



南開大學  
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

密码学实验报告

---

## 分组密码 AES

---

于文明

年级：2020 级

专业：信息安全

指导教师：古力

2022 年 12 月 8 日

# 摘要

关键字：Parallel

# 目录

|                                 |           |
|---------------------------------|-----------|
| <b>一、 实验内容</b>                  | <b>1</b>  |
| (一) 实验目的 . . . . .              | 1         |
| (二) 实验环境 . . . . .              | 1         |
| (三) 实验内容 . . . . .              | 1         |
| (四) 实验要求 . . . . .              | 1         |
| <b>二、 算法流程图</b>                 | <b>1</b>  |
| (一) AES 宏观流程图 . . . . .         | 2         |
| <b>三、 AES 算法介绍</b>              | <b>3</b>  |
| (一) AES 算法原理 . . . . .          | 3         |
| (二) 字节代换 ByteSub 部件 . . . . .   | 3         |
| (三) 行移位变换 ShiftRow . . . . .    | 4         |
| (四) 列混合运算 MixColumn . . . . .   | 4         |
| (五) 密钥加 AddRoundKey . . . . .   | 5         |
| (六) 密钥编排 . . . . .              | 5         |
| (七) 解密过程 . . . . .              | 6         |
| <b>四、 核心代码</b>                  | <b>6</b>  |
| (一) 工具函数 . . . . .              | 6         |
| 1. itob 和 btoi . . . . .        | 6         |
| 2. 伽罗瓦域上的运算 . . . . .           | 7         |
| (二) 字节代换 ByteSub . . . . .      | 9         |
| (三) 行移位 ShiftRow . . . . .      | 9         |
| (四) 列混合运算 MixColumn . . . . .   | 10        |
| (五) 密钥加 AddRoundKey . . . . .   | 10        |
| (六) 密钥扩展 KeyExpansion . . . . . | 11        |
| (七) 加密和解密 . . . . .             | 12        |
| (八) 雪崩效应检验 . . . . .            | 13        |
| <b>五、 实验结果</b>                  | <b>14</b> |
| (一) 加密过程检验 . . . . .            | 14        |
| (二) 解密过程检验 . . . . .            | 14        |
| (三) 雪崩效应检验 . . . . .            | 15        |
| 1. 验证样例 1 . . . . .             | 15        |
| 2. 验证样例 2 . . . . .             | 16        |
| <b>六、 附录</b>                    | <b>16</b> |

## 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

#### 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) 实验要求

(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、 算法流程图

AES 加密的主要过程有: 对明文状态的一次密钥加, 轮函数加密和末尾轮加密, 得到明文。其中轮函数加密每轮有四个部件, 包括字节代换 ByteSub, 行移位变换 ShiftRows, 列混合运算 MixColumn 和一个密钥加 AddRoundKey。末尾轮加密和前面的轮加密类似, 只是少了一个列混合变换 MixColumn。AES 加密解密的宏观流程图如下:

