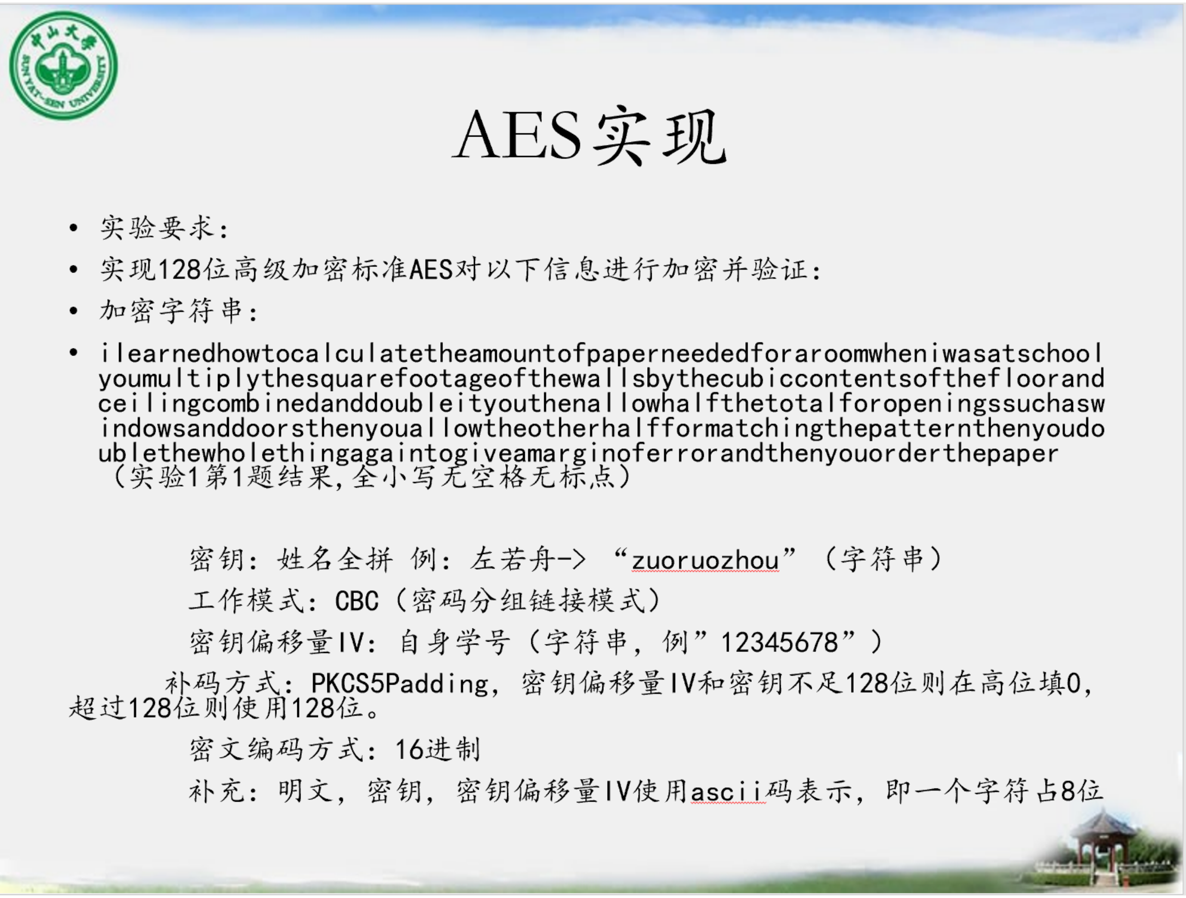
《现代密码学》实验报告

实验名称：AES实现	实验时间：2022.10.19
学生姓名：伍建霖	学号：20337251
学生班级：20网安	成绩评定：

一、实验目的

通过实现在CBC模式下使用128位AES加密信息，理解AES具体结构以及实现细节，提高代码水平。

二、实验内容



三、实验原理

先将AES加密算法分成三部分：首轮，中间轮，结束轮。（128位的明文，密钥还有密钥偏移）

首轮：AddRoundKey；

10轮中间轮：SubBytes，ShiftRows，MixColumns，AddRoundKey；

结束轮：SubBytes，ShiftRows，AddRoundKey

操作解释如下：

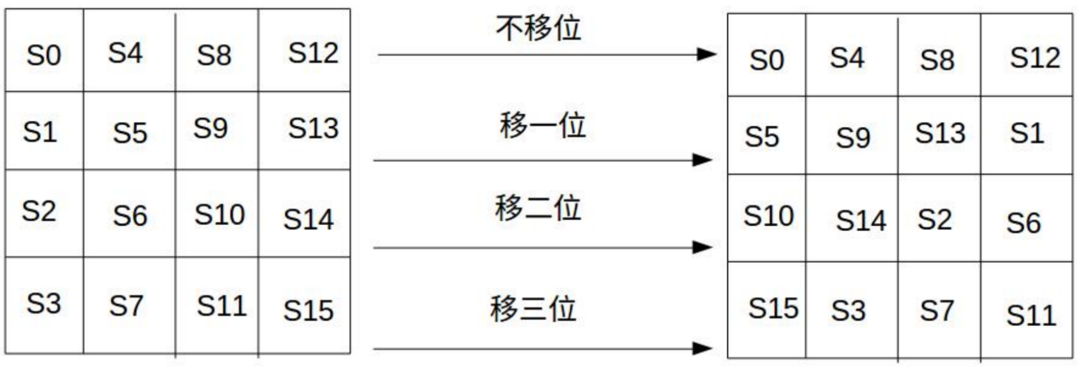
SubBytes

字节代换有两种办法：第一种是。但是这种办法实现起来过于麻烦，所以我选择了第二种办法：查表代换，每两个字节的十六进制形式作为行和列，查表得到对应值，直接代换即可。

行/列	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0x63	0x7c	0x77	0x7b	0xf2	0x6b	0x6f	0xc5	0x30	0x01	0x67	0x2b	0xfe	0xd7	0xab	0x76
1	0xca	0x82	0xc9	0x7d	0xfa	0x59	0x47	0xf0	0xad	0xd4	0xa2	0xaf	0x9c	0xa4	0x72	0xc0
2	0xb7	0xfd	0x93	0x26	0x36	0x3f	0xf7	0xcc	0x34	0xa5	0xe5	0xf1	0x71	0xd8	0x31	0x15
3	0x04	0xc7	0x23	0xc3	0x18	0x96	0x05	0x9a	0x07	0x12	0x80	0xe2	0xeb	0x27	0xb2	0x75
4	0x09	0x83	0x2c	0x1a	0x1b	0x6e	0x5a	0xa0	0x52	0x3b	0xd6	0xb3	0x29	0xe3	0x2f	0x84
