南开大学
Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

密码学 (1091) 第三次实验

分组密码算法 AES

苗发生

年级：2020 级

专业：信息安全 & 法学双学位

指导教师：古力

2022 年 11 月 26 日

目录

一、 实验内容

(一) 实验目的

通过用 AES 算法对实际的数据进行加密和解密来深刻了解 AES 的运行原理。

(二) 实验环境

运行 Windows 操作系统的 PC 机，具有 VC 等语言编译环境

(三) 实验内容和步骤

1) 算法分析：对课本中 AES 算法进行深入分析，对其中用到的基本数学算法、字节代换、行移位变换、列混合变换原理进行详细的分析，并考虑如何进行编程实现。对轮函数、密钥生成等环节要有清晰的了解，并考虑其每一个环节的实现过程

2) AES 实现程序的总体设计：在第一步的基础上，对整个 AES 加密函数的实现进行总体设计，考虑数据的存储格式，参数的传递格式，程序实现的总体层次等，画出程序实现的流程图

3) 在总体设计完成后，开始具体的编码，在编码过程中，注意要尽量使用高效的编码方式

4) 利用 3 中实现的程序，对 AES 的密文进行雪崩效应检验。即固定密钥，仅改变明文中的一位，统计密文改变的位数；固定明文，仅改变密钥中的一位，统计密文改变的位数

(四) AES 算法介绍

AES 是高级加密标准（英语：Advanced Encryption Standard）的缩写，在密码学中又称 Rijndael 加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的 DES，已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院（NIST）于 2001 年 11 月 26 日发布于 FIPS PUB 197，并在 2002 年 5 月 26 日成为有效的标准。2006 年，高级加密标准已然成为对称密钥加密中最流行的算法之一

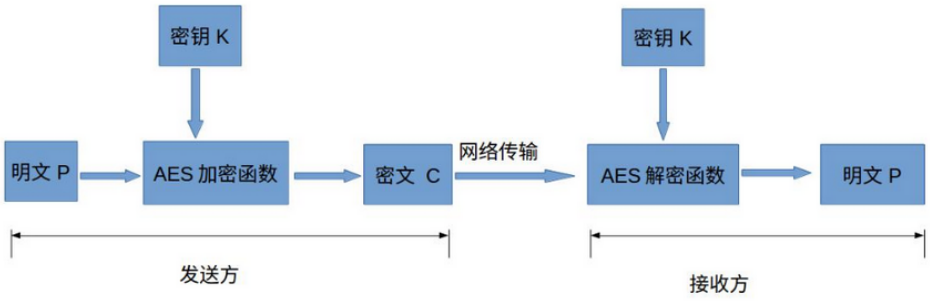

图 1: 使用 AES 的流程

(五) 实验原理

AES 算法本质上是一种对称分组密码体制，采用代替/置换网络，每轮由三层组成：线性混合层确保多轮之上的高度扩散，非线性层由 16 个 S 盒并置起到混淆的作用，密钥加密层将子密

钥异或到中间状态。Rijndael 是一个迭代分组密码，其分组长度和密钥长度都是可变的，只是为了满足 AES 的要求才限定处理的分组大小为 128 位，而密钥长度为 128 位、192 位或 256 位，相应的迭代轮数 N，为 10 轮、12 轮、14 轮。AES 汇聚了安全性能、效率、可实现性、灵活性等优点。最大的优点是可以给出算法的最佳差分特征的概率，并分析算法抵抗差分密码分析及线性密码分析的能力。

加密的主要过程包括：对明文状态的一次密钥加，$N_r - 1$ 轮轮加密和末尾轮轮加密，最后得到密文。其中 $N_r - 1$ 轮轮加密每一轮有四个部件，包括字节代换部件 ByteSub、行移位变换 ShiftRow、列混合变换 NixColumn 和一个密钥加 AddRoundKey 部件，末尾轮加密和前面轮加密类似，只是少了一个列混合变换 NixColumn 部件。解密过程与加密过程类似。

