《现代密码学》实验报告

实验名称：AES-128 的实现	**实验时间**：2025 年 10 月 29 日
学生姓名：谢润文	**学号**：23336262
学生班级：23 计科	**成绩评定**：

一、实验目的

通过实现 AES-128 CBC 工作模式的加密和解密，理解分组密码的工作原理和 AES 生成轮密钥的过程及每一轮的迭代操作过程，提高分块处理的能力、比特串的操作能力和从伪代码到实现的代码编写能力。

二、实验内容

用 C++实现 AES-128 CBC 工作模式的加密和解密。

参数要求：输入数据由以下部分组成：

uint8_t 工作模式和加密/解密,其中 0x01 表示 CBC 模式加密,0x81 表示 CBC 模式解密

uint8_t[16] 密钥

uint8_t[16] 初始向量（Initial Vector,IV）

uint32_t 明文/密文长度

uint8_t[] 明文/密文内容

采用 PKCS#7 Padding。即如果需要填充 N（N>0） bytes 后长度才是 16 bytes 的倍数，则需要填充 N 个 0xON。

三、实验原理

0. AES 提要

0.1 一般的分组密码

分组密码是一种对称密钥加密算法，将明文分成固定长度的块，并使用相同的密钥对每个块进行加密。其中一个典型的设计理念是 SPN 结构（Substitution-Permutation Network）。它通过交替的代换（Substitution）和置换（Permutation）操作来加密数据，代换操作通过 S 盒实现，而置换操作则通过 P 盒实现。

0.2 AES

Advanced Encryption Standard（AES，高级加密标准）由 Joan Daemen 和 Vincent Rijmen 所设计，是典型的块加密，被设计来取代 DES。AES 也采用了 SPN 结构，但是其处理是在矩阵上操作的而非直接对原始字符串处理。AES 将 16 字节的明文看做一个 4*4 的二维矩阵，分别为 s1、s2、……、s15。AES 对这个矩阵的行、列和字节进行变

形，生成最后的密文。AES 分组可以变长，提供 128、192 和 256 三种明文分组长度，AES-128 表示以 128 字节作为分组。

1. 轮密钥的生成

在 AES-128 中，轮密钥有 10 轮，由一个初始密钥生成，其生成伪代码如下：

```
KeyExpansion(byte key[16], word w[44], Nk)
begin
        word temp,i = 0
        while(i < 4){
                w[i]   =   word(key[4*i],   key[4*i+1],   key[4*i+2],
key[4*i+3])
                i = i+1
        }
        i = 4
        while (i < 44){
                temp = w[i-1]
                if (i mod 4 = 0) temp = SubWord(RotWord(temp)) xor
Rcon[i/4]
                w[i] = w[i-4] xor temp
                i = i + 1
        }
end
```

其中，SubWord 和 RotWord 分别是对字中每一个字节做 S 盒代换和行移位。

2. 加密过程

2.1 迭代主流程

AES-128 的主流程是 10 轮的迭代加密，除了最后一轮不进行列混合，每一轮都进行字节代换、行移位、列混合和轮密钥加四步操作。加密伪代码如下：

```
Cipher(byte in[16], byte out[16], word w[44])
begin
        byte state[4][4]
        state = in
        AddRoundKey(state, w[0, 3])
        for round = 1 step 1 to 9
                SubBytes(state)
                ShiftRows(state)
```

```
                MixColumns(state)
                AddRoundKey(state, w[round*4, (round+1)*4-1])
        end for
        SubBytes(state)
        ShiftRows(state)
        AddRoundKey(state, w[40, 43])
        out = state
end
```

2.2 初始化与 AddRoundKey 操作

初始为 in[0...15]，下标对应 4*4 矩阵如下：

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

同理将 w[0...4]按字节填入 4*4 的矩阵中，二者异或相加即可。这样做的目的是让轮密钥参与加密过程。

2.3 SubByte 操作

SubByte 操作相当于字节替换。在 AES 中，S 盒的构造如下：

1）对每个元素在 GF（2^8）上求逆，0 的逆是其本身。

2）对于第 i 个比特 $b_i=b_{(i+4)mod8}$ xor $b_{(i+5)mod8}$ xor $b_{(i+6)mod8}$ xor $b_{(i+7)mod\ 8}$ xor c_i，其中 c 是{01100011}的常比特串。

这样便实现了一个 16*16 的 S 盒，在软件实现上一般采用提前列出 S 盒表格然后查询的形式，对于不方便存储的硬件则计算亦不复杂。可以证明求逆后再进行矩阵乘，其结果不是原始字节的线性表示。

2.4 ShiftRows 操作

在 ShiftRows()操作中，矩阵最后三行的字节会按不同的字节数（偏移量）循环移位。第一行不进行移位。具体表达式如下：

$$s_{r,c}=s_{r,(c+r)mod4}，\text{其中 } 0<r<4,0<=c<4$$

这将导致字节移动到行中的“更低”位置，从而保证矩阵在行上的混乱度。

2.5 MixColumns 操作

列混合操作主要是让各列元素充分混合，可以表达为 $a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$的形式，其中大括号表示 GF($2^8$)中的元素，{01}表示 1，{02}表示 x，{03}表示 x+1，

