误，加了&0xff改正
425         }
426     }  已用时间 <= 1ms
```



2> 实验总结

本次实验为 SMS4 加密实验，相对于 DES 加密相对而言更加简单，但还是有一些地方与 DES 不同，如采用的变量类型，DES 是使用二进制串进行加解密，比较复杂，而 SMS4 使用 unsigned int 型变量，化繁为简，一下简化好多，因此我们编程前应该规划好，决定使用哪种数据结构，一个好的规划能避免很多错误。