



密码工程实验报告

实验名称 AES 算法的实现

实验日期 2022 年 12 月 20 日

目录

1	AES 加密实现	2
1.1	密钥扩展算法	2
1.2	查表优化	3
1.2.1	有限域运算实现	4
1.2.2	查找表构建	5
1.3	加密运算	5
2	AES 解密实现	8
2.1	查找表构建	9
2.2	解密密钥构建	10
2.3	解密运算	11
3	结果验证	13
4	实验心得	15

1 AES 加密实现

AES 是以 SPN 结构为基础设计的算法，这里以 128bit 的 AES 为例。

AES 的操作是在一个 4×4 的 State 矩阵上进行的，首先要把 128bit 的明文解析进 State 矩阵。(按 column 竖向排列),AES 将明文转化为密文通过不断迭代更新 state 进行

每次轮函数过程包含四步

AddRoundKey: State 中的每一个字节和轮密钥中的对应字节做异或运算

SubBytes: State 中的每一个字节按一定规则用 S 盒中的元素进行替换

ShiftRows: State 中第 i 行的元素左移 $i-1$ 个字节

MixColumns: State 每一列的四个字节被混合通过可逆的线性运算

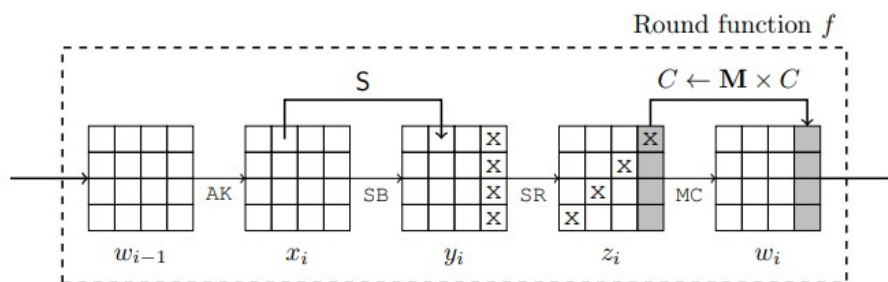


图 1: Round Function

1.1 密钥扩展算法

AES128 中的原始密钥 Key 为 16 个字节，运算中包含原始密钥需要 11 个 State 矩阵大小的密钥，每一列所包含 32 位记为一个 `uint32_t W`，所以密钥扩展一共需要产生 44 个列 `W`，即 `uint32_t W[44]`。

$$W[n] = \begin{cases} W[n-4] \oplus W[n-1], & \text{if } n \neq 4 \text{ 的倍数.} \\ W[n-4] \oplus \text{Mix}(W[n-1]) \oplus \text{rcon}[(n/4) - 1], & \text{if } n == 4 \text{ 的倍数.} \end{cases}$$

其中 $Mix(x) = SubWord(RotWord(x))$, $RotWord()$ 为循环左移一位, $SubWord()$ 为字节替换, $rcon$ 为轮常量异或
具体实现如下

```

1 void KeyExtend() {
2     uint32_t Rcon[10] = { 0x01000000, 0x02000000, 0x04000000, 0x08000000
3         , 0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x1B000000, 0
4         x36000000 };
5     uint8_t *temp;
6     for (int i = 4; i < 44; i++) {
7         if (i % 4 == 0) {
8             Key[i] = shift_left(Key[i - 1], 8);
9             temp = (uint8_t*)&Key[i];
10            for (int i = 0; i < 4; i++)
11                temp[i] = S_replace(temp[i]);
12            Key[i] = Key[i] ^ Key[i - 4] ^ Rcon[i / 4 - 1];
13        }
14        else
15            Key[i] = Key[i - 4] ^ Key[i - 1];
16    }
17 }

```

1.2 查表优化

这里可以将 Mixcolumn 应用 look-up table 进行优化, 在 8bit 处理机上, 线性运算 $2x$ 和 $3x$ 可以进行查表, 需要内存大小为 $2 \times 2^8 \text{Byte} = 0.5KB$. 32 位处理机上可以对一系列 Mixcolumn 简化成 4 次 32 位查找表运算