以此类推。对于每一个矩阵中具体的元素，可以进一步得出表达式如下：

$$s_{0,c}' = (\{02\} * s_{0,c}) \hat{} (\{03\} * s_{1,c}) \hat{} s_{2,c} \hat{} s_{3,c}$$

$$s_{1,c}' = s_{0,c} \hat{} (\{02\} * s_{1,c}) \hat{} (\{03\} * s_{2,c}) \hat{} s_{3,c}$$

$$s_{2,c}' = s_{0,c} \hat{} s_{1,c} \hat{} (\{02\} * s_{2,c}) \hat{} (\{03\} * s_{3,c})$$

$$s_{3,c}' = (\{03\} * s_{0,c}) \hat{} s_{1,c} \hat{} s_{2,c} \hat{} (\{02\} * s_{3,c})$$

带’表示变换后的元素。可以看到，在 MixColumn 操作中需要 8 次有限域的乘法，乘法的效率较为关键。利用 https://github.com/jevillegasd/GF2-8/blob/master/gf28.cpp 中给出的根据俄罗斯数学家的方法对有限域乘法的实现，可以比较快地处理。该算法的思想类似平方-乘算法，利用第二个乘数的二进制展开进行操作。

3. 解密过程

解密过程的轮密钥和加密过程的轮密钥是同一个密钥。解密过程相当于逆着加密过程输出。伪代码如下：

```
InvCipher(byte in[16], byte out[16], word w[44])
begin
        byte state[4,4]
        state = in
        AddRoundKey(state, w[40, 43])
        for round = 9 step -1 downto 1
                InvShiftRows(state)
                InvSubBytes(state)
                AddRoundKey(state,w[round*4,(round+1)*4-1])
                InvMixColumns(state)
        end for
        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state, w[0, 3])
        out=state
end
```

3.1 InvShiftRows 操作与 InvSubBytes 操作

InvShiftRows 操作是 ShiftRows 操作的逆操作，即 $s_{r,(c+r) \bmod 4}=s_{r,c}$，其中 0<r<4,0<=c<4。InvSubBytes 操作则是 SubBytes 操作的逆操作，找出 S 盒的逆变换即可。实际上可以预处理（打表）出逆 S 盒，直接对应即可。

3.2 InvMixColumns 操作

InvMixColumns 操作是 MixColumns 操作的逆操作。事实上，MixColumns 操作可以将[$s_{i,c}$]

向量分离出，则做一次变换相当于矩阵乘向量，求逆相当于对矩阵在 GF(2^8)上求逆。实际上等价于下列操作：

$$s_{0,c}' = (\{0e\} * s_{0,c}) \hat{} (\{0b\} * s_{1,c}) \hat{} (\{0d\} * s_{2,c}) \hat{} (\{09\} * s_{3,c})$$

$$s_{1,c}' = (\{09\} * s_{0,c}) \hat{} (\{0e\} * s_{1,c}) \hat{} (\{0b\} * s_{2,c}) \hat{} (\{0d\} * s_{3,c})$$

$$s_{2,c}' = (\{0d\} * s_{0,c}) \hat{} (\{09\} * s_{1,c}) \hat{} (\{0e\} * s_{2,c}) \hat{} (\{0b\} * s_{3,c})$$

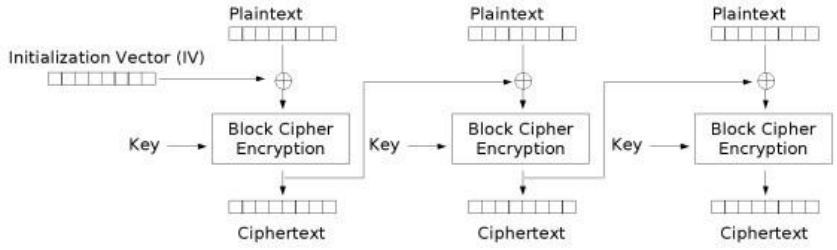
$$s_{3,c}' = (\{0b\} * s_{0,c}) \hat{} (\{0d\} * s_{1,c}) \hat{} (\{09\} * s_{2,c}) \hat{} (\{0e\} * s_{3,c})$$

4. 分组密码的工作模式

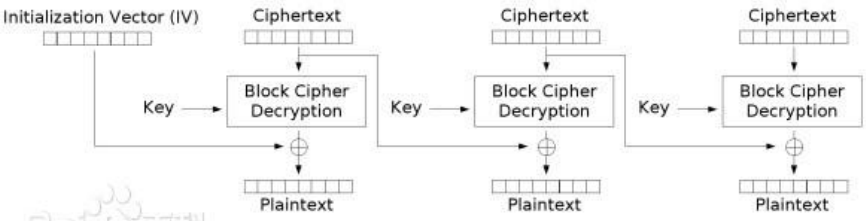
分组密码具有一些安全问题——如果使用相同的密钥对相同的明文进行加密，则会导致频率特征泄露。分组密码的工作模式可以对原有的分组密码算法进行加强，使分组密码能够应用于实际加密中。