## (一) AES 宏观流程图

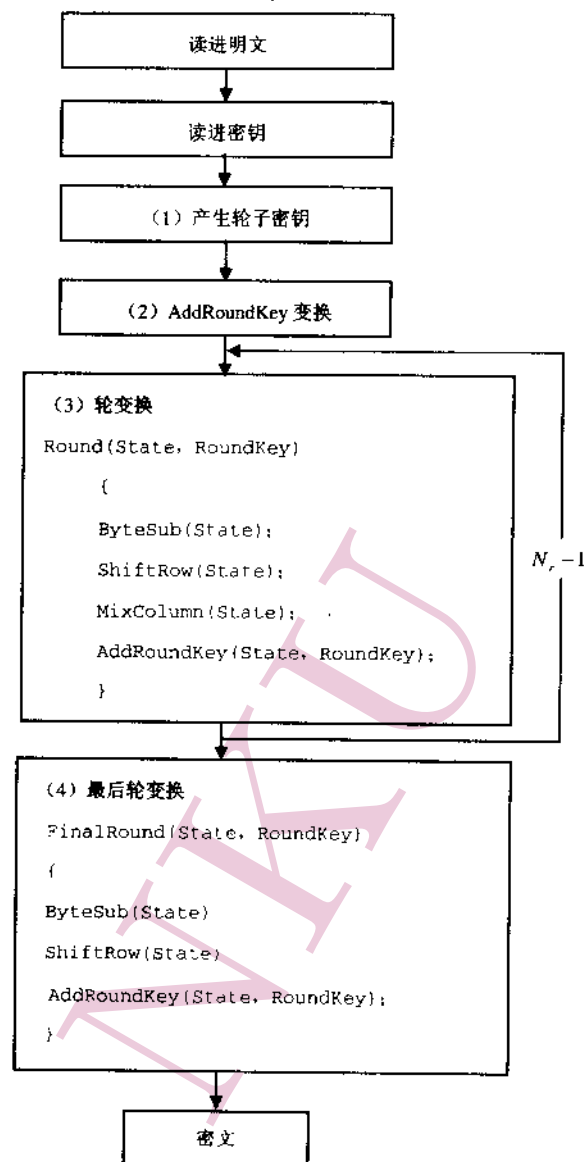


图 1: AES 宏观流程图

关于解密:

AES 的解密过程同加密过程类似,也是先经过一个密钥加,之后进行轮函数解密和末尾轮解密,得到明文。和加密不同的是轮函数解密的四个部件需要用到它们的逆运算部件。值得注意的是末尾轮解密和前面的轮加密类似,只是少了一个逆列混合变换部件。

我们在下面给出了完整的 AES 加密解密流程图

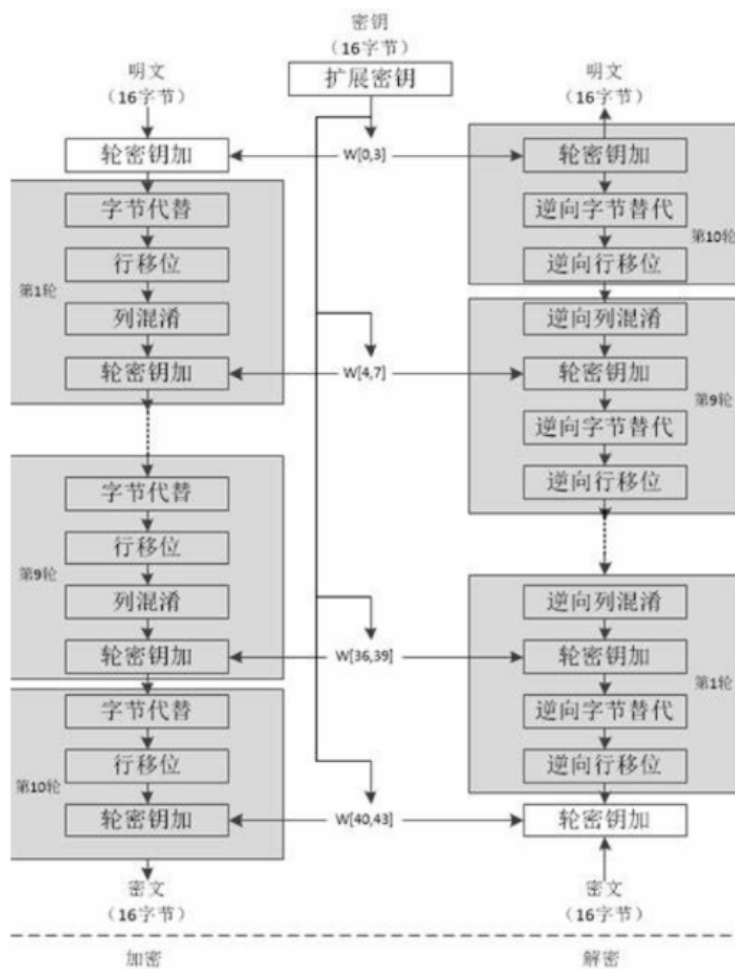


图 2: AES 加密解密流程图 (完整版)

