实验三 分组密码算法AES

学号：　　姓名：　　专业：

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、实验原理

1. 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。
2. 加密的主要过程包括：对明文状态的一次密钥加，轮轮加密和末尾N_r-1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。

三、实验环境

运行Windows操作系统的PC机，具有VC等语言编译环境

四、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。

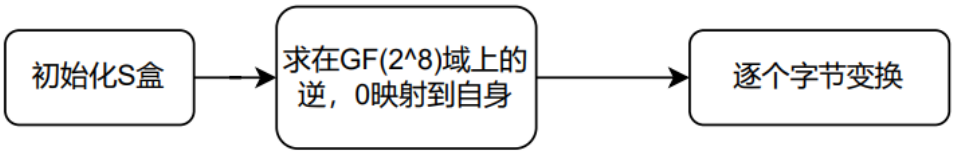
   利用3中实现的程序，对AES的密文进行雪崩效应检验。即固定密钥，仅改变明文中的一位，统计密文改变的位数；固定明文，仅改变密钥中的一位，统计密文改变的位数。

五. AES加密与解密

1. S盒产生

整体框架

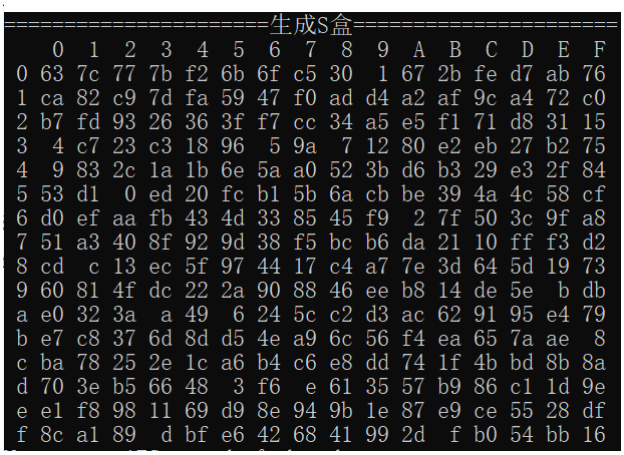
流程图



代码

```
void s_box_gen(void)
{
    int i, j;
    int s_box_ary[16][16] = { 0 };

    for (i = 0; i < 0x10; i++)
        for (j = 0; j < 0x10; j++)
            s_box_ary[i][j] = ((i << 4) & 0xF0) + (j & (0xF));

    for (i = 0; i < 0x10; i++)
    {
        for (j = 0; j < 0x10; j++)
            if (s_box_ary[i][j] != 0)
                s_box_ary[i][j] = externEuc(s_box_ary[i][j], 0x11B);
    }

    for (i = 0; i < 0x10; i++)
    {
        for (j = 0; j < 0x10; j++)
        {
            s_box_ary[i][j] = byteTransformation(s_box_ary[i][j],
0x63);
            S[i][j] = s_box_ary[i][j];
        }
    }
}
```

验证



GF(2^8)多项式的扩展欧几里得算法

代码

```
int externEuc(int a, int m)
{
    int r0, r1, r2;
    int  qn, v0, v1, v2, w0, w1, w2;
    r0 = m;
    r1 = a;
    v0 = 1;
    v1 = 0;
    w0 = 0;
    w1 = 1;
    while (r1 != 1)
    {
        qn = Gfdiv(r0, r1, &r2);
        v2 = v0 ^ GfMulti(qn, v1);
        w2 = w0 ^ GfMulti(qn, w1);
        r0 = r1;
        r1 = r2;
        v0 = v1;
        v1 = v2;
        w0 = w1;
        w1 = w2;
    }
    return w1;
}
```

