《网络空间安全导论实践》报告

题目： AES密码的加密解密实验

姓名：

学院： 网络空间安全学院

专业： 网络空间安全（大类招生）

班级： 2023211805

学号：

联系方式：

邮箱：

2024年5月12日

|  |
| --- |
| 基本信息表 |
| 题目：AES密码的加密解密实验 |
| 作品类别：场景重现 |
| 作品内容摘要：本实验旨在研究和实现AES（Advanced Encryption Standard）密码算法的加解密过程。AES是一种对称密钥加密算法，被广泛应用于保护数据的安全性，其安全性和效率性在很大程度上受到认可。实验首先介绍了AES算法的基本原理，包括其四种不同的密钥长度（128位、192位和256位），以及其密钥扩展、轮密钥生成和轮函数等关键概念。然后，利用编程语言（C）实现了AES算法的加解密过程，并通过实验验证了其正确性和可靠性。实验中还探讨了AES算法在不同密钥长度下的性能表现，并对其加解密速度和资源消耗进行了评估和比较。最后，通过实验结果分析了AES算法在数据安全领域的应用前景和挑战。 |
| 关键词：AES密码算法、对称密钥加密、密钥长度、密钥扩展、轮密钥生成、轮函数、加密、解密、性能评估、数据安全。 |
| 作品提交形式：*（程序、源代码、文档、视频等）*  文档、视频、ppt |

# 作品背景及意义

**1.1 作品背景**

随着信息技术的快速发展，网络安全和数据保护变得越来越重要。在数据传输和存储过程中，保护数据的机密性和完整性对于个人、企业和国家的利益至关重要。因此，加密算法成为了信息安全领域中的关键技术之一。在对称加密算法中，AES（Advanced Encryption Standard）因其高度安全性和效率性而备受青睐，已成为了许多安全应用和协议的首选加密算法之一。

**1.2 作品意义**

通过对AES算法的学习和实现，加深对其基本原理和内部机制的理解，提升密码学知识和加密算法设计能力。

实验实现的AES加密算法可以应用于保护敏感数据的安全传输和存储，提高数据的机密性和完整性，有效防止数据泄露和篡改。

作为一种经典的对称加密算法，AES的研究和实现有助于推动信息安全技术的发展，促进安全通信和数据保护领域的进步。

本实验可作为密码学、网络安全和信息安全等相关课程的教学实践内容，培养学生的加密算法设计和实现能力，促进学术交流和合作。

人们思考更复杂的加密算法和密码学技术，促进信息安全领域的发展和进步。

# 作品介绍

本实验旨在探索AES密码的加密解密过程，并通过实际操作加深对密码学基本概念的理解。通过一系列步骤将明文转换为密文，或将密文还原为明文。本实验将通过编写程序实现AES密码的加密和解密过程，并分析其原理及应用。

