

Lab3——分组密码算法 AES

学号：2111033

姓名：艾明旭

年级：21

专业：信息安全

一、实验内容说明

1、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。

2、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。

3、实验步骤

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。

二、实验环境

- 操作系统：win11
- 软件系统：visual studio
- 编译工具：vs2022
- 编程语言：C++

三、实验过程

本次实验首先翻阅课本，对理论课上的知识进行回顾，然后设计整个实验的流程图以及各个结构体和函数的大致思路，然后进行具体代码的编写实现，以下为具体过程：

1、流程分析

密钥扩展

AES 算法通过密钥扩展程序（Key Expansion）将用户输入的密钥 K 扩展生成 $Nb(Nr+1)$ 个字，存放在一个线性数组 $w[Nb*(Nr+1)]$ 中。具体如下：

1. **位置变换函数** `loop_wordbyte()`，接受一个字 `[a0, a1, a2, a3]` 作为输入，循环左移一个字节后输出 `[a1, a2, a3, a0]`。
2. **S 盒变换函数** `wordbyte_sub()`，接受一个字 `[a0, a1, a2, a3]` 作为输入。S 盒是一个 16x16 的表，其中每一个元素是一个字节。对于输入的每一个字节，前四位组成十六进制数 `x` 作为行号，后四位组成的十六进制数 `y` 作为列号，查找表中对应的值。最后函数输出 4 个新字节组成的 32-bit 字。（S 盒和逆 S 盒在程序中已经提前声明了一个结构体）
3. **轮常数 Rcon[]**，如何计算的就不说了（字节代换，行移位，列混合，密钥加），直接把它当做常量数组。
4. **扩展密钥数组 w[]** 的前 `Nk` 个元素就是外部密钥 `K`，以后的元素 `w[i]` 等于它前一个元素 `w[i-1]` 与前第 `Nk` 个元素 `w[i-Nk]` 的异或，即 `w[i] = w[i-1] XOR w[i-Nk]`；但若 `i` 为 `Nk` 的倍数，则 `w[i] = w[i-Nk] XOR wordbyte_sub(loop_wordbyte(w[i-1])) XOR Rcon[i/Nk-1]`。

在本次的实验中主要采用的是字符型的变量来进行参数的传递和转移，但是在计算的过程中会将字符型的数组转变为数字，但由于 128 位的数字太大，所以将其分为 4 组，每一个都由四个 8 进制数来构成，如下所示：

```
vector<string> group_key(string& key)
{
    // 四组
    vector<string> groups(4);
    // 初始下标
    int index = 0;
    // 分组
    for (string& g : groups)
    {
        g = key.substr(index, 8);
        index += 8;
    }
    return groups;
}
```

在密钥扩展中，当下标为 4 的倍数的时候，进行的过程较为复杂，此部分的流程图如下所示：

字循环
loop_wordbyte



字节代换
wordbyte_sub



轮异或
xor_with_const

AES 加密

AES 加密总共大体上可分为 4 个部分来构造，分别为 S 盒变换、行变换、列变换以及与扩展密钥的异或，各个部分的大致内容如下：

S 盒变换-wordbyte_sub()

在密钥扩展部分已经讲过了，S 盒是一个 16 行 16 列的表，表中每个元素都是一个字节。S 盒变换很简单：函数 wordbyte_sub()接受一个 4x4 的字节矩阵作为输入，对其中的每个字节，前四位组成十六进制数 x 作为行号，后四位组成的十六进制数 y 作为列号，查找表中对应的值替换原来位置上的字节。

行变换-move_row()

行变换也很简单，它仅仅是将矩阵的每一行以字节为单位循环移位：第一行不变，第二行左移一位，第三行左移两位，第四行左移三位。

列变换-col_confuse()