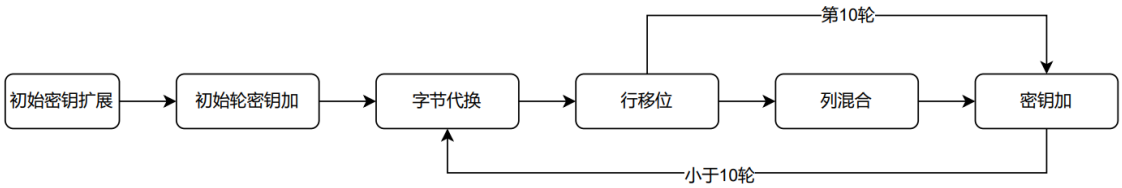
S盒字节变换

代码

```c
int byteTransformation(int a, int x)
{
    int tmp[8] = { 0 };

    for (int i = 0; i < 8; i++)
    {
        tmp[i] = (((a >> i) & 0x1) ^ ((a >> ((i + 4) % 8)) & 0x1) ^ ((a >> ((i + 5) % 8)) & 0x1) ^ ((a >> ((i + 6) % 8)) & 0x1) ^ ((a >> ((i + 7) % 8)) & 0x1) ^ ((x >> i) & 0x1)) << i;
    }
    tmp[0] = tmp[0] + tmp[1] + tmp[2] + tmp[3] + tmp[4] + tmp[5] + tmp[6] + tmp[7];
    return tmp[0];
}
```

2. 加密函数

- **流程图**

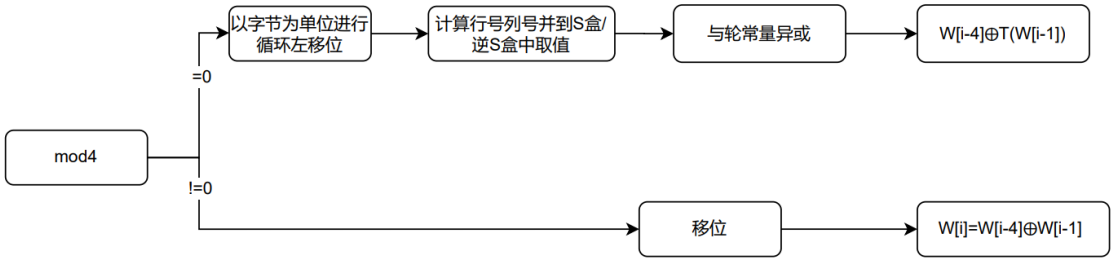


- **代码**

```c
void Encrypt(int in[4][4], int key[4][4])
{
    int type = 1;
    int subKey[11][4][4];
    KeyExpansion(key, subKey);    // 密钥扩展
    AddRoundKey(in, subKey[0]);     //轮密钥加
    for (int i = 1; i <= 10; ++i)
    {
        ByteSub(in, type);  //字节代换
        shiftRow(in, type);    //行移位
        if (i != 10)      //最后一次计算不需要列混合
            mixCol(in, type);   ///列混合
        AddRoundKey(in, subKey[i]);   //密钥加
    }
}
```

3. 密钥扩展

- **流程图**

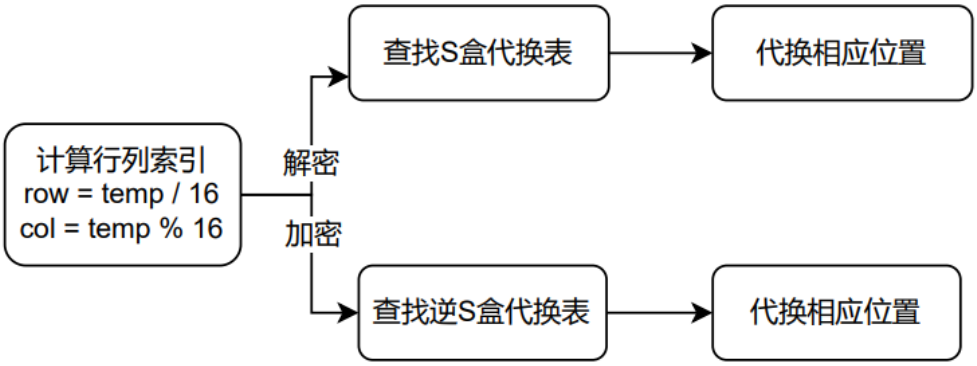


- **代码**

```
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];

    for (int i = 1; i < 11; ++i)
    {
        for (int j = 0; j < 4; ++j)
        {
            int temp[4];
            if (j == 0)
            {
                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)
                        temp[k] = temp[k] ^ rC[i - 1];
                }
            }
            else
            {
                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];
            }

            for (int x = 0; x < 4; x++)
                w[i][j][x] = w[i - 1][j][x] ^ temp[x];
        }
    }
}
```

