实验项目名称： SM4密码算法 实验学时： 4

同组学生姓名： 无 实验地点： 科技楼4-1204

实验日期： 2023.9.20 实验成绩：

批改教师： 黄丹丹 批改时间： 2023.9.30

实验3 SM4密码算法

一、基于OBE模式的实验目的和要求

1、掌握SM4密码算法的基本原理

2、学会使用C/C++实现SM4密码算法

二、实验仪器和设备

Visual C/C++

三、实验过程

SM4密码是我国国家密码管理局公布的国内第一个商用密码标准，于2006年2月公布，是中国无线局域网安全标准推荐使用的分组密码算法。

**1、加密算法**

SM4算法的分组长度为128比特，密钥分组长度也为128比特。加密算法与密钥扩展算法都采用32轮迭代的Feistel结构，以字节（8位）和字（32位）为单位进行数据处理。

设输入的明文为四个字（128比特长），输入的轮密钥为，共32个字。输出的密文为四个字（128比特长）。加密算法可描述如下：

（1） 



（2） 

其中，*F*是轮函数，*T*是合成变换，*rki*是第*i*+1轮轮密钥，*R*是反序变换。

在加密算法之后还需要一个反序变换R，目的是使加密与解密流程保持一致，解密只需将加密密钥逆序使用。

合成变换*T*由非线性变换和线性变换*L*复合而成，设输入为字*X*，则先对*X* 进行非线性变换，再进行线性*L*变换。记为



非线性变换是由4个S盒并置构成，该S盒可以通过查表实现，见表1。设输入为（4个字节），输出为（4个字节），则



线性变换*L*是以字为处理单位的线性变换，其输入输出都是32位的字，设*L*的输入为字*B*，输出为字*C*，则



其中，为循环左移。

表1 SM4密码算法的S盒