1. **设计与实现方案**

**3.1 技术原理**

**3.1.1 密码学相关概念与原理**

（1）加密与解密：加密是将明文转换为密文的过程，而解密则是将密文还原为明文的过程。这两个过程通常依赖于密钥和特定的加密算法。

（2）对称加密与非对称加密：对称加密使用相同的密钥来进行加密和解密，而非对称加密使用一对密钥，即公钥和私钥。公钥用于加密，私钥用于解密。

（3）哈希函数：哈希函数将任意长度的输入数据映射为固定长度的输出，称为哈希值。哈希函数通常用于验证数据的完整性，因为相同的输入总是产生相同的哈希值。

（4）数字签名：数字签名用于验证和确认数据的完整性、来源和认可。它使用私钥对数据进行签名，而公钥用于验证签名的有效性。

（5）公钥基础设施（PKI）：PKI 是一组技术和政策，用于管理数字证书、公钥和相关的身份验证过程，以确保安全的通信和身份认证。

（6）密钥管理：密钥管理涉及生成、分发、存储和更新加密和解密所需的密钥。它是确保系统安全性的重要组成部分。

（7）安全协议：安全协议是在通信过程中使用的一系列规则和步骤，用于确保数据的安全性和隐私性，例如 SSL/TLS 协议用于安全的网站通信。

（8）侧信道攻击：侧信道攻击是通过分析加密系统的物理实现（如功耗、电磁辐射等）来获取密钥或明文的一种攻击方式。

**3.1.2 密码学目标、应用及其安全性考量**

（1）密码学的目标

保密性：保证信息在传输和存储过程中不被未授权的人或系统访问。

完整性：确保信息在传输和存储过程中不被篡改。

认证：确认信息的发送者和接收者的身份。

不可否认性：发送者不能否认发送过的信息。

（2）密码学的应用

数据加密：在通信和存储中使用加密技术来保护敏感信息的安全性。

数字签名：用于验证和确认文档或数据的完整性和来源。

身份认证：使用加密技术来验证用户的身份，确保只有授权用户可以访问特定的系统或服务。

（3）密码学安全性考量

密钥长度和复杂性：更长和更复杂的密钥增加了密码的安全性。

算法的选择：使用被广泛接受和经过充分评估的加密算法，如AES（高级加密标准）等，以确保安全性。

**3.1.3 AES密码操作**

AES加密算法涉及5种操作：秘钥扩展、字节替代、行移位、列混淆和轮密钥加。

秘钥扩展：给定的初始秘钥一般比较短，比如16字节，而算法如果进行10轮运算的话就需要16x(10+1)字节长度的秘钥，需要对原始秘钥进行秘钥扩展。

字节替代：通过非线性的替换函数，用查找表的方式把每个字节替换成对应的字节。

行移位：将矩阵中的每个横列进行循环式移位。

列混淆：为了充分混合矩阵中各个直行的操作。这个步骤使用线性转换来混合每列的四个字节。

轮密钥加：矩阵中的每一个字节都与该次轮秘钥做异或运算；每个子密钥由密钥生成方案产生。

**3.2设计与实现方案**

**3.2.1 实验设计**

首以加密函数为例，如下所示，首先对密钥进行预处理密钥扩展，然后明文进行Nr（Nr与密钥长度有关）次迭代运算，包括字节代换ShiftRows行移位、MixColumns、列混合以及AddRoundKey密钥加。

void Cipher()

{

int i,j,round=0;

// 把明文赋值到状态数组中

for(i=0;i<4;i++)

for(j=0;j<4;j++)

state[j][i] = in[i\*4 + j];

// 先与初始轮密钥相加

AddRoundKey(0);

// 第一轮至（Nr-1）轮的迭代运算，第Nr轮不用进行列混合运算

for(round=1;round<Nr;round++)

{

SubBytes(); //字节代换

ShiftRows(); //行移位

MixColumns(); //列混合

AddRoundKey(round); //密钥加

}

SubBytes();

ShiftRows();

AddRoundKey(Nr);

// 加密结束，将机密结果填入数组out中以便输出

for(i=0;i<4;i++)

for(j=0;j<4;j++)

out[i\*4+j]=state[j][i];