### 三、 AES 算法介绍

#### (一) AES 算法原理

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。

#### (二) 字节替换 ByteSub 部件

字节替换是非线性变换，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: 字节代换 ByteSub

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。

### (三) 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, 具体关系如下表

表 3-10 对应于不同分组长度的位移量

| $N_b$ | $C_1$ | $C_2$ | $C_3$ |
|-------|-------|-------|-------|
| 4     | 1     | 2     | 3     |
| 6     | 1     | 2     | 3     |
| 8     | 1     | 3     | 4     |

图 4: 行位移表

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  $(j + N_b - C_i) \bmod N_b$ 。

### (四) 列混合运算 MixColumn

在列混合变换中, 将状态阵列的每个列视为系数为 GF(28) 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):  $c(x) = '03' x^3 + '01' x^2 + '01' x + '02'$   $c(x)$  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 5: 列混合运算矩阵

这个运算需要做 GF(28) 上的乘法, 但由于所乘的因子是 3 个固定的元素 02、03、01, 所以这些乘法运算仍然是比较简单的。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 ( '03' x3+ '01' x2+ '01' x+ '02' ) ) d(x)= '01' 由此可得 d(x)= '0B' x3+ '0D' x2+ '09' x+ '0E'

### (五) 密钥加 AddRoundKey

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 Nb。密钥加运算的逆运算是其自身。

### (六) 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;

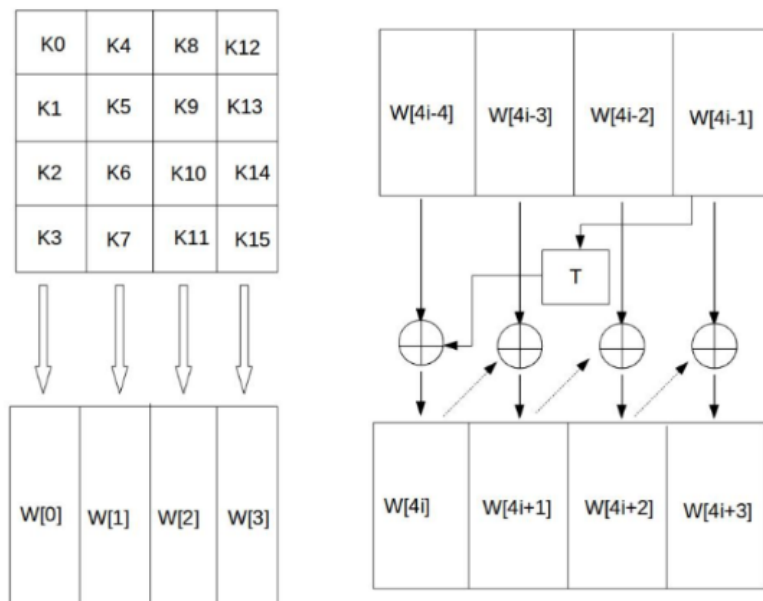


图 6: 密钥扩展流程图

(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。具体的内容请见教材和课程 ppt, 其中 128bit 密钥的扩展伪代码如下:

```

KeyExpansion(byte Key[4 *  $N_k$ ], W[ $N_b * (N_r + 1)$ ])
{
    For(i=0; i< $N_k$ ; i++)
        W[i] = (Key[4 * i], Key[4 * i+1], Key[4 * i+2], Key[4 * i+3]);
    For(i= $N_k$ ; i< $N_b * (N_r + 1)$ ; i++)
    {
        temp=W[i-1];
        if(i %  $N_k$  == 0)
            temp=SubByte(RotByte(temp))^Rcon[i/ $N_k$ ];
        W[i]=W[i- $N_k$ ]^temp;
    }
}

```

图 7: 密钥扩展伪代码

## (七) 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、 $\dots$ 、第  $N_r$  轮的子密钥依次为  $k(0), k(1), k(2), \dots, k(N_r-1), k(N_r)$

则解密算法的初始密钥加、第 1 轮、第 2 轮、 $\dots$ 、第  $N_r$  轮的子密钥依次为  $k(N_r), \text{InvMixColumn}(k(N_r-1)), \text{InvMixColumn}(k(N_r-2)), \dots, \text{InvMixColumn}(k(1)), k(0)$ 。

