SHANDONG UNIVERSITY

密码工程第一次实验报告

姓名: 谢子洋
学院: 网络空间安全学院 (研究院)
专业: 网络空间安全
学号:202100460116

2023 年 11 月 29 日

目录

1 实验原理

1.1 AES 分组加密

AES(Advanced Encryption Standard) 算法即高级加密标准，AES 算法是用来替代原先的 DES，目前已经被全世界广泛使用，同时 AES 已经成为对称密钥加密中最流行的算法之一。根据支持的密钥长度的不同，可以将 AES 分为 AES128，AES192，AES256 三种。本文主要针对 AES128 进行研究.

AES 算法的加密共需要进行 10 轮, 其中每轮 (除最后一轮) 包括四个步骤：字节替换、行移位、列混淆和轮密钥加. 算法总的流程如下:

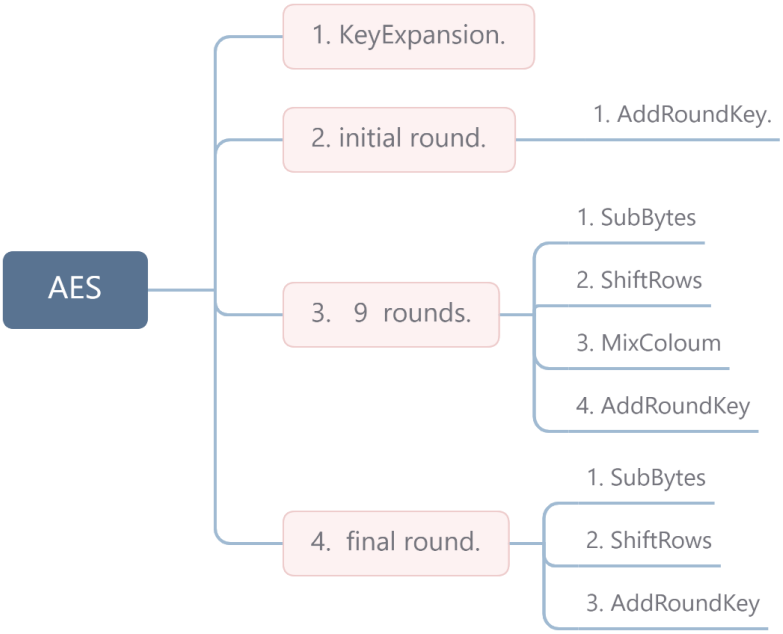

图 1: AES 流程

1.2 AES 轮函数

1.2.1 AddRroundKey

在每次的加密循环中，都会由主密钥产生轮密钥, 密钥大小会跟原矩阵一样. AddRound-Key 步骤将轮密钥与 state 矩阵对应位置字节进行异或.

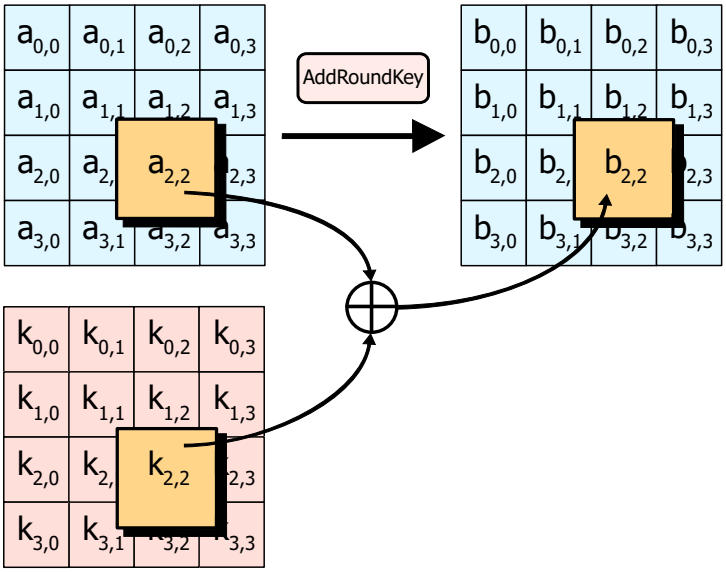

图 2: AES-AddRoundKey

1.2.2 SubBytes

在 SubBytes 步骤中，矩阵中的各字节通过一个 8 位输入输出的 S-box 进行转换, 该步骤为算法提供了非线性. AES 的 SubBytes 步骤可通过查表和计算两种方式进行实现.