}

解密函数的流程和加密函数是一致的，只是对于行变换、列变换、以及相关s盒子为加密的逆过程。

**3.2.2 概念简介**

**3.2.2.1字节替换SubByte**

非线性代换是可逆的，由以下两个变换的合成得到：

首先，将字节看作GF(28)上的元素，映射到自己的乘法逆元，‘00’映射到自己。其次，对字节做如下仿射变换：



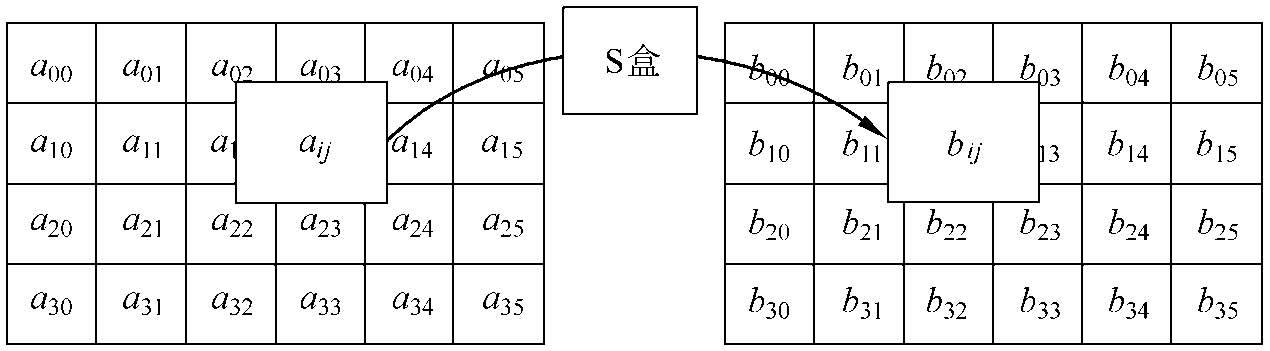


图 1 字节代换示意图

**3.2.2.2行移位ShiftRow**

将状态阵列的各行进行循环移位，移位量与分组长度的关系

0行：不动

|  |  |  |  |
| --- | --- | --- | --- |
| *Nb* | *C1* | *C2* | *C3* |
| *4* | *1* | *2* | *3* |
| *6* | *1* | *2* | *3* |
| *8* | *1* | *3* | *4* |