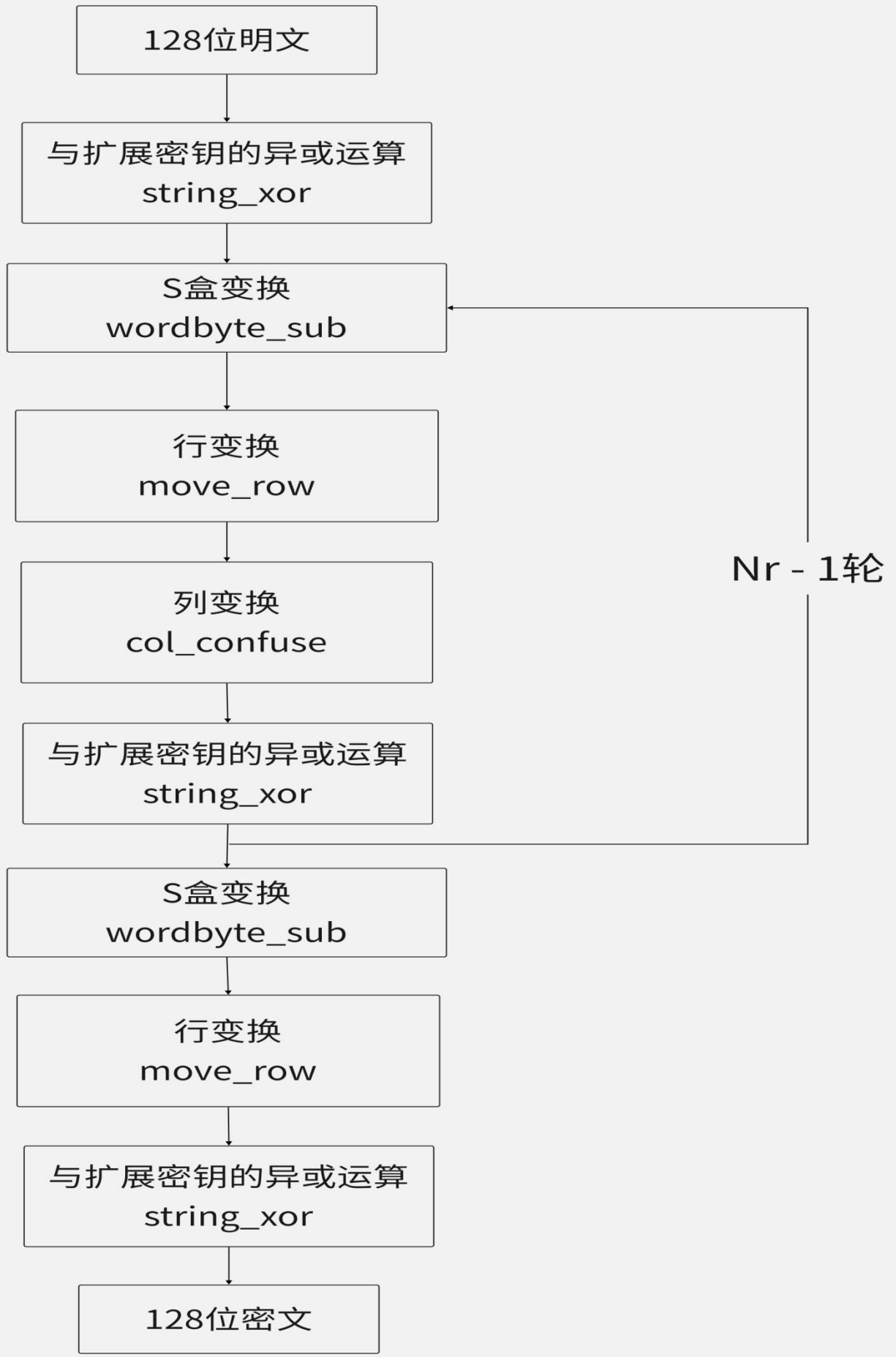
函数 col_confuse()同样接受一个 4x4 的字节矩阵作为输入，并对矩阵进行逐列变换，注意公式中用到的乘法是伽罗华域（GF，有限域）上的乘法。

与扩展密钥的异或-string_xor()

扩展密钥只参与了这一步。根据当前加密的轮数，用 w[]中的 4 个扩展密钥与矩阵的 4 个列进行按位异或。到这里 AES 加密的各个部分差不多了。

流程图

最后 AES 加密的流程图如下图所示：



AES 解密

AES 解密与 AES 加密类似，基本上都是其加密的逆过程，总共大体上也可分为 4 个部分来构造，分别为逆行变换、逆 S 盒变换、逆列变换以及与扩展密钥的异或，各个部分的大致内容如下（由于与扩展密钥的异或和 AES 加密的部分一样，在这里就不再过多赘述）：

逆行变换-in_move_row()

上面讲到 move_row() 是对矩阵的每一行进行循环左移，所以 in_move_row() 是对矩阵每一行进行循环右移。

逆 S 盒变换-in_wordbyte_sub()