5	0x53	0xd1	0x00	0xed	0x20	0xfc	0xb1	0x5b	0x6a	0xcb	0xbe	0x39	0x4a	0x4c	0x58	0xcf
6	0xd0	0xef	0xaa	0xfb	0x43	0x4d	0x33	0x85	0x45	0xf9	0x02	0x7f	0x50	0x3c	0x9f	0xa8
7	0x51	0xa3	0x40	0x8f	0x92	0x9d	0x38	0xf5	0xbc	0xb6	0xda	0x21	0x10	0xff	0xf3	0xd2
8	0xcd	0x0c	0x13	0xec	0x5f	0x97	0x44	0x17	0xc4	0xa7	0x7e	0x3d	0x64	0x5d	0x19	0x73
9	0x60	0x81	0x4f	0xdc	0x22	0x2a	0x90	0x88	0x46	0xee	0xb8	0x14	0xde	0x5e	0x0b	0xdb
A	0xe0	0x32	0x3a	0x0a	0x49	0x06	0x24	0x5c	0xc2	0xd3	0xac	0x62	0x91	0x95	0xe4	0x79
B	0xe7	0xc8	0x37	0x6d	0x8d	0xd5	0x4e	0xa9	0x6c	0x56	0xf4	0xea	0x65	0x7a	0xae	0x08
C	0xba	0x78	0x25	0x2e	0x1c	0xa6	0xb4	0xc6	0xe8	0xdd	0x74	0x1f	0x4b	0xbd	0x8b	0x8a
D	0x70	0x3e	0xb5	0x66	0x48	0x03	0xf6	0x0e	0x61	0x35	0x57	0xb9	0x86	0xc1	0x1d	0x9e
E	0xe1	0xf8	0x98	0x11	0x69	0xd9	0x8e	0x94	0x9b	0x1e	0x87	0xe9	0xce	0x55	0x28	0xdf
F	0x8c	0xa1	0x89	0x0d	0xbf	0xe6	0x42	0x68	0x41	0x99	0x2d	0x0f	0xb0	0x54	0xbb	0x16