# 四、 核心代码

## (一) 工具函数

在代码中实现了 int 转二进制 string itob 和二进制 string 转 int btoi, 以及伽罗瓦域的乘法 GFMult, 具体实现如下:

### 1. itob 和 btoi

itob

```

1 string itob(int text[4][4]) {
2     string result;
3     for (int i = 0; i < 4; i++) {
4         for (int j = 0; j < 4; j++) {
5             string str = "00000000";

```



```

6         int temp = text[j][i];
7         for (int k = 7; k >= 0; k--) {
8             str[k] = '0' + temp % 2;
9             temp /= 2;
10        }
11        result += str;
12    }
13}
14return result;
15}

```

btoi 的实现同 itob 类似

btoi

```

1 void btoi(int text[4][4], string str) {
2     unsigned char* output = new unsigned char[16];
3     for (int i = 0; i <= 15; i++) {
4         int start = i * 8;
5         int temp = 0;
6         for (int j = start; j <= start + 7; j++) {
7             int each = 1;
8             for (int s = 1; s <= 7 - j + start; s++) {
9                 each *= 2;
10            }
11            if (str[i] == '1') {
12                temp += each;
13            }
14        }
15        output[i] = temp;
16    }
17    for (int i = 0; i < 4; i++) {
18        for (int j = 0; j < 4; j++) {
19            text[j][i] = output[j * 4 + i];
20        }
21    }
22}

```

## 2. 伽罗瓦域上的运算

对于伽罗瓦域上的加法运算来说，无所谓正负，等同于异或运算而对于伽罗瓦域上的乘法来说，AES 使用的是  $GF(2^8)$ ，所以将其看作多项式项的次数最高为 7，但是乘法中可能会超越这个数。解决方法是只需要对乘法结果对一个固定数取模即可，这就是  $m(x)$ ,  $m(x)=x^8 + x^4 + x + 1$ ,

MulGF

```

1 // 基本运算
2 int MulGF(int a, int b)
3 {

```

```
4      int third = b & 0x8;
5      int second = b & 0x4;
6      int first = b & 0x2;
7      int firstMod = b % 2;
8      int res = 0;
9
10     if (third)
11     {
12         int temp = a;
13         for (int i = 1; i <= 3; ++i)
14         {
15             temp = temp << 1;
16             if (temp >= 256)
17             {
18                 temp = temp ^ 0x11b;
19             }
20         }
21         temp = temp % 256;
22         res = res ^ temp;
23     }
24     if (second)
25     {
26         int temp = a;
27         for (int i = 1; i <= 2; ++i)
28         {
29             temp = temp << 1;
30             if (temp >= 256)
31             {
32                 temp = temp ^ 0x11b;
33             }
34         }
35         temp = temp % 256;
36         res = res ^ temp;
37     }
38     if (first)
39     {
40         int temp = a;
41         temp = temp << 1;
42         if (temp >= 256)
43         {
44             temp = temp ^ 0x11b;
45         }
46         temp = temp % 256;
47         res = res ^ temp;
48     }
49     if (firstMod)
50     {
51         res = res ^ a;
```

```
52     }
53     return res;
54 }
```

## (二) 字节代换 ByteSub

ByteSub

```
1 void ByteSub(int in[4][4], int type)
2 {
3     for (int i = 0; i < 4; i++)
4     {
5         for (int j = 0; j < 4; j++)
6         {
7             int temp = in[i][j];
8             int row = temp / 16;
9             int col = temp % 16;
10            if (type == 1)
11            {
12                in[i][j] = S[row][col];
13            }
14            if (type == 0)
15            {
16                in[i][j] = rS[row][col];
17            }
18        }
19    }
20 }
```