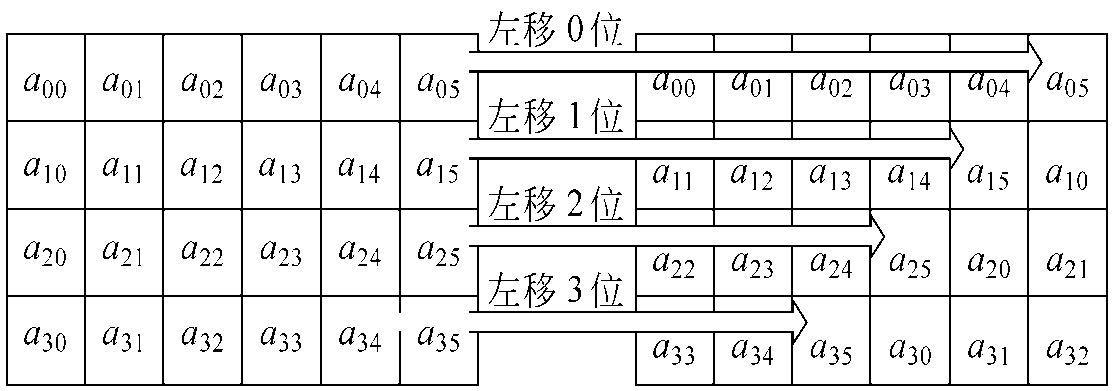


图2 行移位示意图

**3.2.2.3 列混合MixColumn**

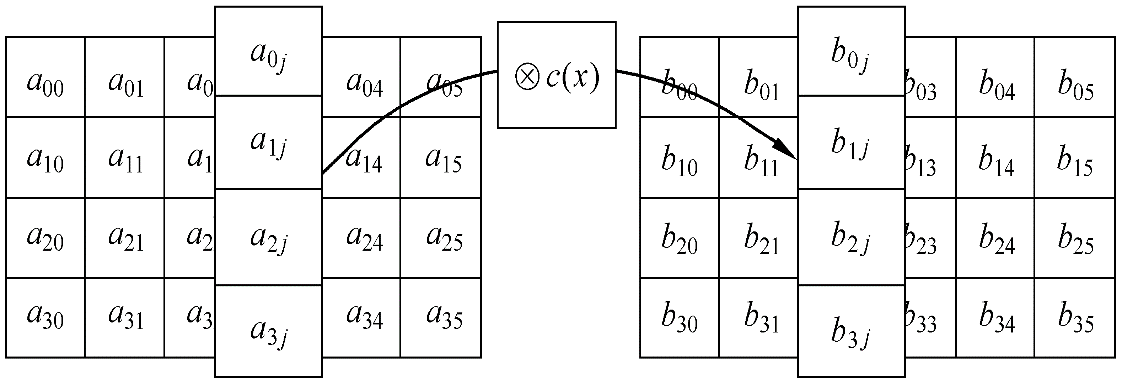


图3 列混合示意图

状态阵列的每个列a(x)与一个固定的多项式c(x)进行模x4+1乘法后混淆为b(x).记为

c(x)是模x4+1可逆的多项式‘03’x3+‘01’x2+‘01’x+‘02’

逆d(x)=‘0B’x3+‘0D’x2+‘09’x+‘0E’



**3.2.2.4 轮密钥加AddRoundKey**

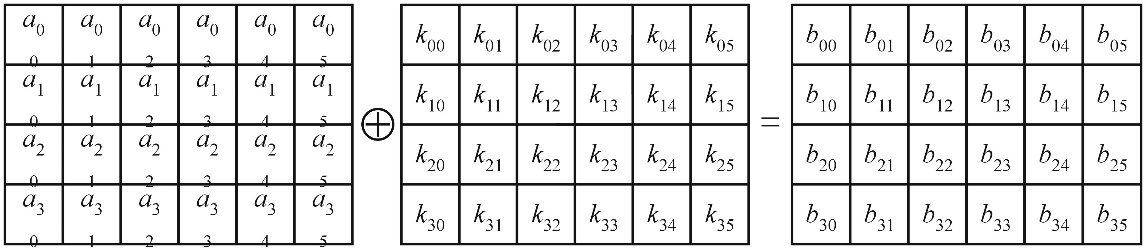


图4 轮密钥加示意图

密钥加:轮密钥与状态进行逐比特异或。轮密钥长度等于分组长度Nb。

密钥加运算的逆运算是其自身。

**3.2.2.5 逆字节替换**

通过逆S盒的映射变换得到

**3.2.2.6逆行移位InvShiftRow**

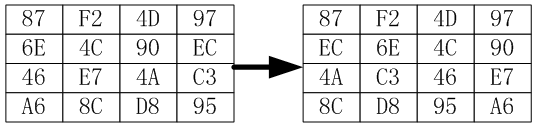


图5 逆行移位示意图

与加密时的行移位区别在于移位方向相反。

**3.2.2.7 逆列混淆**

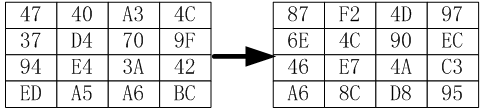


图6 逆列混合示意图

加密与解密系统流程图如下所示，

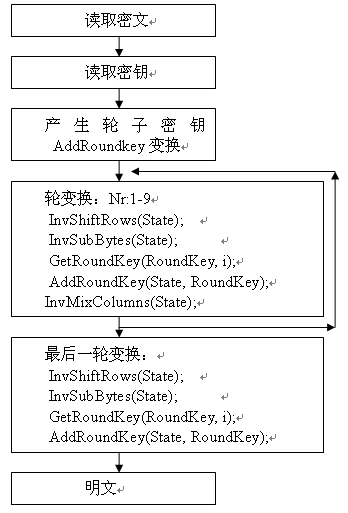


图7 加密解密流程图

**3.2.3 环境介绍**

（1）语言：C语言

（2）编译环境：Visual C++ 6.0

（3）需安装软件：Visual C++ 6.0

**3.2.4 实验代码实现方案**

#include <stdio.h>

static const unsigned char sbox[256]={

0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,

0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,

0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,

0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,

0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,

0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,

0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,

0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,

0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,

0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,

0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,

0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,

0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,

0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,

0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,

0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,

0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,

0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,

0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,

0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,

0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,

0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,

0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,

0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,

0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,

0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,

0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,

0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,

0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,

0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,

0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,

0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16,

};