ShiftRows

以每两个字节作为字节块（即下图中的S0，S1等等），将state（约等于该阶段的明文）表示成一个矩阵。



接着第一行不动，二到四行的每一块分别向右移一到三位，即如上图所示。

MixColumns

列混合操作按照老师课上说的和课本写的则为 b(x) = c(x) * state的一列， 即如下图所示。

$$\begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

但和字节代换一样，完全按照课本上的实现会很麻烦，所以我参考了网上的一种列混合实现方式，可以起到同样的效果。

其中，矩阵元素的乘法和加法都是定义在基于GF(2^8)上的二元运算,并不是通常意义上的乘法和加法。这里涉及到一些信息安全上的数学知识，不过不懂这些知识也行。其实这种二元运算的加法等价于两个字节的异或，乘法则复杂一点。对于一个8位的二进制数来说，使用域上的乘法乘以(00000010)等价于左移1位(低位补0)后，再根据情况同(00011011)进行异或运算，设S1 = (a7 a6 a5 a4 a3 a2 a1 a0)，刚0x02 * S1如下图所示：

$$(00000010) * (a_7a_6a_5a_4a_3a_2a_1a_0) = \begin{cases} (a_6a_5a_4a_3a_2a_1a_00), a_7 = 0 \\ (a_6a_5a_4a_3a_2a_1a_00) \oplus (00011011), a_7 = 1 \end{cases}$$

也就是说，如果a7为1，则进行异或运算，否则不进行。
类似地，乘以(00000100)可以拆分成两次乘以(00000010)的运算：

$$(00000100) * (a_7a_6a_5a_4a_3a_2a_1a_0) = (00000010) * (00000010) * (a_7a_6a_5a_4a_3a_2a_1a_0)$$

乘以(0000 0011)可以拆分成先分别乘以(0000 0001)和(0000 0010)，再将两个乘积异或：

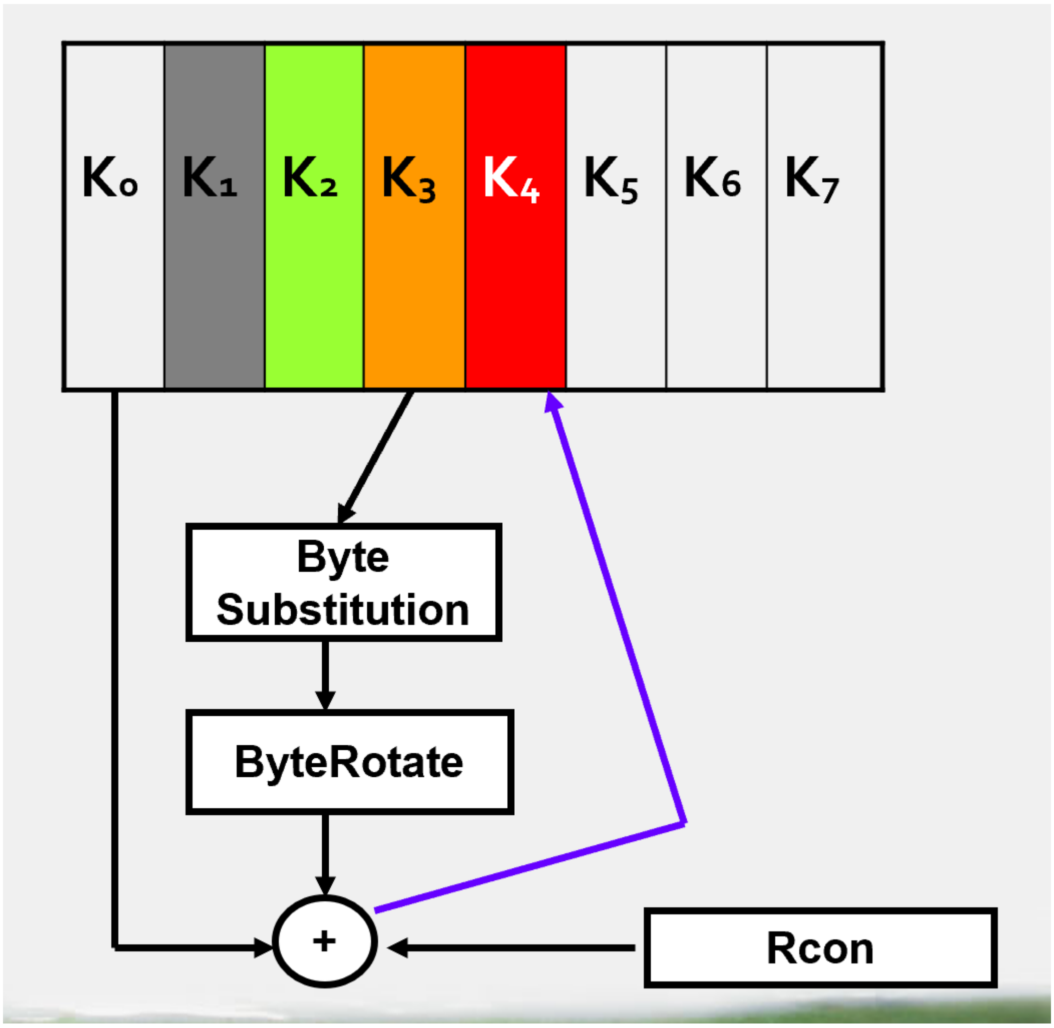
$$\begin{aligned}(00000011) * (a_7a_6a_5a_4a_3a_2a_1a_0) &= [(00000010) \oplus (00000001)] * (a_7a_6a_5a_4a_3a_2a_1a_0) \\ &= [(00000010) * (a_7a_6a_5a_4a_3a_2a_1a_0)] \oplus (a_7a_6a_5a_4a_3a_2a_1a_0)\end{aligned}$$