## (三) 行移位 ShiftRow

ShiftRow

```
1 void ShiftRow(int in[4][4], int type) {
2     for (int i = 0; i < 4; i++)
3     {
4         for (int j = 0; j < i; j++)
5         {
6             if (type == 1)
7             {
8                 int temp = in[i][0];
9                 in[i][0] = in[i][1];
10                in[i][1] = in[i][2];
11                in[i][2] = in[i][3];
12                in[i][3] = temp;
13            }
14            else
15            {
```

```

16         int temp = in[i][3];
17         in[i][3] = in[i][2];
18         in[i][2] = in[i][1];
19         in[i][1] = in[i][0];
20         in[i][0] = temp;
21     }
22 }
23 }
24 }

```

#### (四) 列混合运算 MixColumn

##### MixColumn

```

1 void MixColumn(int in[4][4], int type)
2 {
3     for (int i = 0; i < 4; i++)
4     {
5         int t0 = in[0][i];
6         int t1 = in[1][i];
7         int t2 = in[2][i];
8
9         int t3 = in[3][i];
10        if (type == 1)
11        {
12            in[0][i] = MulGF(t0, 2) ^ MulGF(t1, 3) ^ t2 ^ t3;
13            in[1][i] = t0 ^ MulGF(t1, 2) ^ MulGF(t2, 3) ^ t3;
14            in[2][i] = t0 ^ t1 ^ MulGF(t2, 2) ^ MulGF(t3, 3);
15            in[3][i] = MulGF(t0, 3) ^ t1 ^ t2 ^ MulGF(t3, 2);
16        }
17        else
18        {
19            in[0][i] = MulGF(t0, 14) ^ MulGF(t1, 11) ^ MulGF(t2,
20                13) ^ MulGF(t3, 9);
21            in[1][i] = MulGF(t0, 9) ^ MulGF(t1, 14) ^ MulGF(t2,
22                11) ^ MulGF(t3, 13);
23            in[2][i] = MulGF(t0, 13) ^ MulGF(t1, 9) ^ MulGF(t2,
24                14) ^ MulGF(t3, 11);
25            in[3][i] = MulGF(t0, 11) ^ MulGF(t1, 13) ^ MulGF(t2,
                9) ^ MulGF(t3, 14);
26        }
27    }
28 }

```

#### (五) 密钥加 AddRoundKey

##### AddRoundKey

```

1 void AddRoundKey(int in[4][4], int key[4][4])
2 {
3     for (int i = 0; i < 4; ++i)
4     {
5         for (int j = 0; j < 4; j++)
6         {
7             in[i][j] = in[i][j] ^ key[j][i];
8         }
9     }
10 }

```

## (六) 密钥扩展 KeyExpansion

### KeyExpansion

```

1 void KeyExpansion(int key[4][4], int w[11][4][4])
2 {
3     for (int i = 0; i < 4; ++i)
4     {
5         for (int j = 0; j < 4; j++)
6         {
7             w[0][i][j] = key[j][i];
8         }
9     }
10    for (int i = 1; i < 11; ++i)
11    {
12        for (int j = 0; j < 4; ++j)
13        {
14            int temp[4];
15            if (j == 0)
16            {
17                temp[0] = w[i - 1][3][1];
18                temp[1] = w[i - 1][3][2];
19                temp[2] = w[i - 1][3][3];
20                temp[3] = w[i - 1][3][0];
21                for (int k = 0; k < 4; ++k)
22                {
23                    int m = temp[k];
24                    int row = m / 16;
25                    int col = m % 16;
26                    temp[k] = S[row][col];
27                    if (k == 0)
28                    {
29                        temp[k] = temp[k] ^ rC[i -
30                        1];
31                    }

```

```

32         }
33     }
34     else
35     {
36         temp[0] = w[i][j - 1][0];
37         temp[1] = w[i][j - 1][1];
38         temp[2] = w[i][j - 1][2];
39         temp[3] = w[i][j - 1][3];
40     }
41     for (int x = 0; x < 4; x++)
42     {
43         w[i][j][x] = w[i - 1][j][x] ^ temp[x];
44     }
45 }
46 }
47 }