//逆向S 盒矩阵

static const unsigned char contrary\_sbox[256]={

0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,

0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb,

0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,

0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb,

0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,

0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e,//0x4e

0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,

0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25,

0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,

0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92,

0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,

0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84,

0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,

0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06,

0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,

0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b,

0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,

0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73,

0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,

0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e,

0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,

0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b,

0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,

0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4,

0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,

0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f,

0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,

0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef,

0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,

0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61,

0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,

0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d,

};

/\*轮常量表 The key schedule rcon table\*/

static const unsigned char Rcon[10]={

0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36};

//辅助函数

/\*有限域\*2乘法 The x2time() function \*/

static unsigned char x2time(unsigned char x)

{

if (x&0x80)

{

return (((x<<1)^0x1B)&0xFF);

}

return x<<1;

}

/\*有限域\*3乘法 The x2time() function \*/

static unsigned char x3time(unsigned char x)

{

return (x2time(x)^x);

}

/\*有限域\*4乘法 The x4time() function \*/

static unsigned char x4time(unsigned char x)

{

return ( x2time(x2time(x)) );

}

/\*有限域\*8乘法 The x8time() function \*/

static unsigned char x8time(unsigned char x)

{

return ( x2time(x2time(x2time(x))) );

}

/\*有限域9乘法 The x9time() function \*/

static unsigned char x9time(unsigned char x) //9:1001

{

return ( x8time(x)^x );

}

/\*有限域\*B乘法 The xBtime() function \*/

static unsigned char xBtime(unsigned char x) //B:1011

{

return ( x8time(x)^x2time(x)^x );

}

/\*有限域\*D乘法 The xDtime() function \*/

static unsigned char xDtime(unsigned char x) //D:1101

{

return ( x8time(x)^x4time(x)^x );

}

/\*有限域\*E乘法 The xEtime() function \*/

static unsigned char xEtime(unsigned char x) //E:1110

{

return ( x8time(x)^x4time(x)^x2time(x) );

}

/\*第三类操作：列混合操作 MixColumns: Process the entire block\*/

static void MixColumns(unsigned char \*col)//列混合

{

unsigned char tmp[4],xt[4];

int i;

for(i=0;i<4;i++,col+=4) //col代表一列的基地址，col+4:下一列的基地址

{

tmp[0]=x2time(col[0])^x3time(col[1])^col[2]^col[3]; //2 3 1 1

tmp[1]=col[0]^x2time(col[1])^x3time(col[2])^col[3]; //1 2 3 1

tmp[2]=col[0]^col[1]^x2time(col[2])^x3time(col[3]); //1 1 2 3

tmp[3]=x3time(col[0])^col[1]^col[2]^x2time(col[3]); //3 1 1 2

//修改后的值 直接在原矩阵上修改

col[0]=tmp[0];

col[1]=tmp[1];

col[2]=tmp[2];

col[3]=tmp[3];

}

}

//逆向列混淆

static void Contrary\_MixColumns(unsigned char \*col)

{

unsigned char tmp[4];

unsigned char xt2[4];//colx2

unsigned char xt4[4];//colx4

unsigned char xt8[4];//colx8

int x;

for(x=0;x<4;x++,col+=4)

{

tmp[0]=xEtime(col[0])^xBtime(col[1])^xDtime(col[2])^x9time(col[3]);

tmp[1]=x9time(col[0])^xEtime(col[1])^xBtime(col[2])^xDtime(col[3]);

tmp[2]=xDtime(col[0])^x9time(col[1])^xEtime(col[2])^xBtime(col[3]);

tmp[3]=xBtime(col[0])^xDtime(col[1])^x9time(col[2])^xEtime(col[3]);

col[0]=tmp[0];

col[1]=tmp[1];

col[2]=tmp[2];

col[3]=tmp[3];