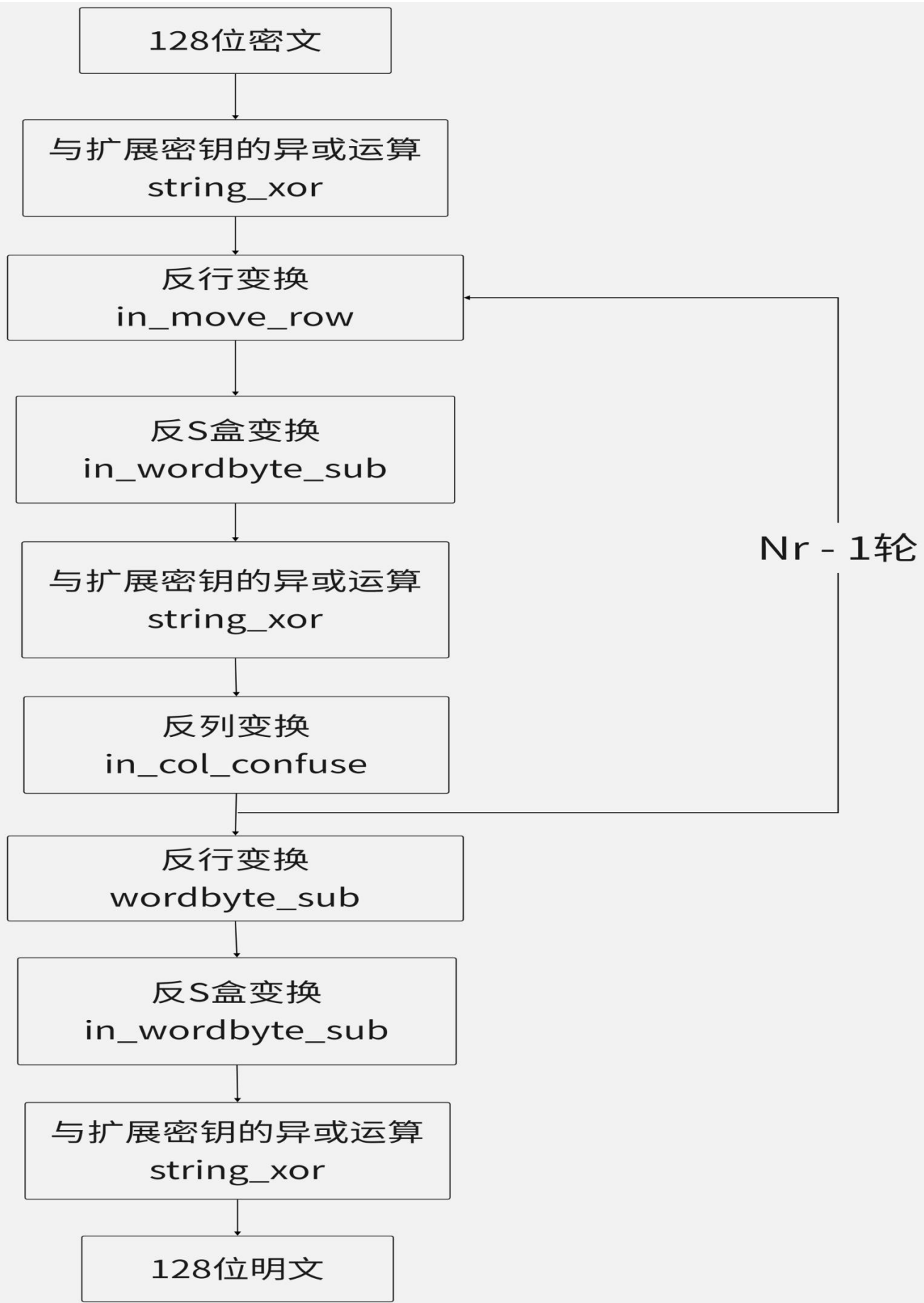
与 S 盒变换一样，也是查表，查表的方式也一样，只不过查的是另外一个置换表（S-Box 的逆表）。

逆列变换-in_col_confuse()

与列变换的方式一样，只不过计算公式的系数矩阵发生了变化，只要写出三个逆变换的函数，然后根据伪代码就很容易实现 AES 解密算法了。

流程图

最后 AES 解密的流程图如下图所示：



2、代码实现

各类的结构体（即书本上出现的置换表以及各种扩展和置换运算）已经预先定义

各类预处理函数

- 首先是字符串，数字，以及 bit 流之间的各种转换，bit 流的转换主要用于雪崩效应时可以检测出共有多少为不相同

// 将 string 类型转换为 bits 类型，方便后续的雪崩效应

```
void getbit(string a, bitset<128>& temp)
{
    int num = 127;
    for (int i = 2; i < a.length(); i++)
    {
        if (a[i] <= '9')
        {
            for (int j = 0; j < 4; j++)
            {
                temp[num--] = HexToBit[a[i] - 48][j];
            }
        }
        else
        {
            for (int j = 0; j < 4; j++)
            {
                temp[num--] = HexToBit[a[i] - 65 + 10][j];
            }
        }
    }
}
```

// 字符 --> 数字

```
int ch_to_int(char& ch)
{
    int ans = 0;
    // 数字的时候
    if (ch >= 48 && ch <= 57)
    {
        ans = ch - '0';
    }
    // 16 进制中 a 到 f
    else if (ch >= 'a' && ch <= 'f')
    {
        ans = ch - 'a' + 10;
    }
    // 16 进制中 A 到 F
    else if (ch >= 'A' && ch <= 'F')
    {
        ans = ch - 'A' + 10;
    }
    return ans;
}
```



```

// 16 进制字符串 --> 数字
long long str_long(string str)
{
    long long ans = 0;
    // 遍历字符串，将字符串的内容变为16 进制数字
    for (char ch : str)
    {
        ans = ans * 16 + ch_to_int(ch);
    }
    return ans;
}