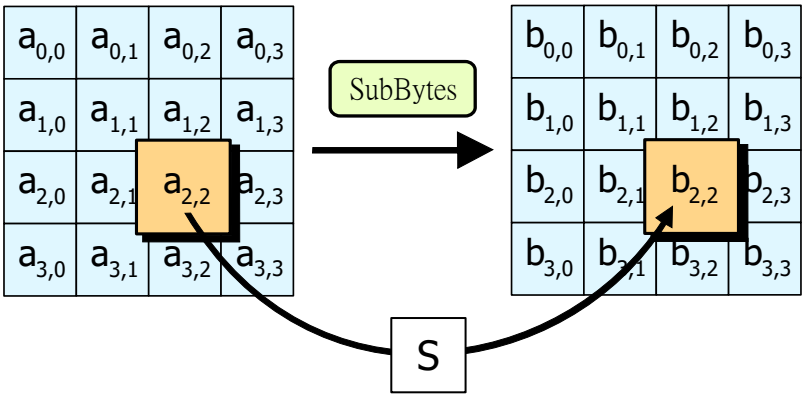

图 3: AES-SubBytes

1.2.3 ShiftRows

ShiftRows 步骤中，每一行都向左循环位移某个偏移量。在 AES128 中, 第一行维持不变，第二行中每个字节向左循环移动 1 字节。第三行和第四行分别向左循环位移 2 和 3 字节。

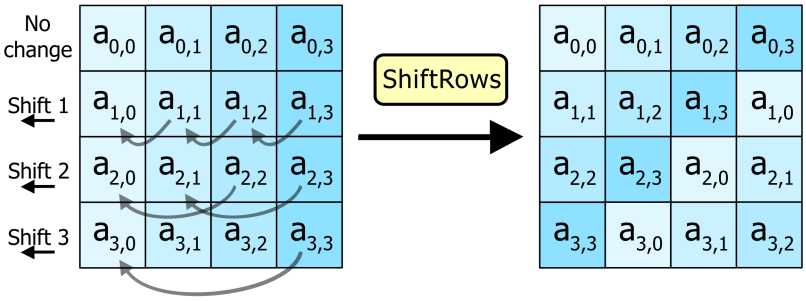

图 4: AES-ShiftRows

1.2.4 MixColoum

MixColumns 步骤将每一列的四个字节透过线性变换互相结合, 输出新的 4 字节. 每一个输入的字节都会对输出的四个字节造成影响, 为密码算法提供扩散效果。

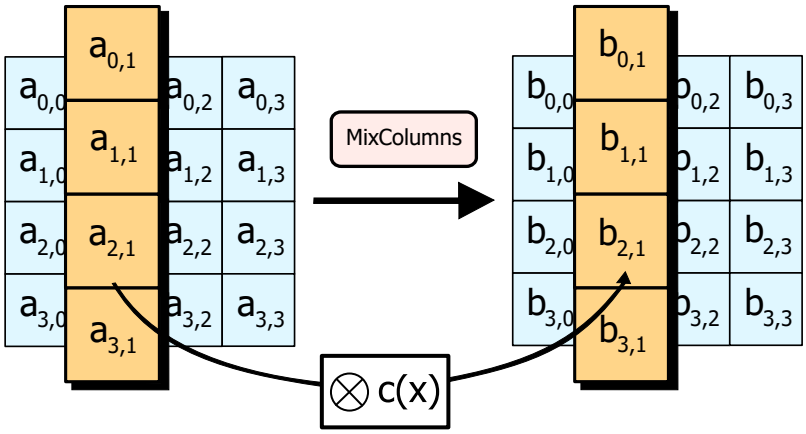

图 5: AES-MixColumns

1.3 AES 密钥拓展

密钥扩展将 128 位的密钥扩展为 10 轮 128bit 轮密钥，并用于每轮加密.