4.1 CBC 模式

密码分组链接（Cipher-Block Chaining,CBC）是指将前一个生成的密文块通过反馈机制与新的明文块结合，每个明文块在加密前需与前一密文块异或。



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

图 1 CBC 工作模式下的加密和解密

注意到加密不可以并行处理，而解密可以并行处理。

四、实验步骤（源代码）

以下是没有使用 x64 CPU 的 AES 指令集（AES-NI）实现 AES-128 的源代码：

```
#include<iostream>
#include<cstdint>
#include<fstream>
```

```
    #include<fcntl.h>
    using namespace std;
    /*
    FILE *fp=fopen("dump.bin","rb");
    FILE *p=fopen("output.bin","wb");
    */

    #define fp stdin
    #define p stdout

    const                                                    uint8_t
rcon[11]={0,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36};
    const uint8_t S[256]={
    0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd
7,0xab,0x76,
    0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa
4,0x72,0xc0,
    0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd
8,0x31,0x15,
    0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x2
7,0xb2,0x75,
    0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe
3,0x2f,0x84,
    0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4
c,0x58,0xcf,
    0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3
c,0x9f,0xa8,
    0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xf
f,0xf3,0xd2,
    0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5
d,0x19,0x73,
    0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5
e,0x0b,0xdb,
    0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x9
```

```
5,0xe4,0x79,
    0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7
a,0xae,0x08,
    0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xb
d,0x8b,0x8a,
    0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc
1,0x1d,0x9e,
    0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x5
5,0x28,0xdf,
    0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x5
4,0xbb,0x16
    };
    const uint8_t inv_S[256]={
    0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf
3,0xd7,0xfb,
    0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xd
e,0xe9,0xcb,
    0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xf
a,0xc3,0x4e,
    0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8
b,0xd1,0x25,
    0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x6
5,0xb6,0x92,
    0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8
d,0x9d,0x84,
    0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb
3,0x45,0x06,
    0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x1
3,0x8a,0x6b,
    0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb
4,0xe6,0x73,
    0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x7
5,0xdf,0x6e,
    0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x1
```

```
8,0xbe,0x1b,
    0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xc
d,0x5a,0xf4,
    0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x8
0,0xec,0x5f,
    0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a,0x9f,0x93,0xc
9,0x9c,0xef,
    0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x5
3,0x99,0x61,
    0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x2
1,0x0c,0x7d
    };
    uint8_t gf_mul(uint8_t a,uint8_t b){
        uint8_t c=0;
        while(a&&b){
            if(b&1) c^=a;
            if(a&0x80) a=(a<<1)^0x11B;
            else a<<=1;
            b>>=1;
        }
        return c;
    }
    void key_expend(uint8_t key[],uint32_t w[]){
        int i;
        uint32_t tmp;
        for(i=0;i<4;++i){

    w[i]=(key[4*i]<<24)^(key[4*i+1]<<16)^(key[4*i+2]<<8)^(key[4*i+3]);
            //cout<<hex<<w[i]<<endl;
        }
        i=4;
        while(i<44){
            tmp=w[i-1];
            if(i%4==0){
```

```
            tmp=(tmp>>24)^(tmp<<8);//rotword
            //cout<<hex<<"rotword"<<tmp<<endl;
            uint8_t t_tmp[4];
            for(int j=0;j<4;++j){//subword
                uint8_t t=tmp;
                tmp>>=8;
                t_tmp[j]=S[t];
            }

   tmp=(t_tmp[0])^(t_tmp[1]<<8)^(t_tmp[2]<<16)^((t_tmp[3]^rcon[i/4])<<2
4);
          }
          w[i]=tmp^w[i-4];
          //cout<<hex<<w[i]<<endl;
          i++;
       }
   }
   void en_AES(uint8_t text[],uint32_t w[],uint8_t out[]){
       uint8_t state[4][4]={0};
       for(int i=0;i<4;++i){//add roundkey
           uint32_t tmp=w[i];
           for(int j=0;j<4;++j){
               state[i][j]^=text[4*j+i];
               uint8_t tmp_8=tmp;
               state[3-j][i]^=tmp_8;
               tmp>>=8;
           }
       }
       for(int rd=1;rd<=10;++rd){
           //cout<<"rd:"<<rd<<endl;
           uint8_t tmp[4][4];
           for(int i=0;i<4;++i){
               for(int j=0;j<4;++j){
                   state[i][j]=S[state[i][j]];//subbytes
```

```
                tmp[i][j]=state[i][j];
            }
        }
        for(int i=0;i<4;++i)  for(int j=0;j<4;++j)
state[i][j]=tmp[i][(i+j)%4];//shiftrows
        for(int i=0;i<4;++i)  for(int j=0;j<4;++j)
tmp[i][j]=state[i][j];
        if(rd!=10){
            for(int i=0;i<4;++i){//mixcolumns

    state[0][i]=gf_mul(0x02,tmp[0][i])^gf_mul(0x03,tmp[1][i])^tmp[2][i]^
tmp[3][i];

    state[1][i]=tmp[0][i]^gf_mul(0x02,tmp[1][i])^gf_mul(0x03,tmp[2][i])^
tmp[3][i];

    state[2][i]=tmp[0][i]^tmp[1][i]^gf_mul(0x02,tmp[2][i])^gf_mul(0x03,t
mp[3][i]);

    state[3][i]=gf_mul(0x03,tmp[0][i])^tmp[1][i]^tmp[2][i]^gf_mul(0x02,t