```

```

// 数字 --> 16 进制字符串 (只考虑小写)
string int_to_chs(long long num)
{
    string ans = "";
    while (num)
    {
        // 通过位运算得到低四位
        int x = num & 0xf;
        // 根据数值进行区分
        if (x <= 9)
        {
            char ch = x + '0';
            ans += ch;
        }
        else
        {
            char ch = x - 10 + 'a';
            ans += ch;
        }
        // 移位，其实相当于 / 16
        num >>= 4;
    }
    // 然后反转字符串
    int left = 0, right = ans.length() - 1;
    // 双指针实现字符串反转
    while (left < right)
    {
        char ch = ans[left];
        ans[left] = ans[right];
        ans[right] = ch;
        left++;
        right--;
    }
    return ans;
}

```

- 然后是对字节循环的实现

```

string loop_wordbyte(string& wi_1)
{
    string ans = wi_1.substr(2) + wi_1.substr(0, 2);
    return ans;
}

```

- 接下来是对字节代换的实现

```

string wordbyte_sub(string& wi_1)
{
    int len = wi_1.length();
    string ans = "";
    for (int i = 0; i < len; i += 2)
    {
        // 先获取当前的下标
        int x = ch_to_int(wi_1[i]), y = ch_to_int(wi_1[i + 1]);
        // 然后获取当前的数字
        int num = S[x][y];
        // 先将数值转化为字符串
        string s = int_to_chs(num);
        // 然后不足的话补0
        while (s.length() < 2)
        {
            s = "0" + s;
        }

        // 加起来
        ans += s;
    }
    return ans;
}

```

- 然后是对密钥的轮常量异或的实现

```

string xor_with_const(string& wi_1, int rounds)
{
    // 先将字符串变为数字
    long long num = 0;
    for (int i = 0; i < 8; ++i)
    {
        char ch = wi_1[i];
        num = num * 16 + ch_to_int(ch);
    }
    // 计算异或结果
    num ^= Rcon[rounds];

    // 将num 转化为字符串
    string res = int_to_chs(num);
    while (res.length() < 8)
    {
        res = "0" + res;
    }
    return res;
}

```

- 密钥拓展的时候，下标为 4 的倍数时，需要使用一个特殊的变换函数

```
string T(string& wi_1, int round)
{
    // T 变换由3 部分构成，用的即为上述描述的三个函数
    // 先进行字循环
    string ans = loop_wordbyte(wi_1);
    // 然后字节代换
    ans = wordbyte_sub(ans);
    // 最后是轮异或
    ans = xor_with_const(ans, round);

    return ans;
}
```

- 密钥编排的实现

```
vector<string> extend_key(string& key)
{
    // 先分组
    vector<string> w_key = group_key(key);
    for (int i = 0; i < 40; ++i)
    {
        string w = "";
        int index = 4 + i;
        string temp = w_key[index - 1];
        // 4 的倍数的时候，需要调用T 函数
        if (index % 4 == 0)
        {
            temp = T(temp, index / 4 - 1);
        }
        w = string_xor(temp, w_key[index - 4]);

        // 压入数组中
        w_key.push_back(w);
    }

    return w_key;
}
```

- 行移位函数的实现

```
vector<string> move_row(vector<string>& s)
{
    vector<string> ans = s;
    // 字符串数组每个对应一列，所以是对应到列进行移位
    // 一行对应有两个16 进制数，所以需要两个一起移动，相当于两列一起移动
    for (int i = 0; i < 4; ++i)
    {
        int k = i * 2;
        // 就原本矩阵对应的行移位，对于字符串数组就是列移位
        for (int j = 0; j < 4; ++j)
        {
            ans[j][k] = s[(j + i) % 4][k];
            ans[j][k + 1] = s[(j + i) % 4][k + 1];
        }
    }
}
```

```

    }
}
return ans;
}

```

- 列混淆函数的实现

```

vector<string> col_confuse(vector<string>& s)
{
    vector<string> ans = s;
    // 算法中对应的是列，这边就直接变成了行，即字符
    for (int i = 0; i < 4; ++i)
    {
        // 需要先将字符串拆分成两两一组，共4组
        auto temp = split_s(s[i]);
        // 先转成数字
        int s0 = str_long(temp[0]), s1 = str_long(temp[1]), s2 = str_long(temp[2]),
            s3 = str_long(temp[3]);
        // 计算混淆后的值
        int t0 = power(s0) ^ power(s1) ^ s1 ^ s2 ^ s3;
        int t1 = s0 ^ power(s1) ^ power(s2) ^ s2 ^ s3;
        int t2 = s0 ^ s1 ^ power(s2) ^ s3 ^ power(s3);
        int t3 = s0 ^ power(s0) ^ s1 ^ s2 ^ power(s3);
        // 转换成字符串再相加
        ans[i] = int_ch2(t0) + int_ch2(t1) + int_ch2(t2) + int_ch2(t3);
    }
    return ans;
}