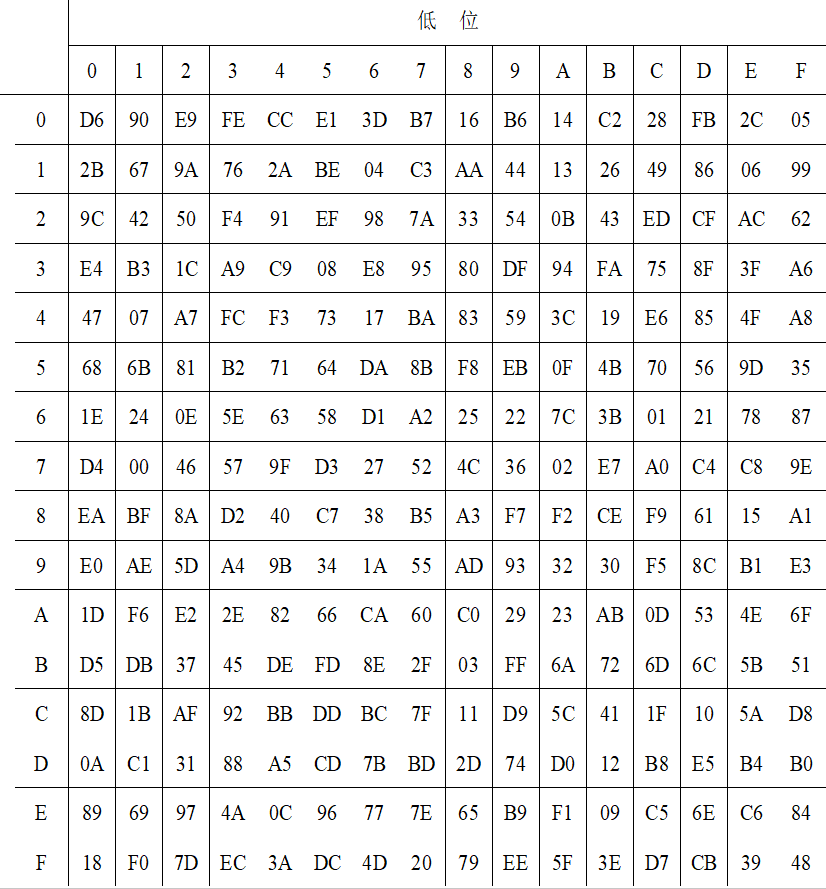




图2 SM4加密算法流程图

**2、解密算法**

SM4解密算法与加密算法相同，只是轮密钥的使用顺序相反，解密轮密钥是加密轮密钥的逆序。

算法的输入为密文和轮密钥，输出为明文。

**3、密钥扩展算法**

设输入的加密密钥为 ，输出轮密钥为，密钥扩展算法可描述如下，其中为中间数据：



变换**与加密算法轮函数中的*T*基本相同，只将其中的线性变化*L*修改为以下的**：



系统参数：（4个字）

*FK* =（*FK0, FK1, FK2, FK3*）=（a3b1bac6, 56aa3350, 677d9197, b27022dc）

固定参数：（32个字）

，

。

**4、编写设计SM4密码算法**

（1）输入128比特明文和密钥，利用SM4密码对明文加密并输出密文。

（2）输入SM4密码加密的128比特密文和密钥，对密文进行解密。

（3）要求有对应的程序调试记录和验证记录。

**提示点：****1、**首先需要定义SM4密码算法中的基本函数，循环左移、非线性变换（S盒）、加密中的线性变换*L*、密钥扩展算法中的线性变换**。部分函数参考代码如下：

（1）Rotl(x, y)定义为将32位的x循环左移y位

**#define Rotl(x, y) ((x << y) | (x >> (32 - y)))**

（2）ByteSub(A)为非线性变换，A是1个字，取A的4个字节分别进行S盒变换

**#define ByteSub(A) (Sbox[(A) >> 24 & 0xFF] << 24 | Sbox[(A) >> 16 & 0xFF] << 16 | Sbox[(A) >> 8 & 0xFF] << 8 | Sbox[(A) & 0xFF])**