mp[3][i]);
            }
        }
        for(int i=0;i<4;++i){//add roundkey
            uint32_t w_tmp=w[4*rd+i];
            //cout<<hex<<w_tmp<<endl;
            for(int j=0;j<4;++j){
                uint8_t tmp_8=w_tmp;
                state[3-j][i]^=tmp_8;
                w_tmp>>=8;
            }
        }
        /*
        for(int i=0;i<4;++i){
```

```
            for(int       j=0;j<4;++j)       cout<<i<<"       "<<j<<"
"<<hex<<(int)state[i][j]<<" ";
            cout<<endl;
        }
        */
    }
    for(int i=0;i<4;++i){
        for(int j=0;j<4;++j){
            out[j*4+i]=state[i][j];
        }
    }
}
void de_AES(uint8_t text[],uint32_t w[],uint8_t out[]){
    uint8_t state[4][4]={0};
    for(int i=0;i<4;++i){//add roundkey
        uint32_t w_tmp=w[40+i];
        for(int j=0;j<4;++j){
            state[i][j]^=text[4*j+i];
            uint8_t tmp_8=w_tmp;
            state[3-j][i]^=tmp_8;
            w_tmp>>=8;
        }
    }
    for(int rd=9;rd>=0;--rd){
        /*
        cout<<"rd:"<<rd<<endl;
        for(int i=0;i<4;++i){
            for(int       j=0;j<4;++j)       cout<<i<<"       "<<j<<"
"<<hex<<(int)state[i][j]<<" ";
            cout<<endl;
        }
        */
        uint8_t tmp[4][4];
        for(int       i=0;i<4;++i)       for(int       j=0;j<4;++j)
```

```
tmp[i][j]=state[i][j];
            for(int i=0;i<4;++i){
                for(int j=0;j<4;++j){
                    state[i][(i+j)%4]=inv_S[tmp[i][j]];//invshiftrows
                }
            }
            /*
            cout<<"after invshiftrows & invsub:"<<endl;
            for(int i=0;i<4;++i){
                for(int       j=0;j<4;++j)       cout<<i<<"       "<<j<<"
"<<hex<<(int)state[i][j]<<" ";
                cout<<endl;
            }
            */
            for(int i=0;i<4;++i){//add roundkey
                uint32_t w_tmp=w[4*rd+i];
                //cout<<hex<<w_tmp<<endl;
                for(int j=0;j<4;++j){
                    uint8_t tmp_8=w_tmp;
                    state[3-j][i]^=tmp_8;
                    w_tmp>>=8;
                }
            }
            for(int        i=0;i<4;++i)        for(int        j=0;j<4;++j)
tmp[i][j]=state[i][j];
            if(rd){
                for(int i=0;i<4;++i){//invmixcolumns

    state[0][i]=gf_mul(0x0e,tmp[0][i])^gf_mul(0x0b,tmp[1][i])^gf_mul(0x0
d,tmp[2][i])^gf_mul(0x09,tmp[3][i]);

    state[1][i]=gf_mul(0x09,tmp[0][i])^gf_mul(0x0e,tmp[1][i])^gf_mul(0x0
b,tmp[2][i])^gf_mul(0x0d,tmp[3][i]);
```

```
    state[2][i]=gf_mul(0x0d,tmp[0][i])^gf_mul(0x09,tmp[1][i])^gf_mul(0x0
e,tmp[2][i])^gf_mul(0x0b,tmp[3][i]);

    state[3][i]=gf_mul(0x0b,tmp[0][i])^gf_mul(0x0d,tmp[1][i])^gf_mul(0x0
9,tmp[2][i])^gf_mul(0x0e,tmp[3][i]);
                }
            }
            /*
            cout<<"after invmixcolumns:"<<endl;
            for(int i=0;i<4;++i){
                for(int       j=0;j<4;++j)       cout<<i<<"       "<<j<<"
"<<hex<<(int)state[i][j]<<" ";
                cout<<endl;
            }
            */
        }
        for(int i=0;i<4;++i){
            for(int j=0;j<4;++j){
                out[j*4+i]=state[i][j];
            }
        }
    }

    int main(){
        #ifdef _WIN32
        setmode(fileno(stdin), O_BINARY);
        setmode(fileno(stdout), O_BINARY);
        #endif
        uint8_t mode,key[16],IV[16],tmp[50];
        uint32_t len;
        fread(&tmp,sizeof(uint8_t),33,fp);
        mode=tmp[0];
        for(int i=1;i<=16;++i){
            key[i-1]=tmp[i];
```

```
            IV[i-1]=tmp[i+16];
        }
        fread(&len,sizeof(uint32_t),1,fp);
        int pad_len=16-len%16;//need to paddle pad_len bytes
        //fwrite(&text,sizeof(uint8_t),l,p);
        uint32_t w[44];
        uint8_t text[20];
        key_expend(key,w);
        if(mode==0x01){
            uint32_t l=len+pad_len;
            uint8_t block[16],out[16];
            for(int i=0;i<l/16;++i){
                fread(&text,sizeof(uint8_t),16,fp);
                if(i==l/16-1){
                    for(int j=0;j<pad_len;++j) text[len%16+j]=pad_len;
                }
                for(int j=0;j<16;++j){
                    block[j]=text[j]^((i==0)?IV[j]:out[j]);
                }
                en_AES(block,w,out);
                fwrite(&out,sizeof(uint8_t),16,p);
            }
        }
        else if(mode==0x81){
            uint8_t block[16],out[16],tmp[16];
            for(int i=0;i<len/16;++i){
                fread(&text,sizeof(uint8_t),16,fp);
                for(int j=0;j<16;++j){
                    block[j]=text[j];
                }
                de_AES(block,w,out);
                for(int j=0;j<16;++j){
                    out[j]^=((i==0)?IV[j]:tmp[j]);
                    tmp[j]=block[j];
```

```
            }
            if(i==len/16-1){
                for(int j=0;j<16;++j){
                    if(out[j]==16-j){
                        int fl=1;
                        for(int k=j;k<16;++k){
                            if(out[k]!=out[j]){
                                fl=0;
                                break;
                            }
                        }
                        if(fl&&j) fwrite(&out,sizeof(uint8_t),j,p);
                    }
                }
            }
            else fwrite(&out,sizeof(uint8_t),16,p);
        }
    }
}
```