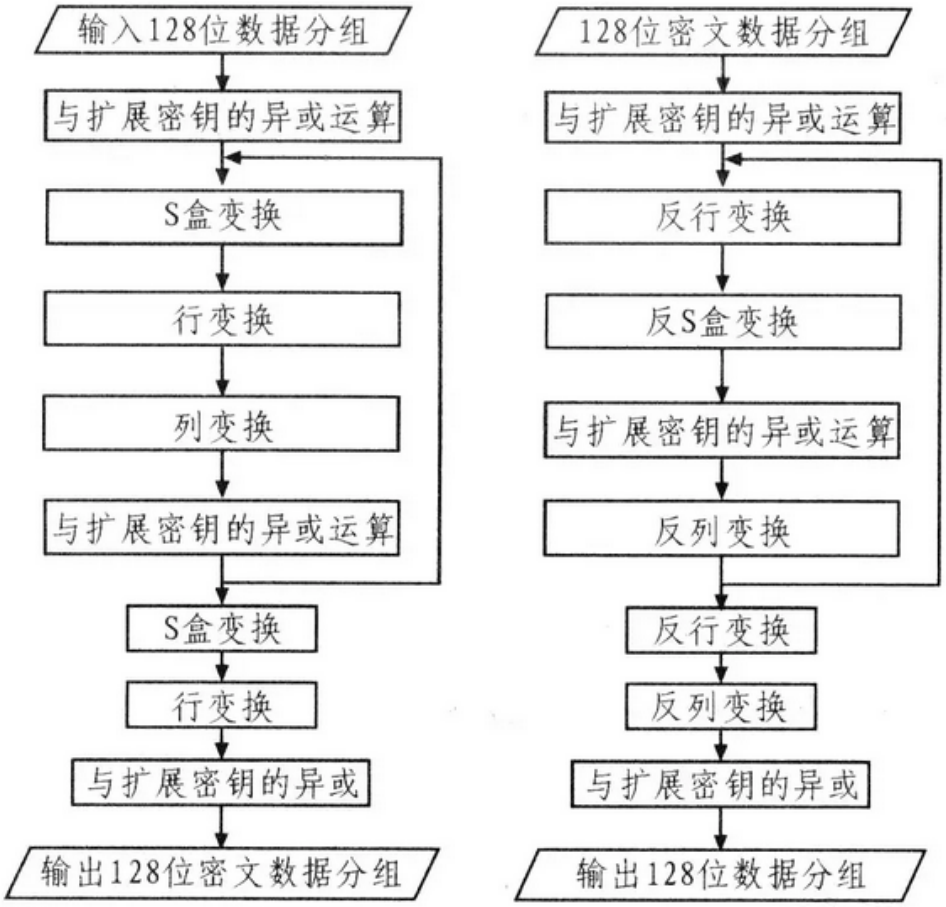

图 2: AES 算法加密和解密

（六） 实验报告和要求

1) 实现 AES 的加密和解密，提交程序代码和执行结果

2) 在检验雪崩效应中，要求至少改变明文和密文中各八位，给出统计结果并计算出平均值

二、 实验具体流程

(一) 算法分析

1. 字节替换 ByteSub 部件

字节代换是非线形变换，独立地对状态的每个字节进行。代换表（即 S-盒）是可逆的，由以下两个变换的合成得到：

1) 首先，将字节看作 $GF(2^8)$ 上的元素，映射到自己的乘法逆元，‘00’映射到自己

2) 其次，对字节做如下的（GF(2) 上的，可逆的）仿射变换：

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

图 3: 仿射变换

字节代换整个过程的示意图如下所示：

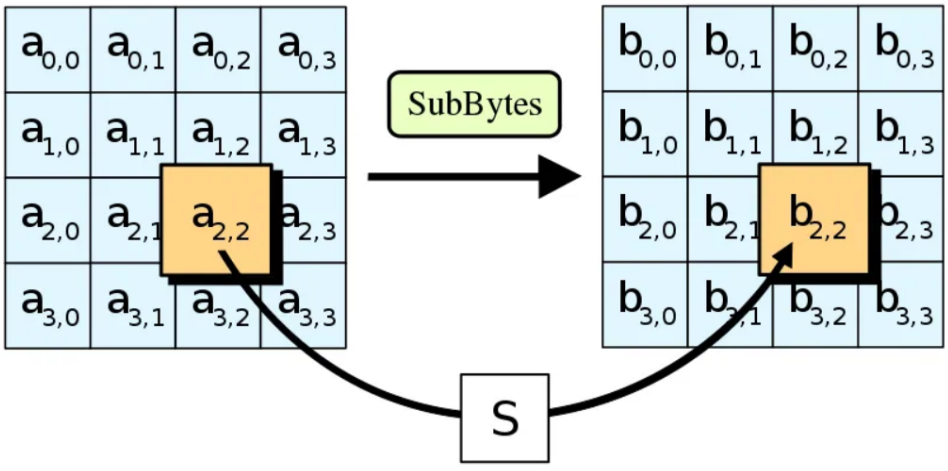

图 4: 字节替换

注意：该部件的逆运算部件就是先对自己做一个逆仿射变换，然后映射到自己的乘法逆元上

2. 行移位变换 ShiftRow

行移位是将状态阵列的各行进行循环移位，不同状态行的位移量不同。第 0 行不移动，第 1 行循环左移 C_1 个字节，第 2 行循环左移 C_2 个字节，第 3 行循环左移 C_3 个字节。位移量 C_1、C_2、C_3 的取值与 N_b 有关，对应于不同长度的位移量如下表所示：

N_b	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

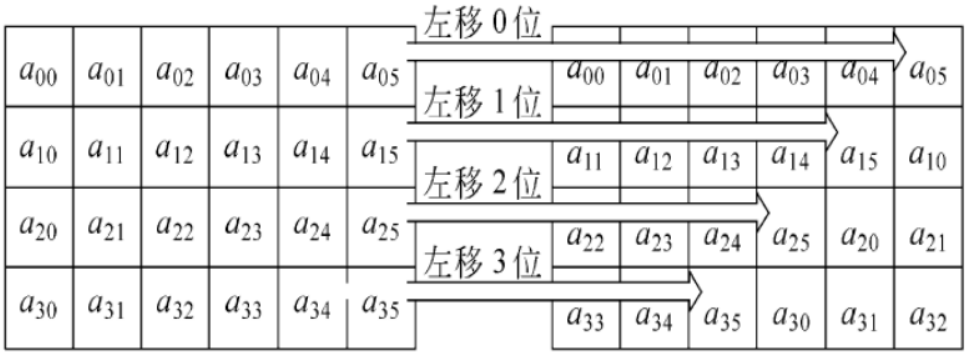

图 5: 行移位示意图

ShiftRow 的逆变换是对状态阵列的后 3 列分别以位移量 N_b-C_1、N_b-C_2、N_b-C_3 进行循环移位，使得第 i 行第 j 列的字节移位到

$$(j + N_b - C_i) mod N_b$$

3. 列混合变换 NixColumn

在列混合变换中，将状态阵列的每个列视为系数为 $GF(2^8)$ 上的多项式，再与一个固定的多项式 c(x) 进行模 x^4+1 乘法。当然要求 c(x) 是模 x^4+1 可逆的多项式，否则列混合变换就是不可逆的，因而会使不同的输入分组对应的输出分组可能相同。Rijndael 的设计者给出的 c(x) 为（系数用十六进制数表示）：