4. 字节代换

- **流程图**



- **代码**

  加密与解密字节代换不同点在于S盒与逆S盒

```
void ByteSub(int in[4][4], int type)
{
    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

- **各行进行循环移位**
- **代码**

```
void ShiftRow(int in[4][4], int type) {
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < i; j++)
        {
            if (type == 1)
            {
                int temp = in[i][0];
                in[i][0] = in[i][1];
                in[i][1] = in[i][2];
                in[i][2] = in[i][3];
                in[i][3] = temp;
```

```
            }
            else
            {
                int temp = in[i][3];
                in[i][3] = in[i][2];
                in[i][2] = in[i][1];
                in[i][1] = in[i][0];
                in[i][0] = temp;
            }
        }
    }
}
```

6. 列混合

- **代码**

```
void MixColumn(int in[4][4], int type)
{
    for (int i = 0; i < 4; i++)
    {
        int t0 = in[0][i];
        int t1 = in[1][i];
        int t2 = in[2][i];
        int t3 = in[3][i];
        if (type == 1)
        {
            in[0][i] = aesMult(t0, 2) ^ aesMult(t1, 3) ^ t2 ^ t3;
            in[1][i] = t0 ^ aesMult(t1, 2) ^ aesMult(t2, 3) ^ t3;
            in[2][i] = t0 ^ t1 ^ aesMult(t2, 2) ^ aesMult(t3, 3);
            in[3][i] = aesMult(t0, 3) ^ t1 ^ t2 ^ aesMult(t3, 2);
        }
        else
        {
            in[0][i] = aesMult(t0, 14) ^ aesMult(t1, 11) ^
aesMult(t2, 13) ^ aesMult(t3, 9);
            in[1][i] = aesMult(t0, 9) ^ aesMult(t1, 14) ^ aesMult(t2,
11) ^ aesMult(t3, 13);
            in[2][i] = aesMult(t0, 13) ^ aesMult(t1, 9) ^ aesMult(t2,
14) ^ aesMult(t3, 11);
            in[3][i] = aesMult(t0, 11) ^ aesMult(t1, 13) ^
aesMult(t2, 9) ^ aesMult(t3, 14);
        }
    }
}
```

7. 轮密钥加

- **将轮密钥与状态进行逐比特异或**

- **代码**

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

七. 运行结果

1. 加密

十六进制密钥：0001, 2001, 7101, 98ae, da79, 1714, 6015, 3594

十六进制明文：0001, 0001, 01a1, 98af, da78, 1734, 8615, 3566

十六进制密文：6cdd, 596b, 8f56, 42cb, d23b, 4798, 1a65, 422a

```
=======请选择：  1——加密  2——解密 ======
1
=================加密==================
请输入128位明文（十六进制）：
00 01 00 01 01 a1 98 af da 78 17 34 86 15 35 66
请输入128位密钥（十六进制）：
00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94

十六进制密文为：
6cdd 596b 8f56 42cb d23b 4798 1a65 422a
=======================================
```

2. 解密

十六进制密钥：2b7e, 1516, 28ae, d2a6, abf7, 1588, 09cf, 4f3c

十六进制密文：3925, 841d, 02dc, 09fb, dc11, 8597, 196a, 0b32

十六进制明文：3243, f6a8, 885a, 308d, 3131, 98a2, e037, 0734

```
=======请选择：  1——加密  2——解密 ======
2
=================解密==================
请输入128位密文（十六进制）：
39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32
请输入128位密钥（十六进制）：
2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
十六进制明文为：
3243 f6a8 885a 308d 3131 98a2 e037 0734
=======================================
```

八. 检验雪崩效应

1. 明文雪崩效应

◦ 核心代码