以下是使用了指令集实现的 AES-128：

```
#include<iostream>
#include<cstdint>
#include<fstream>
#include<fcntl.h>
#include<wmmintrin.h>
#include<immintrin.h>
#include<emmintrin.h>
using namespace std;
/*
FILE *fp=fopen("dump1.bin","rb");
FILE *p=fopen("output.bin","wb");
*/

#define fp stdin
```

```
   #define p stdout


   //const                                                    uint8_t
rcon[11]={0,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36};
   __inline void key_expend(uint8_t key[],__m128i w[],__m128i dw[]){
       __m128i                tmp=_mm_loadu_si128(reinterpret_cast<const
__m128i*>(key)),t,t2;
       w[0]=tmp;
       t=_mm_slli_si128(tmp,4);
       t^=tmp;
       t2=_mm_slli_si128(t,8);
       t^=t2;
       tmp=_mm_aeskeygenassist_si128(tmp,0x01);
       tmp=_mm_shuffle_epi32(tmp,0xFF);
       tmp^=t;
       w[1]=tmp;
       t=_mm_slli_si128(tmp,4);
       t^=tmp;
       t2=_mm_slli_si128(t,8);
       t^=t2;
       tmp=_mm_aeskeygenassist_si128(tmp,0x02);
       tmp=_mm_shuffle_epi32(tmp,0xFF);
       tmp^=t;
       w[2]=tmp;
       t=_mm_slli_si128(tmp,4);
       t^=tmp;
       t2=_mm_slli_si128(t,8);
       t^=t2;
       tmp=_mm_aeskeygenassist_si128(tmp,0x04);
       tmp=_mm_shuffle_epi32(tmp,0xFF);
       tmp^=t;
       w[3]=tmp;
       t=_mm_slli_si128(tmp,4);
       t^=tmp;
```

```
t2=_mm_slli_si128(t,8);
t^=t2;
tmp=_mm_aeskeygenassist_si128(tmp,0x08);
tmp=_mm_shuffle_epi32(tmp,0xFF);
tmp^=t;
w[4]=tmp;
t=_mm_slli_si128(tmp,4);
t^=tmp;
t2=_mm_slli_si128(t,8);
t^=t2;
tmp=_mm_aeskeygenassist_si128(tmp,0x10);
tmp=_mm_shuffle_epi32(tmp,0xFF);
tmp^=t;
w[5]=tmp;
t=_mm_slli_si128(tmp,4);
t^=tmp;
t2=_mm_slli_si128(t,8);
t^=t2;
tmp=_mm_aeskeygenassist_si128(tmp,0x20);
tmp=_mm_shuffle_epi32(tmp,0xFF);
tmp^=t;
w[6]=tmp;
t=_mm_slli_si128(tmp,4);
t^=tmp;
t2=_mm_slli_si128(t,8);
t^=t2;
tmp=_mm_aeskeygenassist_si128(tmp,0x40);
tmp=_mm_shuffle_epi32(tmp,0xFF);
tmp^=t;
w[7]=tmp;
t=_mm_slli_si128(tmp,4);
t^=tmp;
t2=_mm_slli_si128(t,8);
t^=t2;
```

```
        tmp=_mm_aeskeygenassist_si128(tmp,0x80);
        tmp=_mm_shuffle_epi32(tmp,0xFF);
        tmp^=t;
        w[8]=tmp;
        t=_mm_slli_si128(tmp,4);
        t^=tmp;
        t2=_mm_slli_si128(t,8);
        t^=t2;
        tmp=_mm_aeskeygenassist_si128(tmp,0x1b);
        tmp=_mm_shuffle_epi32(tmp,0xFF);
        tmp^=t;
        w[9]=tmp;
        t=_mm_slli_si128(tmp,4);
        t^=tmp;
        t2=_mm_slli_si128(t,8);
        t^=t2;
        tmp=_mm_aeskeygenassist_si128(tmp,0x36);
        tmp=_mm_shuffle_epi32(tmp,0xFF);
        tmp^=t;
        w[10]=tmp;
        dw[0]=w[0];
        for(int i=1;i<=9;++i) dw[i]=_mm_aesimc_si128(w[i]);
        dw[10]=w[10];
    }
    __inline void en_AES(uint8_t text[],__m128i w[],uint8_t out[]){
        __m128i             state=_mm_loadu_si128(reinterpret_cast<const
__m128i*>(text));
        state^=w[0];