$$c(x) = 03x^3 + 01x^2 + 01x + 02$$

c(x) 是与 $x^4 + 1$ 互素的，因此是模 $x^4 + 1$ 可逆的。列混合运算也可写为矩阵乘法。设 b(x)= c(x)a(x)，则

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

图 6: 列混合中的矩阵乘法

这个运算需要做 GF(2^8) 上的乘法，但由于所乘的因子是 3 个固定的元素 02、03、01，所以这些乘法运算也是比较简单的。列混合运算的整体示意图如下：

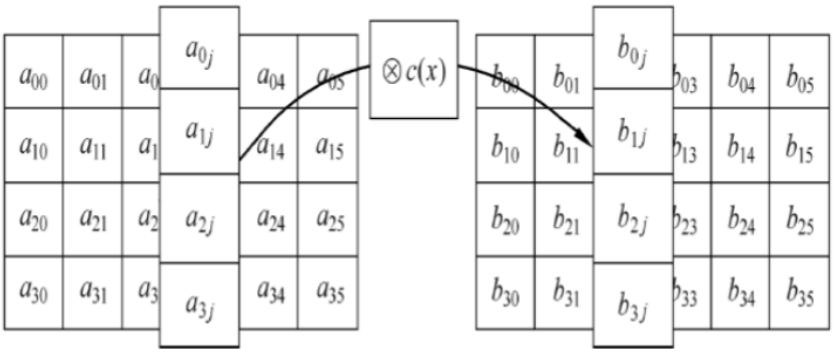

图 7: 列混合运算示意图

注意：列混合运算的逆运算是类似的，即每列都用一个特定的多项式 d(x) 相乘，d(x) 满足如下

$$03x^3 + 01x^2 + 01x + 02 \otimes d(x) = 01$$

由此可得

$$d(x) = 0Bx^3 + 0Dx^2 + 09x + 0E$$

4. 密钥加 AddRoundKey 部件

密钥加是将轮密钥简单地与状态进行逐比特异或。轮密钥由种子密钥通过密钥编排算法得到，轮密钥长度等于分组长度 N_b。密钥加运算的逆运算是其自身。

密钥加运算示意图如下：

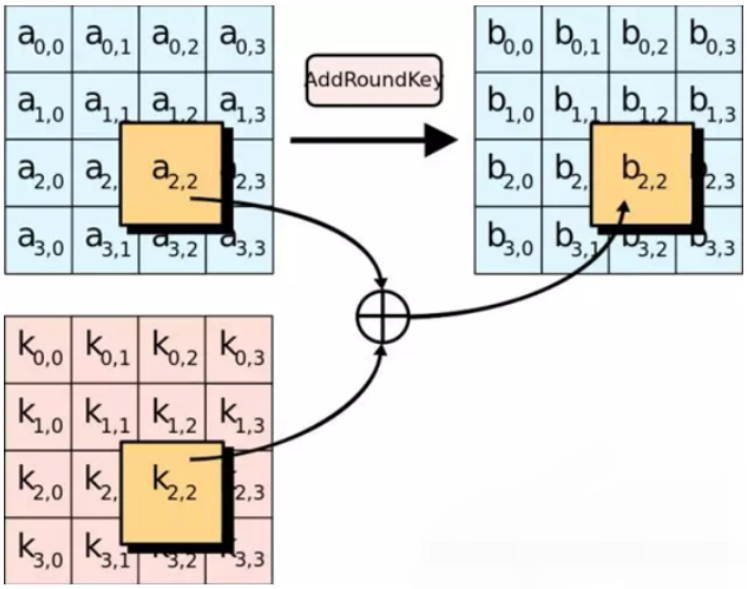

图 8: 密钥加运算示意图

5. 密钥编排

密钥编排指从种子密钥得到轮密钥的过程，它由密钥扩展和轮密钥选取两部分组成。其基本原则如下：(1) 轮密钥的字数（4 比特 32 位的数）等于分组长度乘以轮数加 1；(2) 种子密钥被扩展成为扩展密钥；(3) 轮密钥从扩展密钥中取，其中第 1 轮轮密钥取扩展密钥的前 N_b 个字，第 2 轮轮密钥取接下来的 N_b 个字，如此下去。

6. 解密过程

AES 算法的解密过程和加密过程是相似的，也是先经过一个密钥加，然后进行 $N_r - 1$ 轮轮解密和末尾轮轮解密，最后得到明文。和加密不同的是 $N_r - 1$ 轮轮解密每一轮四个部件都需要用到它们的逆运算部件，包括字节代换部件的逆运算、行移位变换的逆变换、逆列混合变换和一个密钥加部件，末尾轮加密和前面轮加密类似，只是少了一个逆列混合变换部件。

在解密的时候，还要注意轮密钥和加密密钥的区别，设加密算法的初始密钥加、第 1 轮、第 2 轮、…、第 Nr 轮的子密钥依次为 k(0), k(1), k(2), …, k(N_r-1), k(N_r) 则解密算法的初始密钥加、第 1 轮、第 2 轮、…、第 Nr 轮的子密钥依次为 k(N_r), InvMixColumn (k(N_r-1)), InvMixColumn (k(N_r-2)), …, InvMixColumn (k(1)), k(0)。

(二) AES 算法整体设计

1. 头文件的设计

本次 AES 加密、解密、修改明文观察雪崩效应和修改密钥观察雪崩效应，需要一些固定常量（S 盒、逆 S 盒）以及一些功能函数（数字转二进制字符串、二进制字符串转十进制、GF(28) 的多项式乘法, 寻找数字的最高位，GF(28) 的多项式除法, 扩展欧几里得算法,AES 的乘法运算）,本次实验我将这些工具性质的常量和函数，封装到头文件中，在实现 AES 加密、解密、修改明文观察雪崩效应和修改密钥观察雪崩效应的时候，只需要引用该头文件即可！