}

}

/\*第二类操作：行移位：行左循环移位 ShiftRows:Shifts the entire block\*/

static void ShiftRows(unsigned char \*col)//正向行移位

{

unsigned char t;

/\*1nd row\*///左移1位

t=col[1];col[1]=col[5];col[5]=col[9];col[9]=col[13];col[13]=t;

/\*2rd row\*///左移2位，交换2次数字来实现

t=col[2];col[2]=col[10];col[10]=t;

t=col[6];col[6]=col[14];col[14]=t;

/\*3th row\*///左移3位，相当于右移1次

t=col[15];col[15]=col[11];col[11]=col[7];col[7]=col[3];col[3]=t;

/\*4th row\*/ //第4行不移位

}

//逆向行移位

static void Contrary\_ShiftRows(unsigned char \*col)

{

unsigned char t;

/\*1nd row\*/

t=col[13];col[13]=col[9];col[9]=col[5];col[5]=col[1];col[1]=t;

/\*2rd row\*/

t=col[2];col[2]=col[10];col[10]=t;

t=col[6];col[6]=col[14];col[14]=t;

/\*3th row\*/

t=col[3];col[3]=col[7];col[7]=col[11];col[11]=col[15];col[15]=t;

/\*4th row\*/ //第4行不移位

}

/\*第一类操作：s盒字节代换替换 SubBytes\*/

static void SubBytes(unsigned char \*col)//字节代换

{

int x;

for(x=0;x<16;x++)

{

col[x]=sbox[col[x]];

}

}

//逆向字节代换

static void Contrary\_SubBytes(unsigned char \*col)

{

int x;

for(x=0;x<16;x++)

{

col[x]=contrary\_sbox[col[x]];

}

}

/\*第四类操作：轮密钥加 AddRoundKey\*/

static void AddRoundKey(unsigned char \*col,unsigned char \*expansionkey,int round)//密匙加

{

int x;

for(x=0;x<16;x++) //每1轮操作：4\*32bit密钥 = 16个字节密钥

{

col[x]^=expansionkey[(round<<4)+x];

}

}

/\* AES加密总函数 10轮4类操作 Encrypt a single block with Nr Rounds(10,12,14)\*/

void AesEncrypt(unsigned char \*blk,unsigned char \*expansionkey,int Nr)//加密一个区块

{

int round;

//第1轮之前：轮密钥加

AddRoundKey(blk,expansionkey,0);

//第1-9轮：4类操作：字节代换、行移位、列混合、轮密钥加

for(round=1;round<=(Nr-1);round++)

{

SubBytes(blk); //输入16字节数组，直接在原数组上修改

ShiftRows(blk); //输入16字节数组，直接在原数组上修改

MixColumns(blk); //输入16字节数组，直接在原数组上修改

AddRoundKey(blk,expansionkey,round);

}

//第10轮：不进行列混合

SubBytes(blk);

ShiftRows(blk);

AddRoundKey(blk,expansionkey,Nr);

}

//AES 解密总函数

void Contrary\_AesEncrypt(unsigned char \*blk,unsigned char \*expansionkey,int Nr)

{

int x;

AddRoundKey(blk,expansionkey,Nr);

Contrary\_ShiftRows(blk);

Contrary\_SubBytes(blk);

for(x=(Nr-1);x>=1;x--)

{

AddRoundKey(blk,expansionkey,x);

Contrary\_MixColumns(blk);

Contrary\_ShiftRows(blk);

Contrary\_SubBytes(blk);

}

AddRoundKey(blk,expansionkey,0);

}

void ScheduleKey(unsigned char \*inkey,unsigned char \*outkey,int Nk,int Nr)//安排一个保密密钥使用

{

unsigned char temp[4],t;

int x,i;

/\*copy the key\*/

//第0组：[0-3]直接拷贝

for(i=0;i<(4\*Nk);i++)

{

outkey[i]=inkey[i];