        for(int i=1;i<10;++i) state=_mm_aesenc_si128(state,w[i]);
        state=_mm_aesenclast_si128(state,w[10]);
        _mm_storeu_si128(reinterpret_cast<__m128i*>(out),state);
    }
    __inline void de_AES(uint8_t text[],__m128i dw[],uint8_t out[]){
        __m128i             state=_mm_loadu_si128(reinterpret_cast<const
```

```
__m128i*>(text));
        state^=dw[10];
        for(int i=9;i>=1;i--) state=_mm_aesdec_si128(state,dw[i]);
        state=_mm_aesdeclast_si128(state,dw[0]);
        _mm_storeu_si128(reinterpret_cast<__m128i*>(out),state);
    }

    int main(){
        #ifdef _WIN32
        setmode(fileno(stdin), O_BINARY);
        setmode(fileno(stdout), O_BINARY);
        #endif
        uint8_t mode,key[16],IV[16],tmp[50];
        uint32_t len;
        fread(&tmp,sizeof(uint8_t),33,fp);
        mode=tmp[0];
        for(int i=1;i<=16;++i){
            key[i-1]=tmp[i];
            IV[i-1]=tmp[i+16];
        }
        fread(&len,sizeof(uint32_t),1,fp);
        int pad_len=16-len%16;//need to paddle pad_len bytes
        //fwrite(&text,sizeof(uint8_t),l,p);
        __m128i w[12],dw[12];
        uint8_t text[20];
        key_expend(key,w,dw);
        if(mode==0x01){
            uint32_t l=len+pad_len;
            uint8_t block[16],out[16];
            for(int i=0;i<l/16-1;++i){
                fread(&text,sizeof(uint8_t),16,fp);
                for(int j=0;j<16;++j){
                    block[j]=text[j]^((i==0)?IV[j]:out[j]);
                }
```

```
            en_AES(block,w,out);
            fwrite(&out,sizeof(uint8_t),16,p);
        }
        fread(&text,sizeof(uint8_t),16,fp);
        for(int j=0;j<pad_len;++j) text[len%16+j]=pad_len;
        for(int j=0;j<16;++j) block[j]=text[j]^out[j];
        en_AES(block,w,out);
        fwrite(&out,sizeof(uint8_t),16,p);
    }
    else if(mode==0x81){
        uint8_t block[16],out[16],tmp[16];
        for(int i=0;i<len/16-1;++i){
            fread(&text,sizeof(uint8_t),16,fp);
            for(int j=0;j<16;++j){
                block[j]=text[j];
            }
            de_AES(block,dw,out);
            for(int j=0;j<16;++j){
                out[j]^=((i==0)?IV[j]:tmp[j]);
                tmp[j]=block[j];
            }
            fwrite(&out,sizeof(uint8_t),16,p);
        }
        fread(&text,sizeof(uint8_t),16,fp);
        for(int j=0;j<16;++j) block[j]=text[j];
        de_AES(block,dw,out);
        for(int j=0;j<16;++j){
            out[j]^=tmp[j];
            tmp[j]=block[j];
        }
        for(int j=0;j<16;++j){
            if(out[j]==16-j){
                int fl=1;
                for(int k=j;k<16;++k){
```

```
                    if(out[k]!=out[j]){
                        fl=0;
                        break;
                    }
                }
                if(fl&&j) fwrite(&out,sizeof(uint8_t),j,p);
            }
        }
    }
}
```