（3）加密中的线性变换*L*

**#define L1(B) ((B) ^ Rotl(B, 2) ^ Rotl(B, 10) ^ Rotl(B, 18) ^ Rotl(B, 24))**

（4）密钥扩展算法中的线性变换**

**#define L2(B) ((B) ^ Rotl(B, 13) ^ Rotl(B, 23))**

**2、**所使用参数的初始定义：

（1）定义S盒

const unsigned char Sbox[256] = {

0xd6,0x90,0xe9,0xfe,0xcc,0xe1,0x3d,0xb7,0x16,0xb6,0x14,0xc2,0x28,0xfb,0x2c,0x05,

0x2b,0x67,0x9a,0x76,0x2a,0xbe,0x04,0xc3,0xaa,0x44,0x13,0x26,0x49,0x86,0x06,0x99,

0x9c,0x42,0x50,0xf4,0x91,0xef,0x98,0x7a,0x33,0x54,0x0b,0x43,0xed,0xcf,0xac,0x62,

0xe4,0xb3,0x1c,0xa9,0xc9,0x08,0xe8,0x95,0x80,0xdf,0x94,0xfa,0x75,0x8f,0x3f,0xa6,

0x47,0x07,0xa7,0xfc,0xf3,0x73,0x17,0xba,0x83,0x59,0x3c,0x19,0xe6,0x85,0x4f,0xa8,

0x68,0x6b,0x81,0xb2,0x71,0x64,0xda,0x8b,0xf8,0xeb,0x0f,0x4b,0x70,0x56,0x9d,0x35,

0x1e,0x24,0x0e,0x5e,0x63,0x58,0xd1,0xa2,0x25,0x22,0x7c,0x3b,0x01,0x21,0x78,0x87,

0xd4,0x00,0x46,0x57,0x9f,0xd3,0x27,0x52,0x4c,0x36,0x02,0xe7,0xa0,0xc4,0xc8,0x9e,

0xea,0xbf,0x8a,0xd2,0x40,0xc7,0x38,0xb5,0xa3,0xf7,0xf2,0xce,0xf9,0x61,0x15,0xa1,

0xe0,0xae,0x5d,0xa4,0x9b,0x34,0x1a,0x55,0xad,0x93,0x32,0x30,0xf5,0x8c,0xb1,0xe3,

0x1d,0xf6,0xe2,0x2e,0x82,0x66,0xca,0x60,0xc0,0x29,0x23,0xab,0x0d,0x53,0x4e,0x6f,

0xd5,0xdb,0x37,0x45,0xde,0xfd,0x8e,0x2f,0x03,0xff,0x6a,0x72,0x6d,0x6c,0x5b,0x51,

0x8d,0x1b,0xaf,0x92,0xbb,0xdd,0xbc,0x7f,0x11,0xd9,0x5c,0x41,0x1f,0x10,0x5a,0xd8,

0x0a,0xc1,0x31,0x88,0xa5,0xcd,0x7b,0xbd,0x2d,0x74,0xd0,0x12,0xb8,0xe5,0xb4,0xb0,

0x89,0x69,0x97,0x4a,0x0c,0x96,0x77,0x7e,0x65,0xb9,0xf1,0x09,0xc5,0x6e,0xc6,0x84,

0x18,0xf0,0x7d,0xec,0x3a,0xdc,0x4d,0x20,0x79,0xee,0x5f,0x3e,0xd7,0xcb,0x39,0x48

};

（2）定义固定参数（32个字）

const unsigned int CK[32] = {

0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,

0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,

0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,

0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,

0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,

0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,

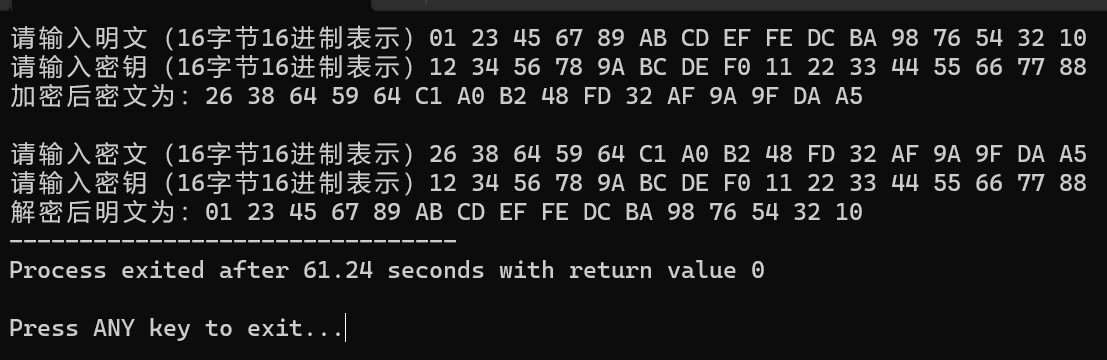
0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,

0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279 };

**5、程序代码：**

#include <stdio.h>#include <stdlib.h>#define Rotl(x, y) ((x << y) | (x >> (32 - y)))  
#define ByteSub(A) (Sbox[(A) >> 24 & 0xFF] << 24 | Sbox[(A) >> 16 & 0xFF] << 16 | Sbox[(A) >> 8 & 0xFF] << 8 | Sbox[(A) & 0xFF])  
#define L1(B) ((B) ^ Rotl(B, 2) ^ Rotl(B, 10) ^ Rotl(B, 18) ^ Rotl(B, 24))  
#define L2(B) ((B) ^ Rotl(B, 13) ^ Rotl(B, 23))  
  