```
cout << "========== 2--检验明文雪崩效应 =========" << endl;
cout << "请输入128位明文（十六进制）: " << endl;
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
    {
        cin >> (hex) >> txt[j][i];
        encrypt_text_original[j][i] = txt[j][i];
    }

cout << "请输入128位密钥（十六进制）: " << endl;
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
        cin >> (hex) >> key[j][i];
cout << endl;
Encrypt(encrypt_text_original, key);  //加密过程

cout << endl;
cout << "=======================================" << endl << endl;
// 分别改变明文每一位检验雪崩效应
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 4; j++)
    {
        for (int k = 0; k < 8; k++)
        {
            int temp_key[4][4], temp_txt[4][4];
            for (int u = 0; u < 4; u++)
                for (int v = 0; v < 4; v++)
                    temp_txt[v][u] = txt[j][i];

            bitset<8> m = Hex_Bin(temp_txt[j][i]);
            m[k] = 1 - m[k];
            int res = 0;
            for (int i = 0; i < 8; i++)
            {
                res *= 2;
                res += m[7 - i];
            }
            temp_txt[j][i] = res;
            Encrypt(temp_txt, key);  //加密过程

            int num = 0;
            for (int u = 0; u < 4; u++)
                for (int v = 0; v < 4; v++)
                {
                    bitset<8> bit_current = Hex_Bin(temp_txt[v][u]);
                    bitset<8> bit_original =
Hex_Bin(encrypt_text_original[v][u]);
                    for (int x = 0; x < 8; x++)
                        if (bit_current[x] != bit_original[x])
                            num++;
```

```
                }
            total += num;
            int index = (i * 32 + j * 8 + k);
            if ( index % 10 == 0)
                cout << "改变第" << index << "位明文-密文改变位数为：" << num << endl;
        }
    }
}
cout << setprecision(4) << total / 128 << endl;
cout << "========================================" << endl;
cout << endl << "========== 1--检验密钥雪崩效应  2--检验明文雪崩效应 =========" << endl;
continue;
```

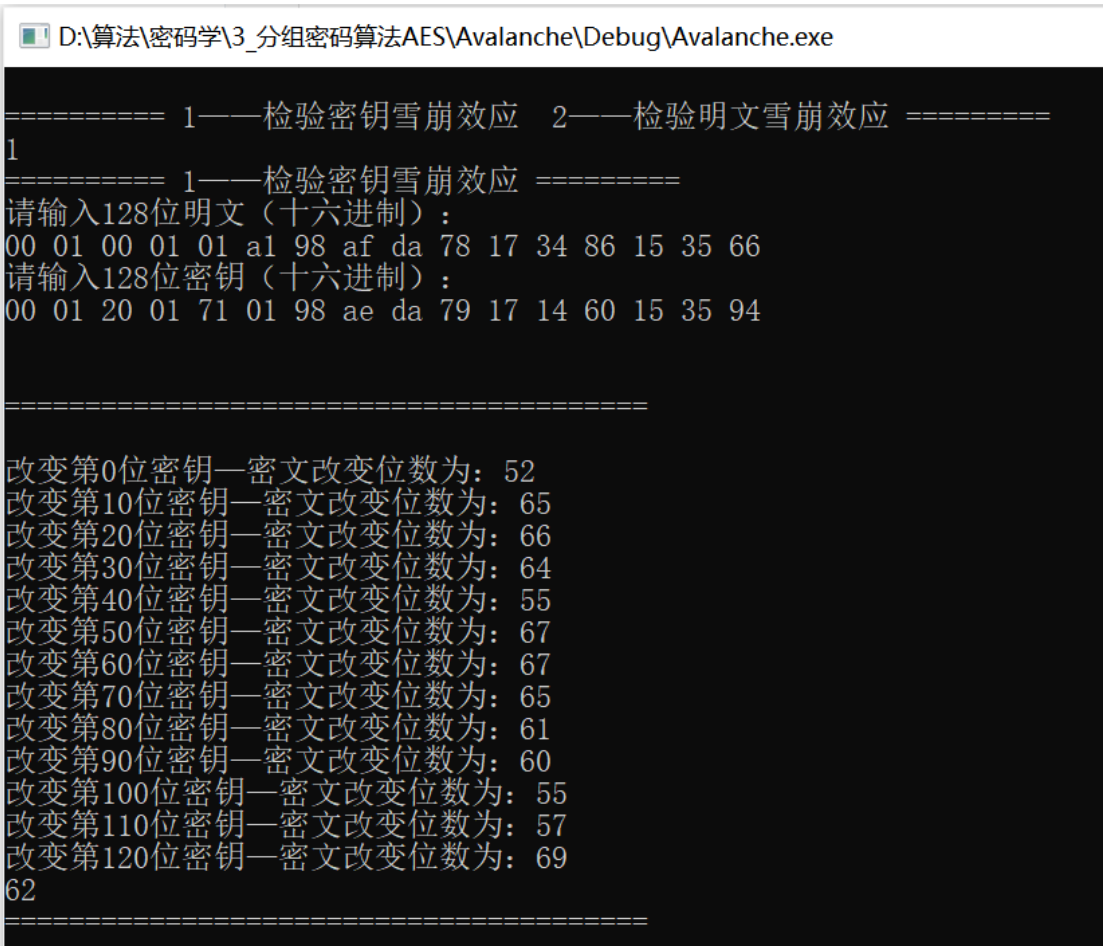
- 128位明文逐一改变，下图中每改变10次输出一次，计算得出平均密文改变的位数位64