五、实验结果

此处以最后一个样例的解密为例：

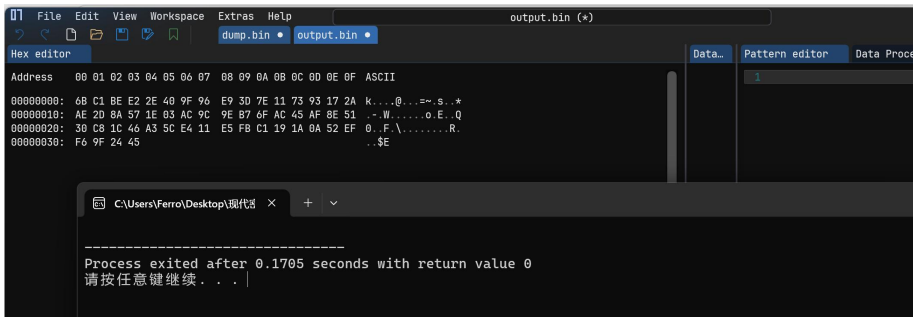


图 2 解密结果

图 2 是样例 6 在第一个代码下的测试结果，可以看到正确输出了非 16 字节对齐的明文。

在 0J 上评测如下（编号 1368）：

图 3 非优化的 AES-128 评测结果

对于用 AES-NI 指令集，用样例#5，本地测试得到结果如下：

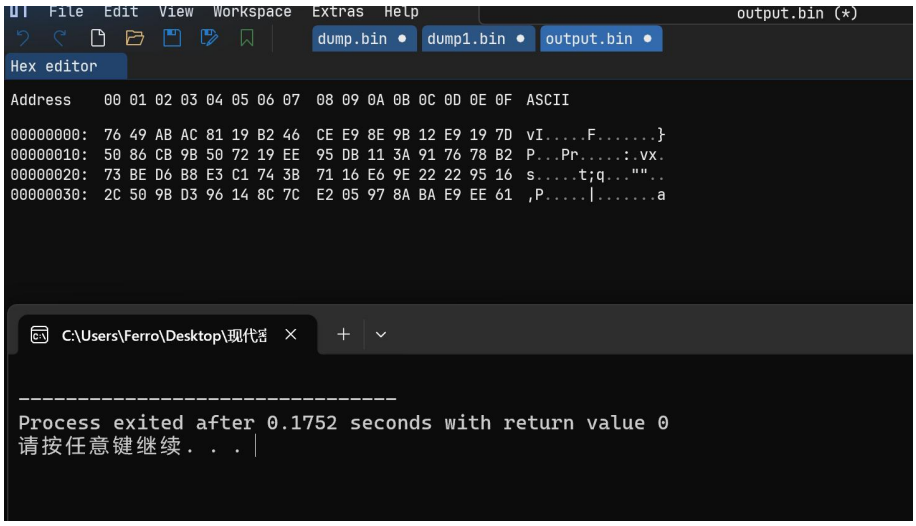


图 4 加密结果

可以看到正确加密了非对齐的明文，且输出是 16 字节对齐的。

在 OJ 上评测如下（编号 1450）：

图 5 指令集优化的 AES-128 评测结果

可以看到指令集优化起到了很大的加速作用。

六、思考题

1. 除了课堂上介绍的五种模式以外，分组加密还有一些其他的工作模式。考虑下面的两个场景，使用什么工作模式可以满足它们的需求？这个工作模式的原理是什么？

1.1 你需要检查密文是否被篡改（不需要单独使用 Hash 函数）。

在 AES-128 下可以采用 CCM 模式。CCM（Counter with Cipher Block Chaining-Message Authentication Code）是 CBC-MAC 与 CTR 的组合，可同时进行数据加密及认证，它基于对称秘钥分组加密算法，分组大小 128bits。

该工作模式的原理：

CCM 采用独立进行加密认证(A + E)的方式，采用两个密钥，加密消息得到 $C=E(K_2,M)$，

然后对明文计算 MAC 值，T=MAC(K_1,M)，最后通过(T,C)来进行认证。

输入：明文消息、需要被认证，但是不需要被加密的数据、随机变量 N（用于防范重放攻击）

算法主流程：

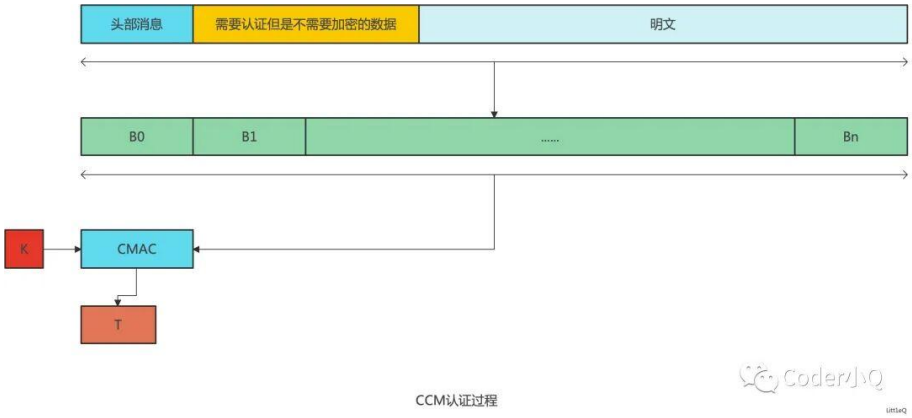


图 6 CCM 模式的先行认证-分块

1） 首先对这些输入进行分块。分为 B0，B1，……，Bn

2） 对于分组之后的消息计算 CMAC，最终 MAC 输出的长度根据前面提到过的方法得到 T = CMAC(B0 || B1 ... || Bn)，至此完成认证。

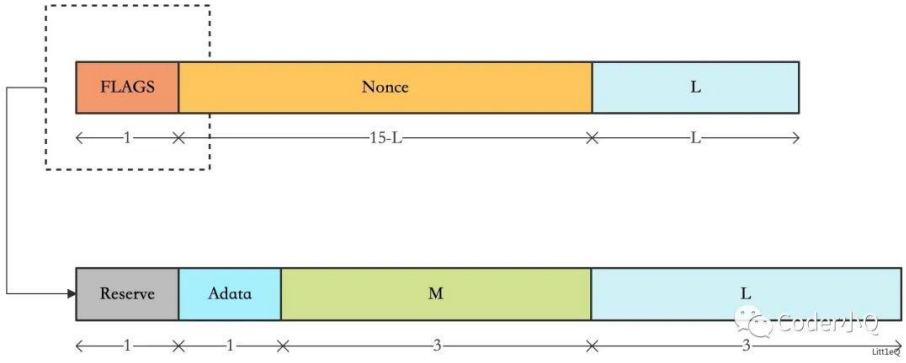


图 7 第一个分块（头部消息），相当于 Flag， 第一位保留，第二位标识有没有额外的不需要加密的数据，后三个字节是 MAC 的长度

3） 再用 CBC 工作模式进行加密。

判断密文有没有被篡改应在解密认证中进行：

1） 如果 CLen < Tlen，这说明出现了不合理的情况，返回认证失败，认为密文被

篡改。

2） 执行计数器生成函数，初始计数器和上文保持一致得到 CtrO ,...,Ctrn

3） 然后对消息进行 CTR 模式的解密，得到明文，到这里解密阶段完成，这里最后一个分组解密之后即为消息认证码，这个分组需要和 CtrO 加密之后的结果异或得到 T。