针对 AES-128 定义:

N 为密钥以 32 位字衡量的长度：4 个字.

$K_0, K_1 ... KN-1$ 作为原始密钥的 32 位字.

R 表示所需的轮密钥的数量:11.

$W_0, W_1 ... W_{4R-1}$ 作为拓展密钥的 32 位字.

RotWord() 为左循环移位一字节.

$$\text{RotWord}(\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \end{bmatrix}) = \begin{bmatrix} b_1 & b_2 & b_3 & b_0 \end{bmatrix}$$

$$\text{SubWord}(\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \end{bmatrix}) = \begin{bmatrix} \text{S}(b_0) & \text{S}(b_1) & \text{S}(b_2) & \text{S}(b_3) \end{bmatrix}$$

$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus rcon_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod{N} \\ W_{i-N} \oplus \text{SubWord}(W_{i-1}) & \text{if } i \geq N, N > 6, \text{ and } i \equiv 4 \pmod{N} \\ W_{i-N} \oplus W_{i-1} & \text{otherwise.} \end{cases}$$

2 实验过程

2.1 AES 轮函数

1) AddRoundKey. 将 state 矩阵和轮密钥矩阵按字节进行异或, 得到新 state 矩阵.

```
uint8* AddRoundKey(uint8 state[4][4], uint8* RoundKey[4])
{
  for (int i = 0; i < 4; i++){
    for (int j = 0; j < 4; j++){
      state[i][j] ^= RoundKey[i][j];
    }
  }return 0;
}
```

2) SubBytes.SubBytes 步骤可通过运算和查表两种方式实现, 本文选择使用查表方式进行实现. 提前算出 S-box 的所有输入输出对应关系矩阵, 针对 state 中每一个字节进行查表得到过 S-box 后的值.

```
uint8* SubBytes(uint8 state[4][4], uint8 SBox[16][16])
{
  for (int i = 0; i < 4; i++){
    for (int j = 0; j < 4; j++){
      uint8 m = state[i][j] >> 4;
      uint8 n = state[i][j] & 0x0F;
      state[i][j] = SBox[m][n];
    }
  }return 0;
}
```

3) ShiftRows. 将 state 矩阵第 i 行按字节循环左移 i 字节 (i=1,2,3,4).

```
uint8* ShiftRow(uint8 state[4][4]){
  for (int i = 0; i < 4; i++){
    for (int j = 0; j < i; j++){
        uint8 temp = state[i][0];
        for (int k = 0; k < 3; k++){
          state[i][k] = state[i][k + 1];
        }
        state[i][3] = temp;
    }
  }return 0;
}
```

4) MixColoum

针对 MixColoum 步骤有两种实现方法, 分别为查表实现和运算实现. 本问对两种方法均进行了实现.

(A) 查表实现. 预计算 MixColumn 运算所有输入对应的输出, 需要使用时直接查表, 无需再次计算.

```
uint8* MixColumnT_table(uint8 state[4][4])
{
  uint8 tempResult[4];
  for (int col = 0; col < 4; col++)
  {
    for (int j = 0; j < 4; j++){
      tempResult[j] = 0x00;
    }
    for (int j = 0; j < 4; j++){
      tempResult[0] ^= MixColumnTable[state[j][col]][(0 + 4 - j) % 4];
      tempResult[1] ^= MixColumnTable[state[j][col]][(1 + 4 - j) % 4];
      tempResult[2] ^= MixColumnTable[state[j][col]][(2 + 4 - j) % 4];
      tempResult[3] ^= MixColumnTable[state[j][col]][(3 + 4 - j) % 4];
    }
    for (int row = 0; row < 4; row++){
      state[row][col] = tempResult[row];
    }
  }return 0;
}
```

(B) 运算实现.

MixColumn 域上运算实现.