2. 参数的传递格式

在本实验中，我们用 int txt[16][16] 的数组分别存储 128 位的明文和密钥，在实现修改明文观察雪崩效应和修改密钥观察雪崩效应的时候，由于是对 bit 进行的修改，我们预先实现 NumtoString () 和 StringtoNum () 函数，将存储的明文或者密钥转化为二进制字符串，进而实现对单个 bit 的修改。

3. AES 算法加解密的整过程

AES-128，也就是密钥的长度为 128 位，加密轮数为 10 轮。AES 的加密公式为 C = E(K,P)，在加密函数 E 中，会执行一个轮函数，并且执行 10 次这个轮函数，这个轮函数的前 9 次执行的操作是一样的，只有第 10 次有所不同。也就是说，一个明文分组会被加密 10 轮。AES 的核心就是实现一轮中的所有操作。

1) 字节代换

a) 字节代换操作: AES 的字节代换其实就是一个简单的查表操作。AES 定义了一个 S 盒和一个逆 S 盒

b) 字节代换逆操作: 逆字节代换也就是查逆 S 盒来变换

2) 行移位

a) 行移位操作: 行移位是一个简单的左循环移位操作

b) 行移位的逆变换: 行移位的逆变换是将状态矩阵中的每一行执行相反的移位操

3) 列混合

列混合操作: 混合变换是通过矩阵相乘来实现的，经行移位后的状态矩阵与固定的矩阵相乘，得到淆后的状态矩阵

4) 轮密钥加

轮密钥加是将 128 位轮密钥 Ki 同状态矩阵中的数据进行逐位异或操作

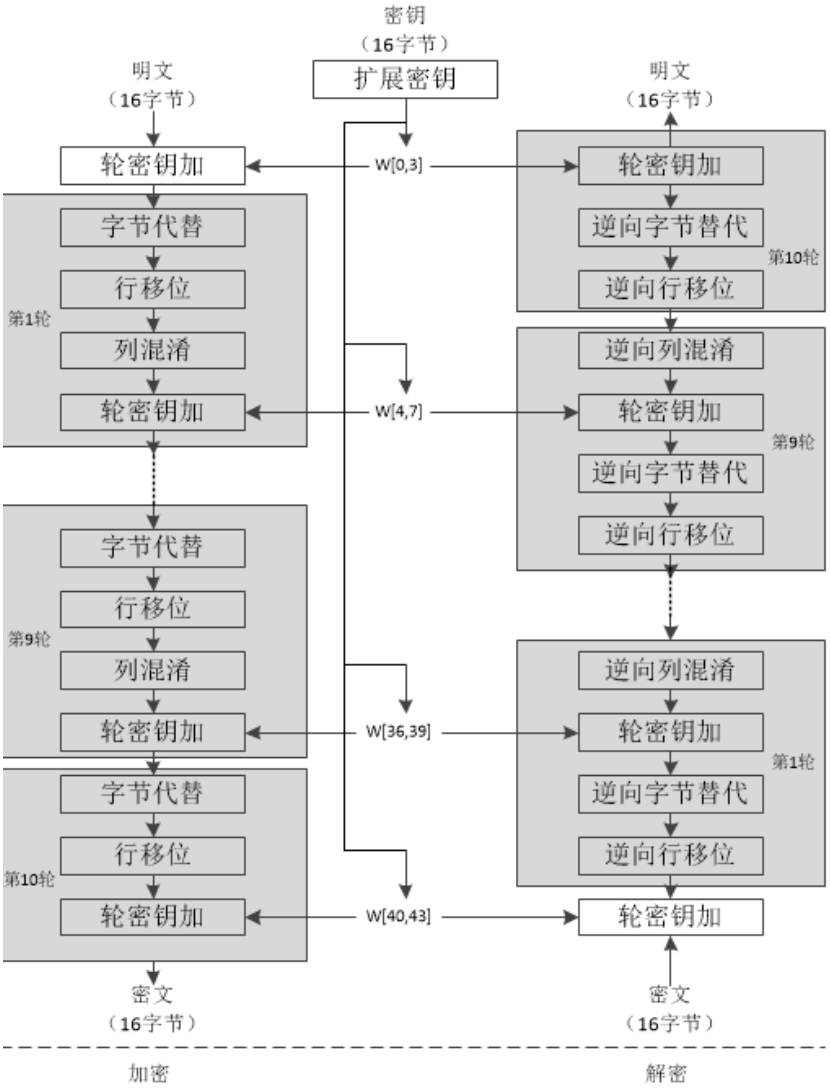

图 9: AES 算法加解密整体设计流程图

(三) 关键功能函数

1. 扩展欧几里得算法

扩展欧几里得算法是最优的求逆元算法之一，它和费马小定理具有同样的时间复杂度 O (l o g (n)) O(log(n)) O(log(n))，但是费马小定理需要模数为质数，扩展欧几里得算法则不需要

扩展欧几里得算法

```
//扩展欧几里得算法
int externEuclid(int a, int m)
{
        int r0, r1, r2, r3, r4, r5, r6, r7, r8, r9;
        r0 = m;
        r1 = a;
        r4 = 1;
```

```
        r5 = 0;
        r7 = 0;
        r8 = 1;
        while (r1 != 1)
        {
                r3 = Division(r0, r1, &r2);
                r6 = r4 ^ Multiplication(r3, r5);
                r9 = r7 ^ Multiplication(r3, r8);
                r0 = r1;
                r1 = r2;
                r4 = r5;
                r5 = r6;
                r7 = r8;
                r8 = r9;
        }
        return r8;
}
```