}

//第1-10组：[4-43]

i=Nk;

while(i<(4\*(Nr+1))) //i=4~43 WORD 32bit的首字节地址，每一个4字节

{

for(x=0;x<4;x++)

temp[x]=outkey[(4\*(i-1))+x]; //i：32bit的首字节地址

//i是4的倍数的时候

if(i%Nk==0)

{

/\*字循环：循环左移1字节 RotWord()\*/

t=temp[0];temp[0]=temp[1];temp[1]=temp[2];temp[2]=temp[3];temp[3]=t;

/\*字节代换：SubWord()\*/

for(x=0;x<4;x++)

{

temp[x]=sbox[temp[x]];

}

/\*轮常量异或：Rcon[j]\*/

temp[0]^=Rcon[(i/Nk)-1];

}

for(x=0;x<4;x++)

{

outkey[(4\*i)+x]=outkey[(4\*(i-Nk))+x]^temp[x];

}

++i;

}

}

int main(void){

unsigned char pt[17],key[17];

unsigned char expansionkey[15\*16];

int i;

int j;

printf("请输入明文：\n");//输入无格式的字符串字符个数不得少于六个！！！！尽量用不同的数字或字母

scanf("%s",pt);

printf("请输入秘钥：\n");//输入加密钥匙密匙个数不得低于六个！！！！尽量用不同的数字或字母

scanf("%s",key);

/\*加密\*/

ScheduleKey(key,expansionkey,4,10); //1、密钥扩展生成

AesEncrypt(pt,expansionkey,10); //2、AES 加密

printf("加密文本为;: "); //输出密码文件

for (i = 0; i < 16; i++)

{

printf("%02x ", pt[i]);

}

printf("\n");

printf("\n");

/\*解密\*/

Contrary\_AesEncrypt(pt,expansionkey,10);//AES 解密

printf("在逆序加密后，明文为: ");//将解密文件输出

for (i = 0; i < 16; i++)

{

printf("%c ", pt[i]);

}

printf("\n");

printf("\n");

while(1);

return 0;

}

**4.实现过程中遇到的问题和解决**

问题：在AES加密解密过程中，密钥的管理和安全性是至关重要的。如何安全地生成、存储和传输密钥是一个挑战，特别是在实际应用中需要考虑密钥的保护和管理。

解决方法：我采用了一种简单而有效的密钥管理方案，即在代码中硬编码密钥。虽然这种方法不够安全，但在学习和实验阶段可以接受。在实际应用中，应该采用更加安全的密钥管理方案，如使用密钥管理服务（KMS）或硬件安全模块（HSM）来保护密钥。

问题：AES算法要求明文的长度必须是块大小（通常为128位）的整数倍。因此，在加密过程中，如果明文长度不是块大小的整数倍，就需要进行填充操作。

解决方法：我选择了使用标准的PKCS7填充方案来填充明文。PKCS7填充方案是一种常见且简单的填充方案，通过在明文末尾添加填充字节，使得明文长度恰好为块大小的整数倍。

**5.结论**

这次实验，代码是从密钥扩展开始写的。因为这一部分是相对独立的。列混合运算这个过程要注意溢出是要即使异或0X1B。实验代码相对比DES的要多，原因我想是AES加解密过程是不对称的，每个加密过程都要写一个相对应的解密过程才行，而DES却是加密解密一视同仁，都是进行一种操作行为，只是解密的过程密钥是加密过程的逆序。

通过本次实验，我对AES密码的加密解密过程有了更深入的理解，并学习到了如何在实际项目中应用对称加密算法。同时，我也意识到了在实现加密算法时需要考虑的安全性和效率等问题，以及如何选择合适的加密库和填充方案。

总的来说，本次实验为我提供了一个良好的学习机会，使我更加熟悉了密码学和加密算法的基本概念，也提高了我的编程能力和解决问题的能力。在未来的学习和工作中，我将继续深入研究加密算法，并努力提高自己在信息安全领域的专业水平。