这里可以实现 8 到 32 位的转化 $\mathbb{F}_{2^8} \rightarrow (\mathbb{F}_{2^8}, \mathbb{F}_{2^8}, \mathbb{F}_{2^8}, \mathbb{F}_{2^8})$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 2 \cdot s_{0,c} \\ 1 \cdot s_{0,c} \\ 1 \cdot s_{0,c} \\ 3 \cdot s_{0,c} \end{bmatrix} + \begin{bmatrix} 3 \cdot s_{1,c} \\ 2 \cdot s_{1,c} \\ 1 \cdot s_{1,c} \\ 1 \cdot s_{1,c} \end{bmatrix} + \begin{bmatrix} 1 \cdot s_{2,c} \\ 3 \cdot s_{2,c} \\ 2 \cdot s_{2,c} \\ 1 \cdot s_{2,c} \end{bmatrix} + \begin{bmatrix} 1 \cdot s_{3,c} \\ 1 \cdot s_{3,c} \\ 3 \cdot s_{3,c} \\ 2 \cdot s_{3,c} \end{bmatrix}$$

图 2: Mixcolumn Operation

这样我们得到 Mixcolumn 的结果就可以通过查找表进行, 需要存储大小为 $4 \times 8\text{Bytes} = 1\text{KB}$, 同时我们可以进一步对查表进行优化, 可以将 Subbytes 和 mixcolumn 进行结合 (SubBytes() 和 Mixcolumn() 交换顺序不影响结果), 这样我们可以将 L 运算结合 SubBytes.

$$L(x) = (2 \cdot S(x), S(x), S(x), 3 \cdot S(x)) \text{ for } x \in \mathbb{F}_2^8$$

1.2.1 有限域运算实现

实现 L 运算包括乘 02 和乘 03, 乘 02 在有限域中实现就是乘 x , 这里可以通过左移一位实现, 其中要对结果的最高位进行判断, 如果为 1 就要进行模运算, 这里可以通过异或模多项式进行实现, 乘 03 的情况可在乘 02 基础上再异或 x 实现.

```

1 uint8_t xtime(uint8_t x) {
2     //00011011
3     uint8_t temp = x << 1;
4     return ((x >> 7) ^ 0x01) ? temp : temp ^ 27;
5 }
6
7 uint8_t xtime2(uint8_t x)
8 {
9     return (xtime(x))^x;
10 }

```

1.2.2 查找表构建

在实现 L 线性运算时, 先将 x 进行 SubByte 替换, 然后将 x 分别与相应的系数进行相乘, 由于最后数据的表示为 uint32, 在高位的 x 要通过移位运算移到相应的位置.

```
1 uint32_t L(uint8_t x) {
2     uint32_t res;
3     uint32_t res1 = xtime2(S_replace(x));
4     uint32_t res2 = S_replace(x) << 8;
5     uint32_t res3 = S_replace(x) << 16;
6     uint32_t res4 = xtime(S_replace(x)) << 24;
7     res = res1 ^ res2 ^ res3 ^ res4;
8     return res;
9 }
10 \\L operation
11 void generateEnTable() {
12     for (int i = 0; i <= 0xFF; i++) {
13         Table[i] = L(i);
14     }
15 }
16 \\attention:x use the int or uint32
```

这里要注意 x 应该用 int 类型, 如果用 uint8 类型, 将陷入死循环

1.3 加密运算

这里对加密运算关键位置实现进行解释, 这里将 ShiftRow 和 look-up table 结合. 首先进行 shiftRow, 在执行 ShiftRow 以后各列的乘法元进行了改变, 要根据变化以后的顺序进行查表运算, 其中查表时应注意线性运算的顺序也发生了变化 (L 运算是按 02 03 01 01 进行), 要进行相应的循环右移操作.