2. AES 的乘法运算

AES 的乘法运算

```
//AES的乘法运算
int Mult(int a, int b)
{


        int third = b & 0x8;   //b&1000
        int second = b & 0x4;   //b&0100
        int first = b & 0x2;   //b&0010
        int firstMod = b % 2;     //b&0001
        int res = 0;
        if (third)
        {
                int temp = a;
                for (int i = 1; i <= 3; ++i)
                {
                        temp = temp << 1;     //总共移3位
                        if (temp >= 256)
                        {
                                temp = temp ^ 0x11b;    //x^8+x^4+x^3+x+1
                        }
                }
                temp = temp % 256;
                res = res ^ temp;
        }
        if (second)
        {
```

```
                int temp = a;
                for (int i = 1; i <= 2; ++i)
                {
                        temp = temp << 1;       //总共移2位
                        if (temp >= 256)
                        {
                                temp = temp ^ 0x11b;
                        }
                }
                temp = temp % 256;
                res = res ^ temp;
        }
        if (first)
        {
                int temp = a;
                temp = temp << 1;           //总共移1位
                if (temp >= 256)
                {
                        temp = temp ^ 0x11b;
                }
                temp = temp % 256;
                res = res ^ temp;
        }
        if (firstMod)
        {
                res = res ^ a;              //不用移位
        }
        return res;
}
```

3. 密钥扩展

1 密钥扩展的实现流程为:

a) 将种子密钥按图预定格式排列，其中 k0、k1、……、k15 依次表示种子密钥的一个字节；排列后用 4 个 32 比特的字表示，分别记为 w[0]、w[1]、w[2]、w[3]

b) 依次求解 w[j]，其中 j 是整数并且属于 [4,43]

c) 若 j % 4=0, 则 $w[j] = w[j-4] \bigoplus (w[j-1])$, 否则 $w[j] = w[j-4] \bigoplus w[j-1]$

密钥扩展

```
//密钥扩展
void keyExpansion(int Key[4][4], int W[11][4][4])
{
        for (int i = 0; i < 4; ++i)
        {
                for (int j = 0; j < 4; j++)
                {
```

```
                W[0][i][j] = Key[j][i];     //key读进来的时候是竖着存的，现在要横着存

            }
        }
        for (int i = 1; i < 11; ++i)
        {
            for (int j = 0; j < 4; ++j)
            {
                int temp[4];
                if (j == 0)     //即mod4==0
                {
                    //先进行字节为单位的循环左移位
                    temp[0] = W[i - 1][3][1];
                    temp[1] = W[i - 1][3][2];
                    temp[2] = W[i - 1][3][3];
                    temp[3] = W[i - 1][3][0];
                    for (int k = 0; k < 4; ++k)
                    {
                        int m = temp[k];
                        int row = m / 16;
                        int col = m % 16;
                        temp[k] = S[row][col];
                        if (k == 0)
                        {
                            //与轮常量异或
                            temp[k] = temp[k] ^ rC[i - 1];
                        }
                    }
                }
                else      //mod4!=0
                {
                    temp[0] = W[i][j - 1][0];
                    temp[1] = W[i][j - 1][1];
                    temp[2] = W[i][j - 1][2];
                    temp[3] = W[i][j - 1][3];
                }
                //mod!=4时，下面的结果为：W[i]=W[i-4] W[i-1]
                //mod==4时，下面的结果为：W[i-4] T(W[i-1])
                for (int x = 0; x < 4; x++)
                {
                    W[i][j][x] = W[i - 1][j][x] ^ temp[x];
                }
            }
        }
}
```

4. 字节代换

字节代换是非线形变换，独立地对状态的每个字节进行。代换表（即 S-盒）是可逆的，由以下两个变换的合成得到：

1) 首先，将字节看作 $GF(2^8)$ 上的元素，映射到自己的乘法逆元，‘00’映射到自己

2) 其次，对字节做（GF(2) 上的，可逆的）仿射变换

字节代换

```
void byteSub(int in[4][4], int type)//解密type=0, 加密type=1
{
        for (int i = 0; i < 4; i++)
        {
                for (int j = 0; j < 4; j++)
                {
                        int temp = in[i][j];
                        int row = temp / 16;
                        int col = temp % 16;
                        if (type == 1)
                        {
                                in[i][j] = S[row][col];
                        }
                        if (type == 0)
                        {
                                in[i][j] = rS[row][col];
                        }
                }
        }
}
```

5. 行移位

行移位是将状态阵列的各行进行循环移位，不同状态行的位移量不同。第 0 行不移动，第 1 行循环左移 C_1 个字节，第 2 行循环左移 C_2 个字节，第 3 行循环左移 C_3 个字节。位移量 C_1、C_2、C_3 的取值与 N_b 有关，对应于不同长度的位移量如下表所示：

N_b	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

根据这些步骤，实现的 C++ 代码如下：

行移位

```
// 行移位，Nb=4时用三角的方式实现移位结果
void shiftRow(int txt[4][4], int type) //解密type=0, 加密type=1
{
        for (int row = 0; row < 4; row++)
```

```
        {
                for (int col = 0; col < row; col++)
                {
                        if (type == 1)
                        {
                                int temp = txt[row][0];
                                txt[row][0] = txt[row][1];
                                txt[row][1] = txt[row][2];
                                txt[row][2] = txt[row][3];
                                txt[row][3] = temp;
                        }
                        else    //type == 0
                        {
                                int temp = txt[row][3];
                                txt[row][3] = txt[row][2];
                                txt[row][2] = txt[row][1];
                                txt[row][1] = txt[row][0];
                                txt[row][0] = temp;
                        }
                }
        }
}
```