const unsigned char Sbox[256] = {    0xd6,0x90,0xe9,0xfe,0xcc,0xe1,0x3d,0xb7,0x16,0xb6,0x14,0xc2,0x28,0xfb,0x2c,0x05,    0x2b,0x67,0x9a,0x76,0x2a,0xbe,0x04,0xc3,0xaa,0x44,0x13,0x26,0x49,0x86,0x06,0x99,    0x9c,0x42,0x50,0xf4,0x91,0xef,0x98,0x7a,0x33,0x54,0x0b,0x43,0xed,0xcf,0xac,0x62,    0xe4,0xb3,0x1c,0xa9,0xc9,0x08,0xe8,0x95,0x80,0xdf,0x94,0xfa,0x75,0x8f,0x3f,0xa6,    0x47,0x07,0xa7,0xfc,0xf3,0x73,0x17,0xba,0x83,0x59,0x3c,0x19,0xe6,0x85,0x4f,0xa8,    0x68,0x6b,0x81,0xb2,0x71,0x64,0xda,0x8b,0xf8,0xeb,0x0f,0x4b,0x70,0x56,0x9d,0x35,    0x1e,0x24,0x0e,0x5e,0x63,0x58,0xd1,0xa2,0x25,0x22,0x7c,0x3b,0x01,0x21,0x78,0x87,    0xd4,0x00,0x46,0x57,0x9f,0xd3,0x27,0x52,0x4c,0x36,0x02,0xe7,0xa0,0xc4,0xc8,0x9e,    0xea,0xbf,0x8a,0xd2,0x40,0xc7,0x38,0xb5,0xa3,0xf7,0xf2,0xce,0xf9,0x61,0x15,0xa1,    0xe0,0xae,0x5d,0xa4,0x9b,0x34,0x1a,0x55,0xad,0x93,0x32,0x30,0xf5,0x8c,0xb1,0xe3,    0x1d,0xf6,0xe2,0x2e,0x82,0x66,0xca,0x60,0xc0,0x29,0x23,0xab,0x0d,0x53,0x4e,0x6f,    0xd5,0xdb,0x37,0x45,0xde,0xfd,0x8e,0x2f,0x03,0xff,0x6a,0x72,0x6d,0x6c,0x5b,0x51,    0x8d,0x1b,0xaf,0x92,0xbb,0xdd,0xbc,0x7f,0x11,0xd9,0x5c,0x41,0x1f,0x10,0x5a,0xd8,    0x0a,0xc1,0x31,0x88,0xa5,0xcd,0x7b,0xbd,0x2d,0x74,0xd0,0x12,0xb8,0xe5,0xb4,0xb0,    0x89,0x69,0x97,0x4a,0x0c,0x96,0x77,0x7e,0x65,0xb9,0xf1,0x09,0xc5,0x6e,0xc6,0x84,    0x18,0xf0,0x7d,0xec,0x3a,0xdc,0x4d,0x20,0x79,0xee,0x5f,0x3e,0xd7,0xcb,0x39,0x48  
  