```

- 行移位的逆操作函数的实现

```

vector<string> in_move_row(vector<string>& s)
{
    vector<string> ans = s;
    // 现在变成了逆操作
    for (int i = 0; i < 4; ++i)
    {
        int k = i * 2;
        // 就原本矩阵对应的行移位，对于字符串数组就是列移位
        for (int j = 0; j < 4; ++j)
        {
            ans[j][k] = s[(j - i + 4) % 4][k];
            ans[j][k + 1] = s[(j - i + 4) % 4][k + 1];
        }
    }
    return ans;
}

```

- 逆字节代换的实现

```

string in_wordbyte_sub(string& wi_1)
{
    int len = wi_1.length();
    string ans = "";
    for (int i = 0; i < len; i += 2)
    {

```

```

// 先获取当前的下标
int x = ch_to_int(wi_1[i]), y = ch_to_int(wi_1[i + 1]);
// 然后获取当前的数字
int num = S1[x][y];
// 先将数值转化为字符串
string s = int_to_chs(num);
// 然后不足的话补0
while (s.length() < 2)
{
    s = "0" + s;
}

// 加起来
ans += s;
}
return ans;
}

```

- 列混淆函数的逆变换实现

```

vector<string> in_col_confuse(vector<string>& s)
{
    // 逆变换其实原来的变换矩阵的逆矩阵, 对应0xe, 0xb, 0xd, 0x9
    // 4 列

    vector<string> ans = s;
    for (int i = 0; i < 4; ++i)
    {
        // 先分割成4 个两位数字
        auto temp = split_s(s[i]);
        // 转换成数字
        vector<int> nums(4);
        for (int j = 0; j < 4; ++j)
        {
            nums[j] = str_long(temp[j]);
        }
        vector<int> t4(4, 0);
        for (int j = 0; j < 4; ++j)
        {
            for (int t = 0; t < 4; ++t)
            {
                int k = (t - j + 4) % 4;
                t4[j] ^= power(power(power(nums[t]))); // 表示8
                switch (k)
                {
                    case 0: // 0xe = 8 + 4 + 2
                    {
                        t4[j] ^= power(power(nums[t])) ^ power(nums[t]);
                        break;
                    }
                    case 1: // 0xb = 8 + 2 + 1
                    {
                        t4[j] ^= power(nums[t]) ^ nums[t];
                    }
                }
            }
        }
    }
}

```

```

        break;
    }
    case 2:          // 0xd = 8 + 4 + 1
    {
        t4[j] ^= power(power(nums[t])) ^ nums[t];
        break;
    }
    default:         // 0x9 = 8 + 1
        t4[j] ^= nums[t];
        break;
    }
}
}
// 将数字转换成字符串存储
ans[i] = int_ch2(t4[0]) + int_ch2(t4[1]) + int_ch2(t4[2]) + int_ch2(t4[3]);
}

return ans;
}

```

加密过程

- 按流程图进行实现

```

vector<string> aes(string& plain_text, string& key)
{
    // 先拓展密钥
    vector<string> keys = extend_key(key);

    int index = 0;
    // 然后就是 10 轮迭代
    // 需要知道明文其实是 32 位，所以需要搞 4 下
    // 可以先把明文也分组

    // 一开始的先进进行一次轮密钥加
    vector<string> texts = group_key(plain_text);
    for (int i = 0; i < 4; ++i)
    {
        texts[i] = string_xor(texts[i], keys[i]);
    }
    index += 4;

    // 然后十次迭代
    for (int k = 0; k < 10; ++k)
    {
        for (int j = 0; j < 4; ++j)
        {
            // 先是字节代换
            texts[j] = wordbyte_sub(texts[j]);
        }
        // 然后是行移位
        texts = move_row(texts);
    }
}