因此，我们只需要实现乘以2的函数，其他数值的乘法都可以通过组合来实现。

AddRoundKey

将state和该轮的密钥异或。

密钥扩展



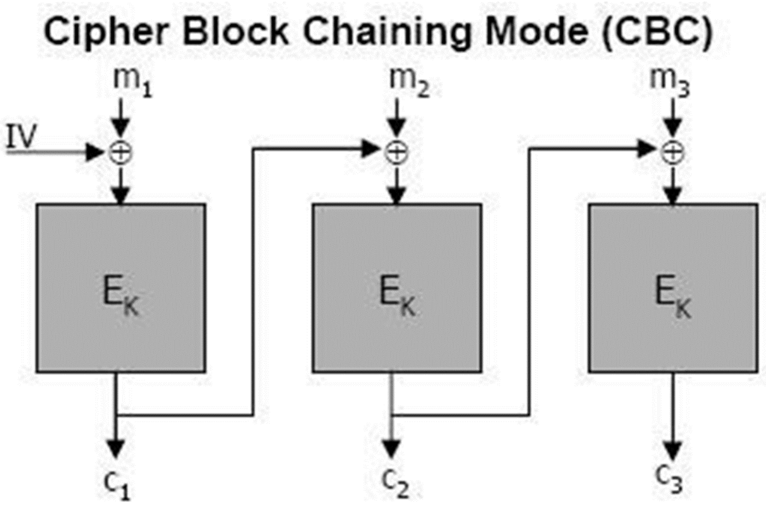
CBC工作模式

填充

题目要求使用PKCS5Padding：若明文需要补充N个字节后比特长度才为128的倍数，则在明文后添加N个0x0N；若明文刚好是128bit的倍数，则在最后补码16个0x10。