4） 按照上面加密过程重新拼接成需要认证的消息，同样得到 BO,...,Bn ， 同样计算 CMAC 得到 T_{verify}。

5） 若 $T_{verify} \neq T$，认为密文被篡改，认证失败；否则认为密文未被篡改。

至此，我们在没有单独引入 Hash 函数的情况下，利用 CBC-MAC 和 CTR 模式的组合，完成了密文是否被篡改的检查。

1.2 你需要加密一块磁盘。磁盘上的数据是以扇区（通常为 512 Bytes）为单位读写的，操作系统可以通过扇区号随机访问扇区。磁盘加密的密文长度必须要和明文相同，不能使用额外的空间。

这里可以采用 XTS 模式。对于磁盘加密来说，数据的存储按一定的格式，比如不同扇区。在加密的过程当中，也不希望为 IV 之类的东西开销单独的空间。为此，我们引入 2002 年 Moses Liskov，Ronald L.Rivest, David Wagner,首次提出的可调整的分组密码这个概念，跟传统的分组密码相比，除了密匙和明文这两个输入外，还引入另外一个输入 tweak，即调整值。引入可调整值之后，我们就可以改变 tweak 值，来改变加密之后的密文。

XTS 模式有两个密钥,其中一个用于执行 AES 分组加密,另一个用于加密调整值(tweak),这种加密调整借助有限域和异或运算，使得每次即使是相同的分组也不会得到相同的密文，确保安全性。

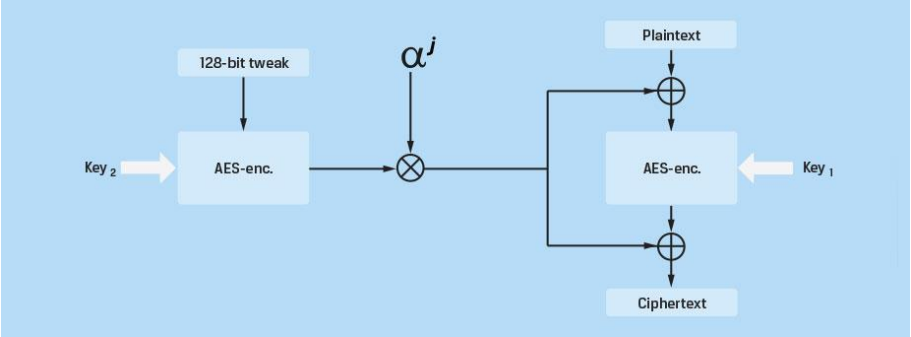


图 8 XTS 工作模式下的块工作示意

加密算法原理如下：

1）基于扇区的地址生成 tweak，通常使用一个简单的函数（如加密或多项式乘法）将扇区地址转换为 128 位的值。

2）tweak 先用密钥 2 通过 AES 加密。结果进行一个有限域数 a^j 的乘运算，得到 T。

3）T 跟明文块 P 异或得到 PP，PP 再使用 K1 进行 AES 加密得到 CC，CC 再和 1 中的 T 异或得到最终的密文 C。

2. 其他的问题？

为什么自己写的 AES-128 和基于 AES-NI 的运行速度差异很大？

Wiki 中提到相比没有加速的 Pentium 4, AES/GCM 的吞吐量从大约每字节 28.0 周期显著提升至每字节 3.5 周期。大概是 8 倍的加速。也就是说预期时间应该是 77ms*8=616ms。实际上运行了四千多 ms，说明存在可改进的地方。分析原因如下：

1）乘法的运行时间还是偏慢。相比于加密过程，每一次解密过程多花费了 8*9=72 次有限域的乘法运算，这是比较耗时的。在指令集中可能采用了模乘-移位及不进位乘法的方法加速这一过程。

2）轮密钥用 uint32_t 表示不合理。这样导致了每一次轮密钥相加都要进行很繁琐的移位截取操作，以及组合操作。其实可以用 uint8_t 表示。

七、实验总结

本次实验主要考察分组密码，包含二进制流文件读/写，SPN 网络的基本结构，分组密码的工作模式，矩阵实现及函数复用。同时，利用 CBC 工作模式完成了 AES-128 的加密和解密，通过指令集 AES-NI 的使用实现了大约 50 倍的加速。该实验旨在熟悉 AES 标准，熟悉分组密码的多种工作模式并能熟练应用。

在算法设计上，采用 FIPS-197 标准的 AES-128 加密过程，根据指令集的加速所得时间也较为理想。美中不足的是自己写的程序效率较慢，需要进一步的优化。

参考资料：

1. Coder 小 Q，【密码学】一文读懂 CCM，
https://mp.weixin.qq.com/s/-hNRN2LCHfo2Q7f_6UM4gQ

2. AES-XTS Block Cipher Mode is used in Kingston IronKey Hardware-Encrypted USB and External SSD Drives，
https://www.kingston.com.cn/en/blog/data-security/xts-encryption

3. AES instruction set - Wikipedia，
https://en.wikipedia.org/wiki/AES_instruction_set