```

```

    if (k < 9)
    {
        // 再来列混淆
        texts = col_confuse(texts);
    }

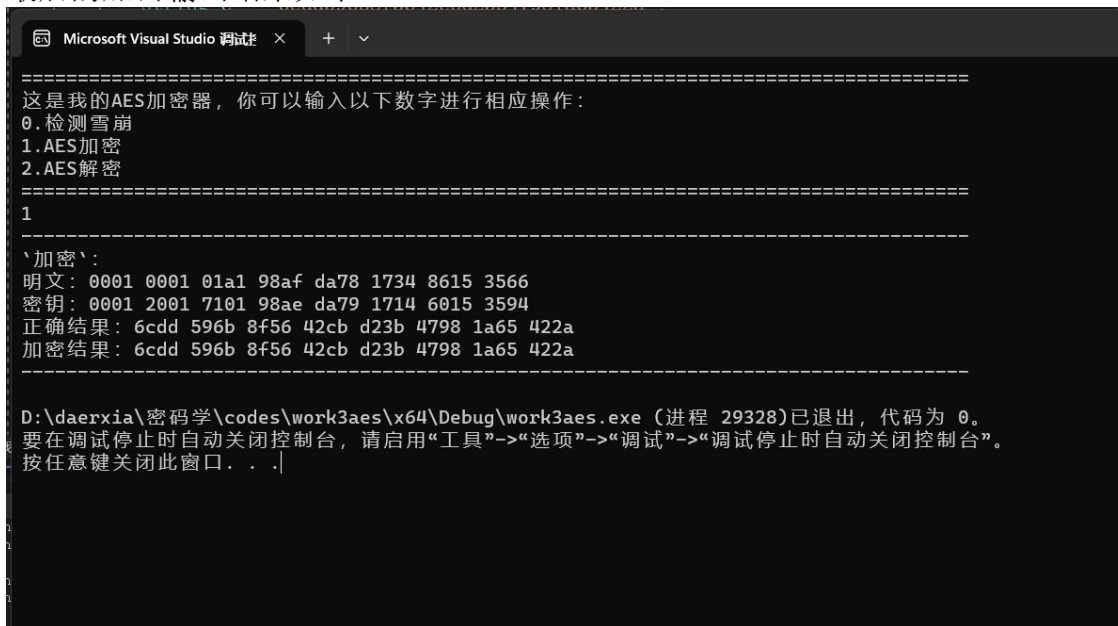
    // 轮密钥加
    for (int i = 0; i < 4; ++i)
    {
        texts[i] = string_xor(texts[i], keys[i + index]);
    }

    index += 4;
}

// 最后进行输出
// show(texts);
return texts;
}

```

- 最后的加密输出结果如下：



```

Microsoft Visual Studio 调试
=====
这是我的AES加密器，你可以输入以下数字进行相应操作：
0.检测雪崩
1.AES加密
2.AES解密
=====
1
=====
`加密`:
明文: 0001 0001 01a1 98af da78 1734 8615 3566
密钥: 0001 2001 7101 98ae da79 1714 6015 3594
正确结果: 6cdd 596b 8f56 42cb d23b 4798 1a65 422a
加密结果: 6cdd 596b 8f56 42cb d23b 4798 1a65 422a
=====

D:\daerxia\密码学\codes\work3aes\x64\Debug\work3aes.exe (进程 29328)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . |

```

```
Microsoft Visual Studio 调试 × + ▾
=====
这是我的AES加密器，你可以输入以下数字进行相应操作：
0. 检测雪崩
1. AES加密
2. AES解密
=====
1
=====
`加密`：
明文：3243 f6a8 885a 308d 3131 98a2 e037 0734
密钥：2b7e 1516 28ae d2a6 abf7 1588 09cf 4f3c
正确结果：3925 841d 02dc 09fb dc11 8597 196a 0b32
加密结果：3925 841d 02dc 09fb dc11 8597 196a 0b32
=====

D:\daerxia\密码学\codes\work3aes\x64\Debug\work3aes.exe (进程 4012)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . |
```

可以看到加密结果与正确结果一样

解密过程

- 按流程图进行实现

```
vector<string> in_aes(string& text, string& key)
```

```
{
    // 先拓展密钥
    auto keys = extend_key(key);

    // 初始下标
    int index = 40;

    // 对密文分组
    vector<string> texts = group_key(text);

    // 一开始先进行依次轮密钥加
    for (int i = 0; i < 4; ++i)
    {
        texts[i] = string_xor(texts[i], keys[index + i]);
    }
    index -= 4;

    // 然后十次迭代
    for (int i = 0; i < 10; ++i)
    {
        // 先逆行移位
        texts = in_move_row(texts);

        // 然后是字节代换逆操作
        for (int j = 0; j < 4; ++j)
```



```

{
    texts[j] = in_wordbyte_sub(texts[j]);
}

// 轮密钥加
for (int j = 0; j < 4; ++j)
{
    texts[j] = string_xor(texts[j], keys[index + j]);
}

// 除了最后一轮，都要列混淆逆变换
if (i < 9)
{
    texts = in_col_confuse(texts);
}
index -= 4;
}

// show(texts);

return texts;
}

```

- 最后的解密输出结果如下：

```

Microsoft Visual Studio 调试  x + v
=====
这是我的AES加密器，你可以输入以下数字进行相应操作：
0. 检测雪崩
1. AES加密
2. AES解密
=====
2
=====
`解密`:
密文: 6cdd 596b 8f56 42cb d23b 4798 1a65 422a
密钥: 0001 2001 7101 98ae da79 1714 6015 3594
正确结果: 0001 0001 01a1 98af da78 1734 8615 3566
解密结果: 0001 0001 01a1 98af da78 1734 8615 3566
=====

D:\daerxia\密码学\codes\work3aes\x64\Debug\work3aes.exe (进程 14292)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...