6. 列混淆

在列混合变换中，将状态阵列的每个列视为系数为 $GF(2^8)$ 上的多项式，再与一个固定的多项式 c(x) 进行模 x^4+1 乘法。当然要求 c(x) 是模 x^4+1 可逆的多项式，否则列混合变换就是不可逆的，因而会使不同的输入分组对应的输出分组可能相同。Rijndael 的设计者给出的 c(x) 为（系数用十六进制数表示）：

$$c(x) = 03x^3 + 01x^2 + 01x + 02$$

c(x) 是与 $x^4 + 1$ 互素的，因此是模 $x^4 + 1$ 可逆的。

列混淆

```
void mixColumn(int txt[4][4], int type)
{
        for (int col = 0; col < 4; col++)
        {
                int t0 = txt[0][col];
                int t1 = txt[1][col];
                int t2 = txt[2][col];
                int t3 = txt[3][col];
                if (type == 1) //加密
                {
                        txt[0][col] = Mult(t0, 2) ^ Mult(t1, 3) ^ t2 ^ t3;
                        txt[1][col] = t0 ^ Mult(t1, 2) ^ Mult(t2, 3) ^ t3;
                        txt[2][col] = t0 ^ t1 ^ Mult(t2, 2) ^ Mult(t3, 3);
                        txt[3][col] = Mult(t0, 3) ^ t1 ^ t2 ^ Mult(t3, 2);
```

```
                }
                else //type==0 解密
                {
                        txt[0][col] = Mult(t0, 14) ^ Mult(t1, 11) ^ Mult(t2,
                            13) ^ Mult(t3, 9);
                        txt[1][col] = Mult(t0, 9) ^ Mult(t1, 14) ^ Mult(t2,
                            11) ^ Mult(t3, 13);
                        txt[2][col] = Mult(t0, 13) ^ Mult(t1, 9) ^ Mult(t2,
                            14) ^ Mult(t3, 11);
                        txt[3][col] = Mult(t0, 11) ^ Mult(t1, 13) ^ Mult(t2,
                            9) ^ Mult(t3, 14);
                }
        }
}
```

7. 密钥加

密钥加（AddRoundKey）是将轮密钥简单地与状态进行逐比特异或，轮密钥由种子密钥通过密钥编排获得，轮密钥长度等于分组密钥长度

密钥加

```
void addRoundKey(int txt[4][4], int key[4][4])
{
        for (int i = 0; i < 4; ++i)
        {
                for (int j = 0; j < 4; j++)
                {
                        txt[i][j] = txt[i][j] ^ key[j][i];
                }
        }
}
```

(四) AES 加密和解密

1. AES 加密

AES-128，也就是密钥的长度为 128 位，加密轮数为 10 轮。AES 的加密公式为 C = E(K,P)，在加密函数 E 中，会执行一个轮函数，并且执行 10 次这个轮函数，这个轮函数的前 9 次执行的操作是一样的，只有第 10 次有所不同。也就是说，一个明文分组会被加密 10 轮，每一轮需要依次执行字节替换、行移位、列混淆以及密钥加，C++ 实现如下：

AES 加密

```
        bool flag;
        do {
                cout << "请输入128位的明文" << endl;
                for (int i = 0; i < 4; i++)
                {
                        for (int j = 0; j < 4; j++)
```

```
                {
                        cin >> (hex) >> txt[j][i];
                }
        }
        cout << "请输入128位的密钥: " << endl;
        for (int i = 0; i < 4; i++)
        {
                for (int j = 0; j < 4; j++)
                {
                        cin >> (hex) >> key[j][i];
                }
        }
        keyExpansion(key, subKey);
        addRoundKey(txt, subKey[0]);        //轮密钥加
        for (int i = 1; i <= 10; ++i)
        {
                byteSub(txt, 1);
                shiftRow(txt, 1);
                if (i != 10)        //最后一次计算不需要列混合
                {
                        mixColumn(txt, 1);
                }
                addRoundKey(txt, subKey[i]);
        }
        cout << "加密结果为: " << endl;
        for (int i = 0; i < 4; i++)
        {
                for (int j = 0; j < 4; j++)
                {
                        if (txt[j][i] < 16)
                                cout << "0";
                        cout << (hex) << txt[j][i] << " ";
                }
        }
        cout << endl;
        cout << "是否继续进行加密，输入1则继续加密，输入0退出程序" <<
             endl;
        cin >> flag;
} while (flag);
```

2. AES 解密

AES 算法的解密过程和加密过程是相似的，也是先经过一个密钥加，然后进行 $N_r - 1$ 轮轮解密和末尾轮轮解密，最后得到明文。和加密不同的是 $N_r - 1$ 轮轮解密每一轮四个部件都需要用到它们的逆运算部件，包括字节代换部件的逆运算、行移位变换的逆变换、逆列混合变换和一个密钥加部件，末尾轮加密和前面轮加密类似，只是少了一个逆列混合变换部件，C++ 实现的代码如下：

AES 解密

```
        bool flag;
        do {
                cout << "请输入128位的密文" << endl;
                for (int i = 0; i < 4; i++)
                {
                        for (int j = 0; j < 4; j++)
                        {
                                cin >> (hex) >> txt[j][i];
                        }
                }
                cout << "请输入128位的密钥: " << endl;
                for (int i = 0; i < 4; i++)
                {
                        for (int j = 0; j < 4; j++)
                        {
                                cin >> (hex) >> key[j][i];
                        }
                }

                int subKey[11][4][4];
                keyExpansion(key, subKey);
                addRoundKey(txt, subKey[10]);
                for (int i = 9; i >= 0; --i)
                {
                        byteSub(txt, 0);
                        shiftRow(txt, 0);
                        addRoundKey(txt, subKey[i]);          //subkey与加密相
                            反
                        if (i != 0)
                        {
                                mixColumn(txt, 0);
                        }
                }
                cout << "解密结果为: " << endl;
                for (int i = 0; i < 4; i++)
                {
                        for (int j = 0; j < 4; j++)
                        {
                                if (txt[j][i] < 16)
                                        cout << "0";
                                cout << (hex) << txt[j][i] << " ";
                        }
                }
                cout << endl;
                cout << "是否继续进行解密，输入1则继续解密，输入0退出程序" <<
                    endl;
                cin >> flag;
```

```
46          } while (flag);
```