```
D:\算法\密码学\3_分组密码算法AES\Avalanche\Debug\Avalanche.exe

========== 1———检验密钥雪崩效应  2———检验明文雪崩效应 =========
2
========== 2———检验明文雪崩效应 =========
请输入128位明文（十六进制）：
00 01 00 01 01 a1 98 af da 78 17 34 86 15 35 66
请输入128位密钥（十六进制）：
00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94


========================================

改变第0位明文—密文改变位数为：62
改变第10位明文—密文改变位数为：64
改变第20位明文—密文改变位数为：67
改变第30位明文—密文改变位数为：61
改变第40位明文—密文改变位数为：65
改变第50位明文—密文改变位数为：73
改变第60位明文—密文改变位数为：59
改变第70位明文—密文改变位数为：57
改变第80位明文—密文改变位数为：78
改变第90位明文—密文改变位数为：68
改变第100位明文—密文改变位数为：64
改变第110位明文—密文改变位数为：72
改变第120位明文—密文改变位数为：68
64
========================================
```

2. 密钥雪崩效应

代码与明文雪崩效应检验大同小异，因此不再赘述

128位密钥逐一改变，下图中每改10次输出一次，计算出平均密文改变位数为62


D:\算法\密码学\3_分组密码算法AES\Avalanche\Debug\Avalanche.exe
========== 1——检验密钥雪崩效应 2——检验明文雪崩效应 =========
1
========== 1——检验密钥雪崩效应 =========
请输入128位明文（十六进制）：
00 01 00 01 01 a1 98 af da 78 17 34 86 15 35 66
请输入128位密钥（十六进制）：
00 01 20 01 71 01 98 ae da 79 17 14 60 15 35 94
==
改变第0位密钥—密文改变位数为：52
改变第10位密钥—密文改变位数为：65
改变第20位密钥—密文改变位数为：66
改变第30位密钥—密文改变位数为：64
改变第40位密钥—密文改变位数为：55
改变第50位密钥—密文改变位数为：67
改变第60位密钥—密文改变位数为：67
改变第70位密钥—密文改变位数为：65
改变第80位密钥—密文改变位数为：61
改变第90位密钥—密文改变位数为：60
改变第100位密钥—密文改变位数为：55
改变第110位密钥—密文改变位数为：57
改变第120位密钥—密文改变位数为：69
62
==