```

### (七) 加密和解密

根据实验原理部分的叙述，将上述部件组装起来实现 AES 的加密和解密

#### Encrypt

```

1 void Encrypt(int in[4][4], int key[4][4])
2 {
3     int type = 1;
4     int subKey[11][4][4];
5     KeyExpansion(key, subKey);
6     AddRoundKey(in, subKey[0]);
7     for (int i = 1; i <= 10; ++i)
8     {
9         ByteSub(in, type);
10        ShiftRow(in, type);
11        if (i != 10)
12        {
13            MixColumn(in, type);
14        }
15        AddRoundKey(in, subKey[i]);
16    }
17 }
18 void Decrypt(int in[4][4], int key[4][4])
19 {
20     int type = 0;
21     int subKey[11][4][4];
22     KeyExpansion(key, subKey);
23     AddRoundKey(in, subKey[10]);
24     for (int i = 9; i >= 0; --i)
25     {
26        ShiftRow(in, type);

```

```

27         ByteSub(in, type);
28         AddRoundKey(in, subKey[i]);
29         if (i != 0)
30         {
31             MixColumn(in, type);
32         }
33     }
34 }
35 }

```

## (八) 雪崩效应检验

同 DES 的雪崩检验类似，分别改变明文和密钥的位数各 8 次，检测密文改变的位数，具体实现如下：

### Encrypt

```

1  cout << "雪崩测试：修改明文~" << endl;
2  string result = itob(text\_new);
3  for (int i = 0; i <= 7; i++) {
4      cout << "明文改变" << (i + 1) << "位时： ";
5      if (result[i] == '0')
6          result[i] = '1';
7      else
8          result[i] = '0';
9      btoi(text\_new, result);
10     Encrypt(text\_new, key);
11     int result\_text = 0;
12     string s\_new = itob(text\_new);
13     string s = itob(text);
14     for (int i = 0; i < 128; i++) {
15         if (s\_new[i] == s[i]) {
16             result\_text++;
17         }
18     }
19     cout << "改变了";
20     cout << (dec) << result\_text << "位" << endl;
21 }
22 cout << "雪崩测试：修改密钥~" << endl;
23 result = itob(key\_new);
24 for (int i = 0; i <= 7; i++) {
25     cout << "密钥改变" << (i + 1) << "位时： ";
26     if (result[i] == '0')
27         result[i] = '1';
28     else
29         result[i] = '0';
30     btoi(key\_new, result);
31     Encrypt(text\_old, key\_new);
32     int result\_text = 0;

```

```

33     string s_new = itob(text_old);
34     string s = itob(text);
35     for (int i = 0; i < 128; i++) {
36         if (s_new[i] == s[i]) {
37             result_text++;
38         }
39     }
40     cout << "改变了";
41     cout << (dec) << result_text << "位" << endl;
42 }

```

## 五、 实验结果

测试样例如下

```

明文 (16进制) : 0001, 0001, 01a1, 98af, da78, 1734, 8615, 3566
密钥 (16进制) : 0001, 2001, 7101, 98ae, da79, 1714, 6015, 3594
密文 (16进制) : 6cdd, 596b, 8f56, 42cb, d23b, 4798, 1a65, 422a
明文 (16进制) : 3243, f6a8, 885a, 308d, 3131, 98a2, e037, 0734
密钥 (16进制) : 2b7e, 1516, 28ae, d2a6, abf7, 1588, 09cf, 4f3c
密文 (16进制) : 3925, 841d, 02dc, 09fb, dc11, 8597, 196a, 0b32

```

图 8: 测试样例

### (一) 加密过程检验

```

输入明文(hex 128bit):32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
输入密钥(hex 128bit):2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
密文(hex 128bit):0x39 0x25 0x84 0x1d 0x02 0xdc 0x09 0xfb 0xdc 0x11 0x85 0x97 0x19 0x6a 0x0b 0x32

```

图 9: 加密测试 1

```

输入明文(hex 128bit):00 01 00 01 01 a1 98 af da 78 17 34 86 15 35 66
输入密钥(hex 128bit):00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94
密文(hex 128bit):0x6c 0xdd 0x59 0x6b 0x8f 0x56 0x42 0xcb 0xd2 0x3b 0x47 0x98 0x1a 0x65 0x42 0x2a