（五） 检验雪崩效应

1. 改变明文

由于明文中的每一位均是内容，故我们更改 128 位明文的每一位观察雪崩现象，更改明文观察雪崩现象的过程中，密钥并未进行更改可以复用，故我们只需更改明文后重新利用密钥进行加密即可

更改明文观察雪崩

```
1           cout << "雪崩测试：修改明文~" << endl;
2           string result = NumtoString(txt_new);
3           for (int i = 0; i < 128; i++) {
4                   if (i % 4 == 1)
5                           cout << endl;
6                   cout << "明文第" << (i + 1) << "位改变：";
7                   if (result[i] == '0')
8                           result[i] = '1';
9                   else
10                          result[i] = '0';
11                  StringtoNum(txt_new, result);
12                  //==========================利用修改后的明文进行加密
                        ==========================
13                  keyExpansion(key, subKey);
14                  addRoundKey(txt_new, subKey[0]);        //轮密钥加
15                  for (int i = 1; i <= 10; ++i)
16                  {
17                          byteSub(txt_new, 1);
18                          shiftRow(txt_new, 1);
19                          if (i != 10)        //最后一次计算不需要列混合
20                          {
21                                  mixColumn(txt_new, 1);
22                          }
23                          addRoundKey(txt_new, subKey[i]);
24                  }
25                  //==========================利用修改后的明文进行加密
                        ==========================
26                  int result_text = 0;
27                  string s_new = NumtoString(txt_new);
28                  string s = NumtoString(txt);
29                  for (int i = 0; i < 128; i++) {
30                          if (s_new[i] == s[i]) {
31                                  result_text++;
32                          }
33                  }
34                  cout << "密文改变";
35                  cout << (dec) << result_text << "位      ";
```

```
                ans += result_text;
                if (result[i] == '0')
                        result[i] = '1';
                else
                        result[i] = '0';
        }
        cout << endl << "依次修改明文的每一位，平均每次改变" << ans / 128 <<
            "位" << endl;
```

2. 改变密钥

由于密钥的 128 位均为有效位，每次更改完密钥后，均需要利用更改后的密钥重新进行加密（即进行 AES 加密步骤），再依次统计更改每一位后的雪崩效应，核心代码如下：

更改密钥观察雪崩

```
        cout << "雪崩测试：修改密钥~" << endl;
        string result = NumtoString(key_new);
        for (int i = 0; i < 128; i++) {
                if (i % 4 == 1)
                        cout << endl;
                cout << "密钥第" << (i + 1) << "位改变：";
                if (result[i] == '0')
                        result[i] = '1';
                else
                        result[i] = '0';
                StringtoNum(key_new, result);
                //=========================利用修改后的密钥进行加密
                    ==========================
                keyExpansion(key_new, subKey);
                addRoundKey(txt_new, subKey[0]);        //轮密钥加
                for (int i = 1; i <= 10; ++i)
                {
                        byteSub(txt_new, 1);
                        shiftRow(txt_new, 1);
                        if (i != 10)        //最后一次计算不需要列混合
                        {
                                mixColumn(txt_new, 1);
                        }
                        addRoundKey(txt_new, subKey[i]);
                }
                //=========================利用修改后的密钥进行加密
                    ==========================
                int result_text = 0;
                string s_new = NumtoString(txt_new);
                string s = NumtoString(txt);
                for (int i = 0; i < 128; i++) {
                        if (s_new[i] == s[i]) {
                                result_text++;
```

```
32                          }
33                  }
34                  cout << "密文改变";
35                  cout << (dec) << result_text << "位      ";
36                  ans += result_text;
37                  if (result[i] == '0')
38                          result[i] = '1';
39                  else
40                          result[i] = '0';
41          }
42          cout << endl << "依次修改密钥的每一位，平均每次改变" << ans / 128 <<
                "位" << endl;
```

（六） 实验结果演示

1. AES 加密

本次实验 AES 加密，AES 解密，更改明文观察雪崩效应以及更改密文观察雪崩效应四个我均单独实现了一套代码，只需要运行对应的 exe 文件，按照命令行提示进行输入即可，现对老师给的样例进行测试，本次展示的测试样例为：