0	1	2	3
1	2	3	0
2	3	0	1
3	0	1	2

表 1: ShiftRows

first column:0 1 2 3

second column:1 2 3 0

third column:2 3 0 1

forth column:3 0 1 2

```

1 void Encrypt() {
2     //首先将in装入state
3     for (int i = 0; i < 4; i++) {
4         State[i] = in[i];
5     }
6     //生成查找表，这个不算在时间计算之中
7     generateEnTable();
8     //异或密钥，uint32
9     AddRoundKey(0);
10    //密钥扩展
11    KeyExtend();
12    for (int i = 1; i < 10; i++) {
13        //shiftRow+Subbyte+Mixcol
14        /*
15        shiftRow:
16        1 2 3 4
17        2 3 4 1
18        3 4 1 2
19        4 1 2 3
20        first column = 1234
21        second column = 2341
22        third column = 3412

```

```

23      forth column = 4123
24
25      */
26      uint32_t temp[4];
27      for (int k = 0; k < 4; k++)
28          temp[k] = State[k];
29      //
30      State[0] = Table[temp[0] >> 24 & 0xFF];
31      State[0] ^= shift_right(Table[temp[1] >> 16 & 0xFF], 8);
32      State[0] ^= shift_right(Table[temp[2] >> 8 & 0xFF], 16);
33      State[0] ^= shift_right(Table[temp[3] & 0xFF], 24);
34      //
35      State[1] = Table[temp[1] >> 24 & 0xFF];
36      State[1] ^= shift_right(Table[temp[2] >> 16 & 0xFF], 8);
37      State[1] ^= shift_right(Table[temp[3] >> 8 & 0xFF], 16);
38      State[1] ^= shift_right(Table[temp[0] & 0xFF], 24);
39      //
40      State[2] = Table[temp[2] >> 24 & 0xFF];
41      State[2] ^= shift_right(Table[temp[3] >> 16 & 0xFF], 8);
42      State[2] ^= shift_right(Table[temp[0] >> 8 & 0xFF], 16);
43      State[2] ^= shift_right(Table[temp[1] & 0xFF], 24);
44      //
45      State[3] = Table[temp[3] >> 24 & 0xFF];
46      State[3] ^= shift_right(Table[temp[0] >> 16 & 0xFF], 8);
47      State[3] ^= shift_right(Table[temp[1] >> 8 & 0xFF], 16);
48      State[3] ^= shift_right(Table[temp[2] & 0xFF], 24);
49
50
51      AddRoundKey(i);
52      //for (int j = 0; j < 4; j++) {
53      //      printf("%02x", State[j]);
54      //      cout << endl;
55      //}
56  }
57      uint32_t temp[4];

```



```

58     for (int k = 0; k < 4; k++)
59         temp[k] = State[k];
60     //shift_row+Subbytes
61     State[0] = S_replace((temp[0] >> 24) & 0xFF) << 24;
62     State[0] |= S_replace((temp[1] >> 16) & 0xFF) << 16;
63     State[0] |= S_replace((temp[2] >> 8) & 0xFF) << 8;
64     State[0] |= S_replace((temp[3]) & 0xFF);
65     //
66     State[1] = S_replace((temp[1] >> 24) & 0xFF) << 24;
67     State[1] |= S_replace((temp[2] >> 16) & 0xFF) << 16;
68     State[1] |= S_replace((temp[3] >> 8) & 0xFF) << 8;
69     State[1] |= S_replace((temp[0]) & 0xFF);
70     //
71     State[2] = S_replace((temp[2] >> 24) & 0xFF) << 24;
72     State[2] |= S_replace((temp[3] >> 16) & 0xFF) << 16;
73     State[2] |= S_replace((temp[0] >> 8) & 0xFF) << 8;
74     State[2] |= S_replace((temp[1]) & 0xFF);
75     //
76     State[3] = S_replace((temp[3] >> 24) & 0xFF) << 24;
77     State[3] |= S_replace((temp[0] >> 16) & 0xFF) << 16;
78     State[3] |= S_replace((temp[1] >> 8) & 0xFF) << 8;
79     State[3] |= S_replace((temp[2]) & 0xFF);
80     AddRoundKey(10);
81     for (int i = 0; i < 4; i++)
82         out[i] = State[i];
83 }