```

```
Microsoft Visual Studio 调试 x + v
=====
这是我的AES加密器，你可以输入以下数字进行相应操作：
0.检测雪崩
1.AES加密
2.AES解密
=====
2
=====
\解密\：
密文：3925 841d 02dc 09fb dc11 8597 196a 0b32
密钥：2b7e 1516 28ae d2a6 abf7 1588 09cf 4f3c
正确结果：3243 f6a8 885a 308d 3131 98a2 e037 0734
解密结果：3243 f6a8 885a 308d 3131 98a2 e037 0734
=====

D:\daerxia\密码学\codes\work3aes\x64\Debug\work3aes.exe (进程 21668)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . |
```

可以看到解密结果与正确结果一样

雪崩效应

- 在输入为 0 时进行雪崩效应
- 先固定密钥，每次改变一个明文，进行八次，观察更改的结果的不同位数
- 然后固定明文，每次改变一个密钥，进行八次，观察更改的结果的不同位数 最后的雪崩效应的输出结果如下：
- 改变明文：

```

0
+++++
接下来每次改变一位明文，观察结果
原始明文：0001000101a198afda78173486153566
改后明文：0021000101a198afda78173486153566
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：99e61e19bca2ab796a232d1bc362ec81
不同的数字位数：49
原始明文：0001000101a198afda78173486153566
改后明文：0041000101a198afda78173486153566
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：2c4307250d09c6255ef498a5a46c874a
不同的数字位数：65
原始明文：0001000101a198afda78173486153566
改后明文：0081000101a198afda78173486153566
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：d885aa71ece8e1f32ade19e66906b6bc
不同的数字位数：56
原始明文：0001000101a198afda78173486153566
改后明文：0011000101a198afda78173486153566
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：92c9f4fd295da2041a317efde517ea24
不同的数字位数：58
原始明文：0001000101a198afda78173486153566
改后明文：1001000101a198afda78173486153566
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：f28f6980b970db9063fe04b79e97b8d7
不同的数字位数：61
原始明文：0001000101a198afda78173486153566
改后明文：2001000101a198afda78173486153566
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：34690b1d5b8475cfe8639f2e3db0479b
不同的数字位数：57
原始明文：0001000101a198afda78173486153566
改后明文：4001000101a198afda78173486153566
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：10d398e757543a670e3a55e5e3b5f8a4
不同的数字位数：48
原始明文：0001000101a198afda78173486153566

```

```

Microsoft Visual Studio 调试
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：34690b1d5b8475cfe8639f2e3db0479b
不同的数字位数：57
原始明文：0001000101a198afda78173486153566
改后明文：4001000101a198afda78173486153566
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：10d398e757543a670e3a55e5e3b5f8a4
不同的数字位数：48
原始明文：0001000101a198afda78173486153566
改后明文：8001000101a198afda78173486153566
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：27bf3c8d7d5e54bb3df1f23a255e748f
不同的数字位数：66
改变明文获得不同的数字位数的平均数为：57.5

```

```

-----
接下来每次改变一位密钥，观察结果
原始密钥：00012001710198aeda79171460153594
改后密钥：10012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：e431b6f5460da74b20d68c054581b60e
不同的数字位数：68
原始密钥：00012001710198aeda79171460153594
改后密钥：20012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：7dd8488ee90ebfc5a11c93e2bc1f3498
不同的数字位数：61
原始密钥：00012001710198aeda79171460153594
改后密钥：40012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：c128e07e73523d4cd57b2bd7b7ef3abb
不同的数字位数：55
原始密钥：00012001710198aeda79171460153594
改后密钥：80012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：8d9ef35b7d823ea52ffa98fa3cdaeb43
不同的数字位数：59
原始密钥：00012001710198aeda79171460153594
改后密钥：01012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a

```

使密钥不变，每次仅更改一位明文，重复八次，平均每次有 57.5 位发生变化

这是我的AES加密器，你可以输入以下数字进行相应操作：

0. 检测雪崩
1. AES加密
2. AES解密

0
+++++
接下来每次改变一位明文，观察结果
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：1243f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：52a0e5a9dc679d10789f91d0f4960b2a
不同的数字位数：56
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：2243f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：785c931a63f24aa8ac546b2c4f9674c0
不同的数字位数：64
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：4243f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：2bfd6c46f58e99d43206ec8553a80c2b
不同的数字位数：60
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：8243f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：507d7bb70c900c45741c1edc33a8e9ed
不同的数字位数：66
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：3143f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：b7c3921e8d70ca69a8d8d02659b4e3a6
不同的数字位数：57
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：3443f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：89072b35a56395c7e083afa1fb1c4588
不同的数字位数：58

改后明文：4243f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：2bfd6c46f58e99d43206ec8553a80c2b
不同的数字位数：60
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：8243f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：507d7bb70c900c45741c1edc33a8e9ed
不同的数字位数：66
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：3143f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：b7c3921e8d70ca69a8d8d02659b4e3a6
不同的数字位数：57
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：3443f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：89072b35a56395c7e083afa1fb1c4588
不同的数字位数：58
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：3643f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：c0fd5e72c3c25782486636afcc6c27a3
不同的数字位数：61
原始明文：3243f6a8885a308d313198a2e0370734
改后明文：3843f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：eb740eae17a93703fb00221574b34402
不同的数字位数：50
改变明文获得不同的数字位数的平均数为：59