```

图 10: 加密测试 2

### (二) 解密过程检验

```

输入密文(hex 128bit):6c dd 59 6b 8f 56 42 cb d2 3b 47 98 1a 65 42 2a
输入密钥(hex 128bit):00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94
明文:0x00 0x01 0x00 0x01 0x01 0xa1 0x98 0xaf 0xda 0x78 0x17 0x34 0x86 0x15 0x35 0x66

```

图 11: 解密验证 2



```
输入密文(hex 128bit):39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32
输入密钥(hex 128bit):2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
明文:0x32 0x43 0xf6 0xa8 0x88 0x5a 0x30 0x8d 0x31 0x31 0x98 0xa2 0xe0 0x37 0x07 0x34
```

图 12: 解密验证 2

### (三) 雪崩效应检验

#### 1. 验证样例 1

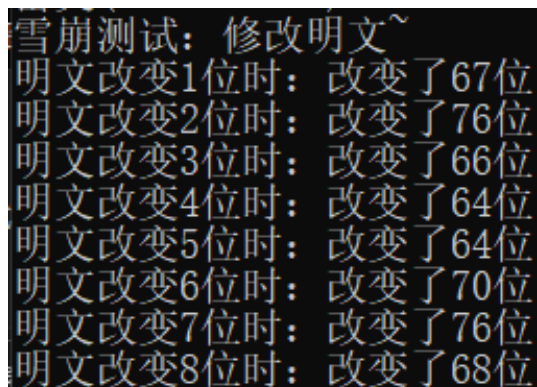
```
雪崩测试: 修改明文~
明文改变1位时: 改变了66位
明文改变2位时: 改变了67位
明文改变3位时: 改变了55位
明文改变4位时: 改变了62位
明文改变5位时: 改变了64位
明文改变6位时: 改变了62位
明文改变7位时: 改变了63位
明文改变8位时: 改变了61位
雪崩测试: 修改密钥~
```

图 13: 改变明文

```
雪崩测试: 修改密钥~
密钥改变1位时: 改变了73位
密钥改变2位时: 改变了64位
密钥改变3位时: 改变了58位
密钥改变4位时: 改变了62位
密钥改变5位时: 改变了65位
密钥改变6位时: 改变了53位
密钥改变7位时: 改变了75位
密钥改变8位时: 改变了63位
```

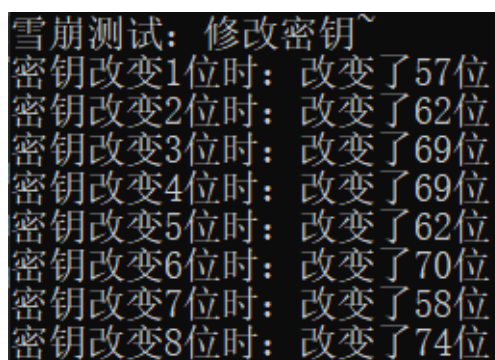
图 14: 改变密钥

## 2. 验证样例 2



```
雪崩测试：修改明文~  
明文改变1位时：改变了67位  
明文改变2位时：改变了76位  
明文改变3位时：改变了66位  
明文改变4位时：改变了64位  
明文改变5位时：改变了64位  
明文改变6位时：改变了70位  
明文改变7位时：改变了76位  
明文改变8位时：改变了68位
```

图 15: 改变明文



```
雪崩测试：修改密钥~  
密钥改变1位时：改变了57位  
密钥改变2位时：改变了62位  
密钥改变3位时：改变了69位  
密钥改变4位时：改变了69位  
密钥改变5位时：改变了62位  
密钥改变6位时：改变了70位  
密钥改变7位时：改变了58位  
密钥改变8位时：改变了74位
```

图 16: 改变密钥

参考文献 [1]

## 六、 附录

本次实验相关资源已经上传至 [github:https://github.com/Metetor/NKU\\_Crypter](https://github.com/Metetor/NKU_Crypter)

## 参考文献

- [1] 吴世忠、宋晓龙、郭涛等译 Paul Garrett 著. *An Introduction to Cryptology*. 机械工业出版社, 2003.

NIKU