```

2 AES 解密实现

对于 AES 算法, 可将加密变换进行逆转得到解密算法, 同时轮密钥逆序进行使用.

```

InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

    for round = Nr-1 step -1 downto 1
        InvShiftRows(state)                // See Sec. 5.3.1
        InvSubBytes(state)                 // See Sec. 5.3.2
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        InvMixColumns(state)               // See Sec. 5.3.3
    end for

    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[0, Nb-1])

    out = state
end

```

图 3: Pseudo Code for the Inverse Cipher

2.1 查找表构建

对于 AES 的解密算法, 同样可以用查表进行优化, 这里和加密时相同, `InvSubBytes()` 和 `InvShiftRows()` 可以进行交换, 我们将 `InvSubBytes()` 和 `InvMixColumn()` 共同构建解密查找表. `InvSubBytes()` 直接可以用 S 盒的逆进行构建, `InvMixColumn()` 则需要进行 `MixColumn()` 的逆变换, 这里需要乘逆变换矩阵.

$s'(x) = a^{-1}(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \leq c < Nb.$$

图 4: InvMixColumn

逆变换矩阵中需要对多项式乘法进行改进, 我们在乘 x 的基础上增加移位和位判断, 扩展多项式乘法.

```
1 uint8_t Mult(uint8_t x, uint8_t y) {
2     return (((y & 1) * x) ^ ((y >> 1 & 1) * xtime(x)) ^ ((y >> 2 & 1)
      * xtime(xtime(x))) ^ ((y >> 3 & 1) * xtime(xtime(xtime(x)))) ^
      ((y >> 4 & 1) * xtime(xtime(xtime(xtime(x))))));
3 }
```

按照加密的逻辑, 构造一个 8bit 输入 32bit 输出的查找表, 将遍历的结果存入 Table2 中.

2.2 解密密钥构建

列混合运算 MixColumns() 和 InvMixColumns() 是关于列输入线性变换, 这意味着

$$\text{InvMixcolumns}(\text{state XOR Round Key}) = \text{InvMixcolumns}(\text{state}) \oplus \text{InvMixcolumns}(\text{Round Key})$$

因此对于解密运算, 要对 1 到 Nr-1 轮的解密密钥进行编排处理, 对这些轮密钥进行逆列混合运算. 这里也对轮密钥的处理进行查表优化, 由于查找表是 InvSubBytes()+InvMixcolumns(), 这里在查找表之前可以再做一步 SubBytes() 以确保查表输出只进行了 InvMixcolumns().

```
1 void Keyinv_col() {
2     uint32_t temp[44];
3     for (int i = 0; i < 44; i++)
```

```

4         temp[i] = Key[i];
5     for (int i = 4; i < 40; i++) {
6         Key[i] = Table2[S_replace(temp[i] >> 24 & 0xFF)];
7         Key[i]^= shift_right(Table2[S_replace(temp[i] >> 16 & 0xFF
8             )],8);
9         Key[i] ^= shift_right(Table2[S_replace(temp[i] >> 8 & 0xFF
10            )], 16);
11        Key[i] ^= shift_right(Table2[S_replace(temp[i] >> 0 & 0xFF
12            )], 24);
13    }
14 }

```

2.3 解密运算

这里解密运算和加密运算逻辑相同, 密钥要进行反序输入, 采取由最后一轮循环递减迭代的方式来保证轮密钥逆序. InvShiftRows() 运算采取循环右移, 与加密过程相反.

```

1 void Decrypt() {
2     generateDeTable();
3     for (int i = 0; i < 4; i++)
4         State[i] = out[i];
5     //将输出内容拷贝到State
6     //这时候要反序使用轮密钥
7     AddRoundKey(10);
8     //for (int j = 0; j < 4; j++) {
9     //     printf("%02x", State[j]);
10    //     cout << endl;
11    //}
12    Keyinv_col();
13    for (int i = 0; i < 44; i++) {
14        printf("%02x\n", Key[i]);
15    }
16    }
17    for (int i = 9; i > 0; i--) {
18        //inverse shiftRows+InvSubbytes+InvMixcols