};const unsigned int CK[32] = {    0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,    0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,    0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,    0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,    0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,    0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,    0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,    0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279  
};void KeyExpansion(unsigned int\* rk, unsigned char\* key){    unsigned int K[36];    unsigned int MK[4];    unsigned int tmp;    int i;      
    for (i = 0; i < 4; ++i) {        MK[i] = (key[4 \* i] << 24) | (key[4 \* i + 1] << 16) | (key[4 \* i + 2] << 8) | key[4 \* i + 3];    }      
    K[0] = MK[0] ^ 0xA3B1BAC6;    K[1] = MK[1] ^ 0x56AA3350;    K[2] = MK[2] ^ 0x677D9197;    K[3] = MK[3] ^ 0xB27022DC;      
    for (i = 0; i < 32; ++i) {        tmp = K[i + 1] ^ K[i + 2] ^ K[i + 3] ^ CK[i];        tmp = ByteSub(tmp);        K[i + 4] = K[i] ^ L2(tmp);        rk[i] = K[i + 4];    }}unsigned int F(unsigned int X0){    unsigned int t, y;      
    t = X0 ^ Rotl(X0, 2) ^ Rotl(X0, 10) ^ Rotl(X0, 18) ^ Rotl(X0, 24);    y = ByteSub(t);    y ^= Rotl(t, 13) ^ Rotl(t, 23);      
    return y;}void Encrypt(unsigned char\* input, unsigned char\* output, unsigned int\* rk){    unsigned int X[36];    unsigned int i, t, tmp;      
    for (i = 0; i < 4; ++i) {        X[i] = (input[4 \* i] << 24) | (input[4 \* i + 1] << 16) | (input[4 \* i + 2] << 8) | input[4 \* i + 3];    }      
    for (i = 0; i < 32; ++i) {        t = X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rk[i];        tmp = F(t);        X[i + 4] = X[i] ^ tmp;    }      
    for (i = 0; i < 4; ++i) {        output[4 \* i] = (X[35 - i] >> 24) & 0xFF;        output[4 \* i + 1] = (X[35 - i] >> 16) & 0xFF;        output[4 \* i + 2] = (X[35 - i] >> 8) & 0xFF;        output[4 \* i + 3] = X[35 - i] & 0xFF;    }}void Decrypt(unsigned char\* input, unsigned char\* output, unsigned int\* rk){    unsigned int X[36];    unsigned int i, t, tmp;      
    for (i = 0; i < 4; ++i) {        X[i] = (input[4 \* i] << 24) | (input[4 \* i + 1] << 16) | (input[4 \* i + 2] << 8) | input[4 \* i + 3];    }      
    for (i = 0; i < 32; ++i) {        t = X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rk[31 - i];        tmp = F(t);        X[i + 4] = X[i] ^ tmp;    }      
    for (i = 0; i < 4; ++i) {        output[4 \* i] = (X[35 - i] >> 24) & 0xFF;        output[4 \* i + 1] = (X[35 - i] >> 16) & 0xFF;        output[4 \* i + 2] = (X[35 - i] >> 8) & 0xFF;        output[4 \* i + 3] = X[35 - i] & 0xFF;    }}int main(){    unsigned int rk[32];    printf("请输入明文（16字节16进制表示）");    unsigned char plaintext[16];    for(int i=0;i<16;i++){        scanf("%hhx", &plaintext[i]);    }    unsigned char ciphertext[16];    unsigned char decryptedtext[16];    printf("请输入密钥（16字节16进制表示）");    unsigned char key[16];    for(int i=0;i<16;i++){        scanf("%hhx", &key[i]);    }    KeyExpansion(rk, key);      
    Encrypt(plaintext, ciphertext, rk);      
    printf("加密后密文为：");    for (int i = 0; i < 16; ++i) {        printf("%02X ", ciphertext[i]);    }    printf("\n");    unsigned char ciphertext2[16];    printf("\n请输入密文（16字节16进制表示）");    for (int i = 0; i < 16; ++i) {        scanf("%hhx", &ciphertext2[i]);    }      
    printf("请输入密钥（16字节16进制表示）");    unsigned char key2[16];    for(int i=0;i<16;i++){        scanf("%hhx", &key2[i]);    }      
    unsigned int rk2[32];    KeyExpansion(rk2, key2);    Decrypt(ciphertext, decryptedtext, rk2);      
    printf("解密后明文为：");    for (int i = 0; i < 16; ++i) {        printf("%02X ", decryptedtext[i]);    }      
    return 0;}四、实验结果与分析

测试结果截图与分析如下：



先输入16字节（128位）明文，再输入16字节密钥，经过sm4算法后加密成16字节的密文，再把加密出来的密文输入进去，输入相同的密钥，经过解密后解出原文。

五、基于OBE模式的学生自我评价与体会

（该实验对自身分析、设计、思辨、创新等个人综合能力与素质的影响）

本次实验，我们尝试用C语言实现SM4算法，SM4是国标商用密码，实现难度比前几次实验要难得多，但是好在老师给了不少提示。在实验过程中，我按照给定的步骤进行了加密算法、解密算法和密钥扩展算法的编写设计，并进行了相应的调试和验证。通过这个实验，我对SM4密码算法有了更深入的理解。我了解到SM4算法是一种具有高安全性和高效率的分组密码算法。我熟悉了SM4算法的加密和解密过程，以及密钥扩展算法的实现方法。