密钥和密钥偏移若不足128位则在高位填充0，使其达到128位。

机制



四、实验步骤

```
#include <iostream>
#include <string>
#include <vector>
#include <bitset>
#include <cassert>
using std::vector;
using byte = std::bitset<8>;

const vector<byte> rcon{
    0x01,
    0x00,
    0x00,
    0x00,
    0x02,
    0x00,
    0x00,
    0x00,
    0x04,
    0x00,
    0x00,
    0x00,
    0x08,
    0x00,
    0x00,
    0x00,
    0x10,
    0x00,
    0x00,
    0x00,
    0x20,
```

```
    0x00,
    0x00,
    0x00,
    0x40,
    0x00,
    0x00,
    0x00,
    0x80,
    0x00,
    0x00,
    0x00,
    0x1B,
    0x00,
    0x00,
    0x00,
    0x36,
    0x00,
    0x00,
    0x00,
};

const vector<byte> s_box{
    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};
```

```
vector<byte> next_key(vector<byte> former_key, size_t turn)
{
    assert(former_key.size() == 16);
    vector<byte> ret;

    vector<byte> former_key_last_word_after_rot{
        former_key[13], former_key[14],
        former_key[15], former_key[12]};
    size_t cnt = 0;
    for (byte elem : former_key_last_word_after_rot)
    {
        size_t col_index = (elem & byte(0x0F)).to_ulong();
        size_t row_index = (elem & byte(0xF0)).to_ulong() >> 4;
        byte res{former_key[cnt] ^ s_box[row_index * 16 + col_index] ^ rcon[turn
* 4 + cnt]};
        ret.push_back(res);
        ++cnt;
    }

    for (size_t standard = 4; standard < 16; standard += 4)
    {
        ret.push_back(former_key[standard] ^ ret[standard - 4]);
        ret.push_back(former_key[standard + 1] ^ ret[standard - 3]);
        ret.push_back(former_key[standard + 2] ^ ret[standard - 2]);
        ret.push_back(former_key[standard + 3] ^ ret[standard - 1]);
    }
    return ret;
}

vector<vector<byte>> Key_Expansion(vector<byte> input, size_t total_turn)
{
    vector<vector<byte>> ret{input};
    for (size_t index = 0; index < total_turn; ++index)
        ret.push_back(next_key(ret[index], index));
    return ret;
}

vector<byte> byte_to_vector(std::bitset<128> input)
{
    /* cos tmp is reverse */
    vector<byte> ret, tmp;
    for (size_t index = 0; index < 128 / 8; ++index)
    {
        tmp.push_back(std::bitset<8>{
            (input & std::bitset<128>{0xFF}).to_ulong()});
        input >>= 8;
    }
    while (tmp.size())
    {
        ret.push_back(tmp.back());
        tmp.pop_back();
    }
    return ret;
}
```

```
vector<vector<byte>> vector_to_2_vector(vector<byte> input)
{
    assert(input.size() == 16);
    return vector<vector<byte>>{
        vector<byte>{input[0], input[1], input[2], input[3]},
        vector<byte>{input[4], input[5], input[6], input[7]},
        vector<byte>{input[8], input[9], input[10], input[11]},
        vector<byte>{input[12], input[13], input[14], input[15]},
    };
}

vector<byte> AddRoundKey(vector<byte> input, vector<byte> round_key)
{
    assert(input.size() == 16 && round_key.size() == 16);
    vector<byte> ret;
    for (size_t index = 0; index < input.size(); ++index)
        ret.push_back(input[index] ^ round_key[index]);
    return ret;
}

vector<byte> SubBytes(vector<byte> input)
{
    assert(input.size() == 16);
    vector<byte> ret;
    for (byte elem : input)
    {
        size_t col_index = (elem & byte(0x0F)).to_ulong();
        size_t row_index = (elem & byte(0xF0)).to_ulong() >> 4;
        ret.push_back(s_box[row_index * 16 + col_index]);
    }
    return ret;
}

vector<byte> ShiftRows(vector<byte> input)
{
    return vector<byte>{
        input[0], input[5], input[10], input[15],
        input[4], input[9], input[14], input[3],
        input[8], input[13], input[2], input[7],
        input[12], input[1], input[6], input[11]};
}

byte FieldMult(byte input)
{
    std::bitset<1> top{input[7]};
    byte tmp{input << 1};
    if (top == 1)
        return tmp ^ byte(0x1b);
    else
        return tmp;
}

vector<byte> MixColumns(vector<byte> input)
{
    vector<byte> ret;
```

```
    vector<vector<byte>> total{vector_to_2_vector(input)};