```

```

19      /*Inv_shiftRows
20      0 1 2 3
21      3 0 1 2
22      2 3 0 1
23      1 2 3 0
24
25      */
26      uint32_t temp[4];
27      for (int k = 0; k < 4; k++)
28          temp[k] = State[k];
29      //将state复制到temp
30      State[0] = Table2[temp[0] >> 24 & 0xFF];
31      State[0] ^= shift_right(Table2[temp[3] >> 16 & 0xFF], 8);
32      State[0] ^= shift_right(Table2[temp[2] >> 8 & 0xFF], 16);
33      State[0] ^= shift_right(Table2[temp[1] & 0xFF], 24);
34      //
35      State[1] = Table2[temp[1] >> 24 & 0xFF];
36      State[1] ^= shift_right(Table2[temp[0] >> 16 & 0xFF], 8);
37      State[1] ^= shift_right(Table2[temp[3] >> 8 & 0xFF], 16);
38      State[1] ^= shift_right(Table2[temp[2] & 0xFF], 24);
39      //
40      State[2] = Table2[temp[2] >> 24 & 0xFF];
41      State[2] ^= shift_right(Table2[temp[1] >> 16 & 0xFF], 8);
42      State[2] ^= shift_right(Table2[temp[0] >> 8 & 0xFF], 16);
43      State[2] ^= shift_right(Table2[temp[3] & 0xFF], 24);
44      //
45      State[3] = Table2[temp[3] >> 24 & 0xFF];
46      State[3] ^= shift_right(Table2[temp[2] >> 16 & 0xFF], 8);
47      State[3] ^= shift_right(Table2[temp[1] >> 8 & 0xFF], 16);
48      State[3] ^= shift_right(Table2[temp[0] & 0xFF], 24);
49
50
51      AddRoundKey(i);
52
53      }

```

```

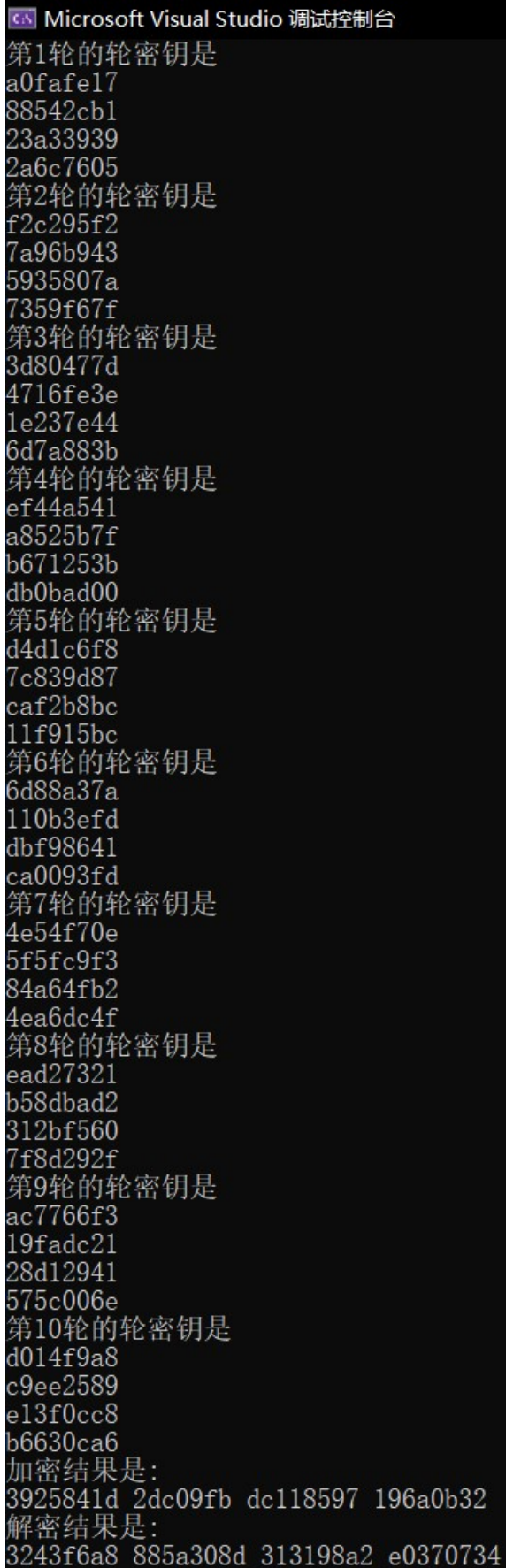
54     uint32_t temp[4];
55     for (int k = 0; k < 4; k++)
56         temp[k] = State[k];
57     State[0] = Siv_replace((temp[0] >> 24) & 0xFF) << 24;
58     State[0] |= Siv_replace((temp[3] >> 16) & 0xFF) << 16;
59     State[0] |= Siv_replace((temp[2] >> 8) & 0xFF) << 8;
60     State[0] |= Siv_replace((temp[1]) & 0xFF);
61     //
62     State[1] = Siv_replace((temp[1] >> 24) & 0xFF) << 24;
63     State[1] |= Siv_replace((temp[0] >> 16) & 0xFF) << 16;
64     State[1] |= Siv_replace((temp[3] >> 8) & 0xFF) << 8;
65     State[1] |= Siv_replace((temp[2]) & 0xFF);
66     //
67     State[2] = Siv_replace((temp[2] >> 24) & 0xFF) << 24;
68     State[2] |= Siv_replace((temp[1] >> 16) & 0xFF) << 16;
69     State[2] |= Siv_replace((temp[0] >> 8) & 0xFF) << 8;
70     State[2] |= Siv_replace((temp[3]) & 0xFF);
71     //
72     State[3] = Siv_replace((temp[3] >> 24) & 0xFF) << 24;
73     State[3] |= Siv_replace((temp[2] >> 16) & 0xFF) << 16;
74     State[3] |= Siv_replace((temp[1] >> 8) & 0xFF) << 8;
75     State[3] |= Siv_replace((temp[0]) & 0xFF);
76     AddRoundKey(0);
77     for (int i = 0; i < 4; i++)
78         in[i] = State[i];
79     //system("pause");
80 }