样例一：

密钥：00 01 00 01 01 a1 98 af da 78 17 34 86 15 35 66

明文：00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94

密文：6c dd 59 6b 8f 56 42 cb d2 3b 47 98 1a 65 42 2a

样例二：

密钥：32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

明文：2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

密文：39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32

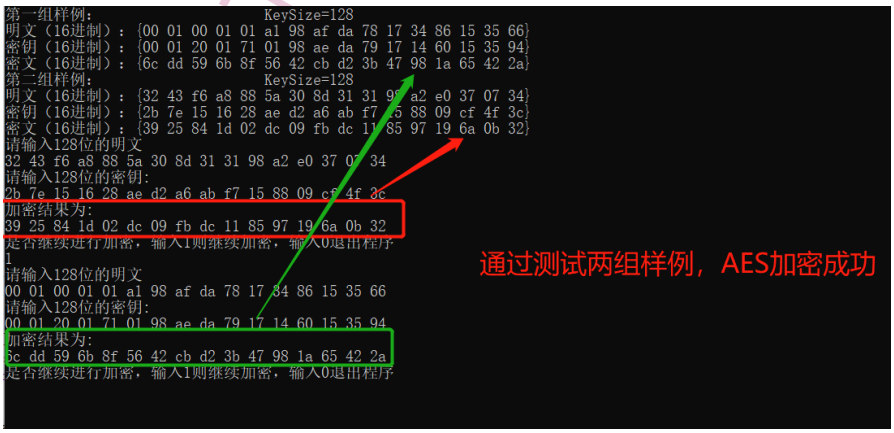

图 10: AES 加密演示

经过检验，AES 加密成功！

2. DES 解密

AES 解密和 AES 加密过程完全相同，将 AES 加密中的加密函数改为解密函数，并对输入输出做修改即可，现对老师给的样例进行测试，本次展示的测试样例为：

样例一：

密钥：00 01 00 01 01 a1 98 af da 78 17 34 86 15 35 66

明文：00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94

密文：6c dd 59 6b 8f 56 42 cb d2 3b 47 98 1a 65 42 2a

样例二：

密钥：32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

明文：2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

密文：39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32

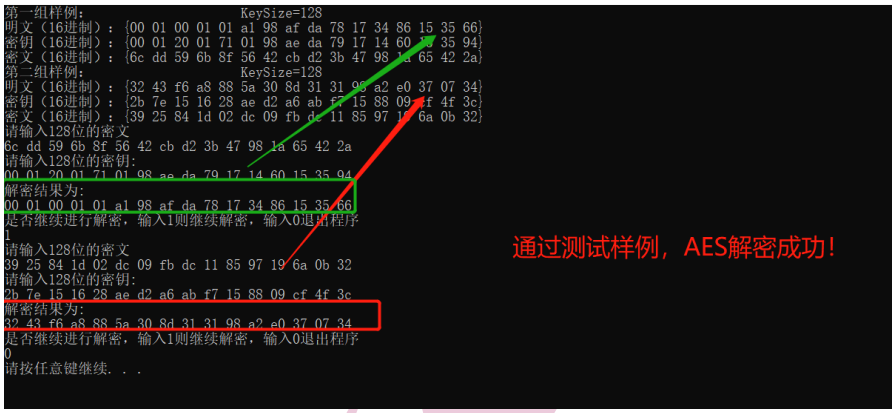

图 11: AES 解密演示

经过检验，AES 解密成功！

3. 改变密钥雪崩效应

AES 算法的密钥均为有效密钥，现对 128 位密钥每次改变一个 bit 位，观察雪崩现象，最后记录平均的改变位数，实现截图如下，观测到雪崩现象：

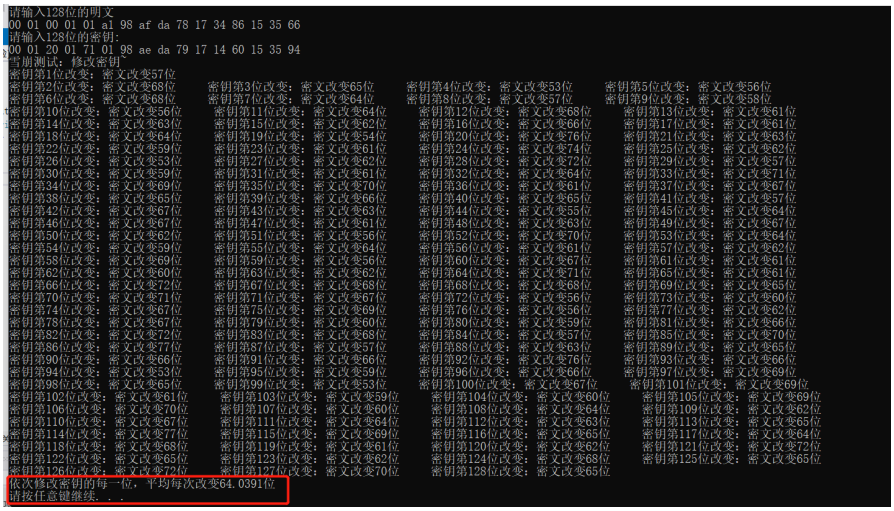

图 12: 修改密钥观察雪崩现象

经过检验，观测到雪崩现象，平均修改 64.0391 个 bit！

4. 改变明文雪崩效应

AES 算法的明文的每一位均为有效明文，现对 128 位明文每次改变一个 bit 位，观察雪崩现象，最后记录平均的改变位数，实现截图如下，观测到雪崩现象：

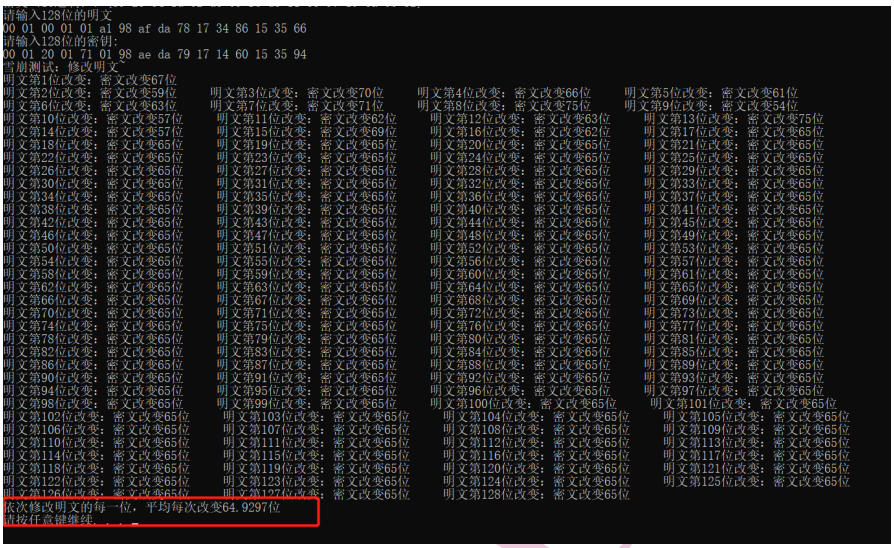

图 13: 修改明文观察雪崩现象

经过检验，观测到雪崩现象，平均修改 64.9297 个 bit！