接下来每次改变一位密钥，观察结果

原始密钥：2b7e151628aed2a6abf7158809cf4f3c
改后密钥：3143f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：0d23befc04d2420d4d6ad644ba07a713
不同的数字位数：54
原始密钥：2b7e151628aed2a6abf7158809cf4f3c
改后密钥：3443f6a8885a308d313198a2e0370734

对于实验二的测试数据，使密钥不变，每次仅更改一位明文，重复八次，平均每次有 54 位发生变化
改变密钥：

使明文不变，每次仅更改一位密钥，重复八次，平均每次有 58.375 位发生变化

```
Microsoft Visual Studio 调试 × + ~
不同的数字位数：68
原始密钥：00012001710198aeda79171460153594
改后密钥：20012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：7dd8488ee90ebfc5a11c93e2bc1f3498
不同的数字位数：61
原始密钥：00012001710198aeda79171460153594
改后密钥：40012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：c128e07e73523d4cd57b2bd7b7ef3abb
不同的数字位数：55
原始密钥：00012001710198aeda79171460153594
改后密钥：80012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：8d9ef35b7d823ea52ffa98fa3cdaeb43
不同的数字位数：59
原始密钥：00012001710198aeda79171460153594
改后密钥：01012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：a77a41f8e720c5f97ab7c853273ba980
不同的数字位数：63
原始密钥：00012001710198aeda79171460153594
改后密钥：02012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：ceb940bb9c86278beb8628ee008d2f0c
不同的数字位数：53
原始密钥：00012001710198aeda79171460153594
改后密钥：04012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：eea969de8bc235081f620e059bc10658
不同的数字位数：62
原始密钥：00012001710198aeda79171460153594
改后密钥：08012001710198aeda79171460153594
原始结果：6cdd596b8f5642cbd23b47981a65422a
改后结果：6e88d04be20b94abdabc652e21e281f3
不同的数字位数：46
改变明文获得不同的数字位数的平均数为：58.375
+++++
```



```
不同的数字位数：54
原始密钥：2b7e151628aed2a6abf7158809cf4f3c
改后密钥：3443f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：7251c152b32ee3302a33ca18e9463459
不同的数字位数：62
原始密钥：2b7e151628aed2a6abf7158809cf4f3c
改后密钥：3643f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：5736d596b1ac9ab451329ac08ada9fbe
不同的数字位数：54
原始密钥：2b7e151628aed2a6abf7158809cf4f3c
改后密钥：3843f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：e35d2a33a78ee16c624d4f756c9de473
不同的数字位数：67
原始密钥：2b7e151628aed2a6abf7158809cf4f3c
改后密钥：3213f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：224fdbaa8b69214e76ed0b1fa27697fe
不同的数字位数：59
原始密钥：2b7e151628aed2a6abf7158809cf4f3c
改后密钥：3263f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：d3b455fd68ee9a036601cbbf6fd1b243
不同的数字位数：51
原始密钥：2b7e151628aed2a6abf7158809cf4f3c
改后密钥：3223f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：de9b9e738d7a8572f1ff4747eb7ff0a2
不同的数字位数：58
原始密钥：2b7e151628aed2a6abf7158809cf4f3c
改后密钥：3293f6a8885a308d313198a2e0370734
原始结果：3925841d02dc09fbdc118597196a0b32
改后结果：5796cd8565bad43ed0983c61cfb06995
不同的数字位数：71
改变明文获得不同的数字位数的平均数为：59.5
+++++
```

对第二组数据，使明文不变，每次仅更改一位密钥，重复八次，平均每次有 59.5 位发生变化

综上所述可以看出即使小小的改动，加密结果也会发生很大的变化

四、总结与展望

1、总结

本次是密码学的第三次实验，在这次实验中对理论课上讲解的 AES 加解密进行编程，使得对于这种密码算法的原理和攻击方法更加的了解，也对密码学方面的编程更加的熟悉。

在本次实验中，我们主要掌握了以下几个方面的知识：AES 加密算法基本原理和实现方式。我们了解了 AES 加密算法的基本结构，包括 AES 加密过程中的四个核心算法：字节替代、行移位、列混合和轮密钥加。同时，我们掌握了 AES 加密算法和解密算法的详细过程和实现方式。

2、展望

通过本次实验，我们对 AES 分组密码加密算法有了更深入的了解和掌握，也明确了其在信息安全领域中的重要作用和应用前景。我们相信，通过更深入的学习和实践，我们将能够更好地应用 AES 分组密码算法来提高信息安全性和保障数据的安全传输和存储。