    for (vector<byte> s : total)
    {
        byte x0{s[0]}, x1{s[1]}, x2{s[2]}, x3{s[3]};
        s[0] = x1 ^ x2 ^ x3;
        s[1] = x0 ^ x2 ^ x3;
        s[2] = x0 ^ x1 ^ x3;
        s[3] = x0 ^ x1 ^ x2;

        x0 = FieldMult(x0);
        x1 = FieldMult(x1);
        x2 = FieldMult(x2);
        x3 = FieldMult(x3);

        s[0] = s[0] ^ x0 ^ x1;
        s[1] = s[1] ^ x1 ^ x2;
        s[2] = s[2] ^ x2 ^ x3;
        s[3] = s[3] ^ x3 ^ x0;

        for (byte s_i : s)
            ret.push_back(s_i);
    }
    return ret;
}

vector<byte> aes(std::bitset<128> input, std::bitset<128> key, size_t round)
{
    vector<vector<byte>> key_expan{Key_Expansion(byte_to_vector(key), round)};
    vector<byte> vec{byte_to_vector(input)};
    assert(key_expan.size() == round + 1);

    vec = AddRoundKey(vec, key_expan[0]);

    size_t index;
    for (index = 1; index < round; ++index)
    {
        vec = SubBytes(vec);
        vec = ShiftRows(vec);
        vec = MixColumns(vec);
        vec = AddRoundKey(vec, key_expan[index]);
    }

    vec = SubBytes(vec);
    vec = ShiftRows(vec);
    vec = AddRoundKey(vec, key_expan[index]);
    return vec;
}

std::string string_to_bitstring(std::string str)
{
    std::string retstr;
    for (int i = 0; i < str.length(); i++)
    {
        std::bitset<8> temp(str[i]);
        retstr += temp.to_string();
```

```cpp
    }
    return retstr;
}

std::string PKCS5Padding(std::string str)
{
    if (!str.length() % 128)
        return str + "10101010101010101010101010101010";

    int i = 1;
    for (;; i++)
        if (i * 128 > str.length())
            break;
    int n = (i * 128 - str.length()) / 8;
    for (int i = 0; i < n; i++)
    {
        std::bitset<8> temp(n);
        str += temp.to_string();
    }
    return str;
}

std::bitset<128> vector_to_byte(vector<byte> input)
{
    std::string retstr;
    for (byte elem : input)
    {
        retstr = elem.to_string() + retstr;
    }
    std::bitset<128> ret(retstr);
    return ret;
}

int main()
{
    std::string raw_ciphertext =
"ilearnedhowtocalculatetheamountofpaperneededforaroomwheniwasatschoolyoumultiplythesquarefootageofthewallsbythecubiccontentsofthefloorandceilingcombinedanddoubleityouthenallowhalfthetotalforopeningssuchaswindowsanddoorsthenyouallowtheotherhalfformatchingthepatternthenyoudoublethewholethingagaintogiveamarginoferrorandthenyouorderthepaper";
    std::string bitstr_ct_padding =
PKCS5Padding(string_to_bitstring(raw_ciphertext));
    std::bitset<128> key(string_to_bitstring("wujianlin"));
    std::bitset<128> IV(string_to_bitstring("20337251"));

    for (int i = 0; i < 3; i++)
    {
        std::bitset<128> subct(bitstr_ct_padding.substr(i, i + 128));
        vector<byte> res{aes(subct ^ IV, key, 10)};
        for (byte elem : res)
            std::cout << elem << " ";
        std::cout << '\n';
        IV = vector_to_byte(res);
    }
```

```
}

// 密钥 wujianlin
// 明文
ilearnedhowtocalculatetheamountofpaperneededforaroomwheniwasatschoolyoumultiplythesquarefootageofthewallsbythecubiccontentsofthefloorandceilingcombinedanddoubleityouthenallowhalfthetotalforopeningssuchaswindowsanddoorsthenyouallowtheotherhalfformatchingthepatternthenyoudoublethewholethingagaintogiveamarginoferrorandthenyouorderthepaper
// 密钥偏移 20337251
```

五、实验结果

```
10100010 01110100 11100111 11101000 10101010 11001011 10010010 11110110 10111100 10011011 10000000 11011001
00011101 01001000 00010011 01001001
11110001 01010000 01011110 10010100 01001111 01010010 00100100 01100011 11101000 11110001 00000101 00100110
01010010 01000011 01010100 01010100
01110101 11000101 11000110 11001010 00011000 01111111 01001001 11010001 10000001 01001010 01001101 11101010
11000000 00101110 00101000 01001000
(base) PS D:\CodeField\Cryptography\lab>
```

六、实验总结

https://www.youtube.com/watch?v=gP4PqVGudtg

AES加密算法的详细介绍与实现*TimeShatter的博客-CSDN博客*aes

从这次实验中我学习到了如何实现AES，了解实现AES过程中的难点。这次实验基本上就照着老师课上讲的流程做即可。