```

3 结果验证

我们按照 IST.FIPS.197 上给出的附录 B 示例进行验证, 可以看到解密结果和 input 一致.

Input= 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34



```
Microsoft Visual Studio 调试控制台
第1轮的轮密钥是
a0fafe17
88542cb1
23a33939
2a6c7605
第2轮的轮密钥是
f2c295f2
7a96b943
5935807a
7359f67f
第3轮的轮密钥是
3d80477d
4716fe3e
1e237e44
6d7a883b
第4轮的轮密钥是
ef44a541
a8525b7f
b671253b
db0bad00
第5轮的轮密钥是
d4dlc6f8
7c839d87
caf2b8bc
11f915bc
第6轮的轮密钥是
6d88a37a
110b3efd
dbf98641
ca0093fd
第7轮的轮密钥是
4e54f70e
5f5fc9f3
84a64fb2
4ea6dc4f
第8轮的轮密钥是
ead27321
b58dbad2
312bf560
7f8d292f
第9轮的轮密钥是
ac7766f3
19fadc21
28d12941
575c006e
第10轮的轮密钥是
d014f9a8
c9ee2589
e13f0cc8
b6630ca6
加密结果是:
3925841d 2dc09fb dc118597 196a0b32
解密结果是:
3243f6a8 885a308d 313198a2 e0370734
```

图 5: Verify

4 实验心得

实验过程中进一步理解的 AES 的设计与实现, 在软件实现的过程中, 通过移位和异或来进行运算提升效率, 同时使用查找表将字节代换和列混合运算进行合并, 同时在代码中将查找表和行移位运算结合, 进一步增加了效率. 同时还实现了解密运算, 实现逆行移位运算. 在多项式运算上, 将传统的乘除运算变化为异或和移位, 通过掩码进行模运算, 这些实现上的技巧让我收获很大.

参考文献

[1] IST.FIPS.197

[2] software_aes_implementation