MixColumn 过程输入矩阵 B, 在 $GF(2^8)$ 域上进行如下运算, 得到输出矩阵 C.

$$\begin{pmatrix} C_0 & C_4 & C_8 & C_{12} \\ C_1 & C_5 & C_9 & C_{13} \\ C_2 & C_6 & C_{10} & C_{14} \\ C_3 & C_7 & C_{11} & C_{15} \end{pmatrix} = \begin{pmatrix} 01 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \end{pmatrix}$$

$GF(2^8)$ 域上加法实质为按位异或, 乘法实质为 $GF(2^8)$ 域上多项式乘法. 因为 AES 加密只需计算域内元素 0x01,0x02,0x03 与 state 元素的乘法, 因此可将 0x03 拆分成 0x01+0x02, 最终只需实现 $GF(2^8)$ 域上 0x02 与其他元素的乘法, 可利用固定公式计算该结果.

$$\{02\}_{16} \times \{b_7b_6b_5b_4b_3b_2b_1b_0\}_2 = \begin{cases} \{b_6b_5b_4b_3b_2b_1b_00\}_2 & if\ b_7 = 0 \\ \{b_6b_5b_4b_3b_2b_1b_00\}_2 \oplus 00011011_2 & if\ b_7 = 1 \end{cases}$$

$$\{03\}_{16} \times \{b_7b_6b_5b_4b_3b_2b_1b_0\}_2 = \{02\}_{16} \times \{b_7b_6b_5b_4b_3b_2b_1b_0\}_2 \oplus \{b_7b_6b_5b_4b_3b_2b_1b_0\}_2$$

```
uint8 GF8Mul(uint8 a, uint8 b)
{
  if (a == 1)     { return b; }
```

```
    uint8 result = b << 1;
    if (b >> 7 == 1){result ^= 0b00011011;}
    if (a == 3)     {result ^= b;}
    return result;
  }
  uint8* MixColumn(uint8 state[4][4])
  {
    uint8 matrix[4][4] ={
      {2,3,1,1},
      {1,2,3,1},
      {1,1,2,3},
      {3,1,1,2}
    };
    uint8 tempResult[4];
    for (int col = 0; col < 4; col++)
    {
      for (int row = 0; row < 4; row++){
        tempResult[row] = GF8Mul(matrix[row][0], state[0][col]);
        for (int k = 1; k < 4; k++){
          tempResult[row] ^= GF8Mul(matrix[row][k], state[k][col]);
        }
      }
      for (int row = 0; row < 4; row++){
        state[row][col] = tempResult[row];
      }
    }return 0;
  }
```

2.2 AES 密钥拓展

按照如下公式计算所有轮密钥.

$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus rcon_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod N \\ W_{i-N} \oplus \text{SubWord}(W_{i-1}) & \text{if } i \geq N, N > 6, \text{ and } i \equiv 4 \pmod N \\ W_{i-N} \oplus W_{i-1} & \text{otherwise.} \end{cases}$$

```
uint8** KeyGenerateOnce(uint8* key[4], uint8 &KeyGenerateOrdinal)
{
  //新的轮密钥放置在堆中
  uint8** roundKey = new uint8 * [4];
  for (int i = 0; i < 4; i++)
  {
    roundKey[i] = new uint8[4];
  }
  //1. 生成第0列
  for (int i = 0; i < 4; i++)
  {
    uint8 temp = key[(i + 1) % 4][3];
    uint8 m = temp >> 4;
    uint8 n = (temp << 4);
    n = n >> 4;
```

```
16      roundKey[i][0] = SBox[m][n];
17      roundKey[i][0] ^= key[i][0];
18      if (i == 0)
19      {
20        roundKey[i][0] ^= RCon[KeyGenerateOrdinal];
21        KeyGenerateOrdinal = (KeyGenerateOrdinal + 1) % 10;
22      }
23    }
24    //2. 生成1-3列
25    for (int row = 0; row < 4; row++)
26    {
27      for (int col = 1; col < 4; col++)
28      {
29        roundKey[row][col] = key[row][col] ^ roundKey[row][col - 1];
30      }
31    }
32    return roundKey;
33  }
```

3 实验结果及分析

本文实现了 AES 的 C++ 软件实现. 在此基础上我们测试了不同长度 (MB) 输入下算法的时间开销, 并计算了其对应的数据吞吐率.

使用 matlab 软件对原始测试数据进行处理后绘制数据图, 结果如下:

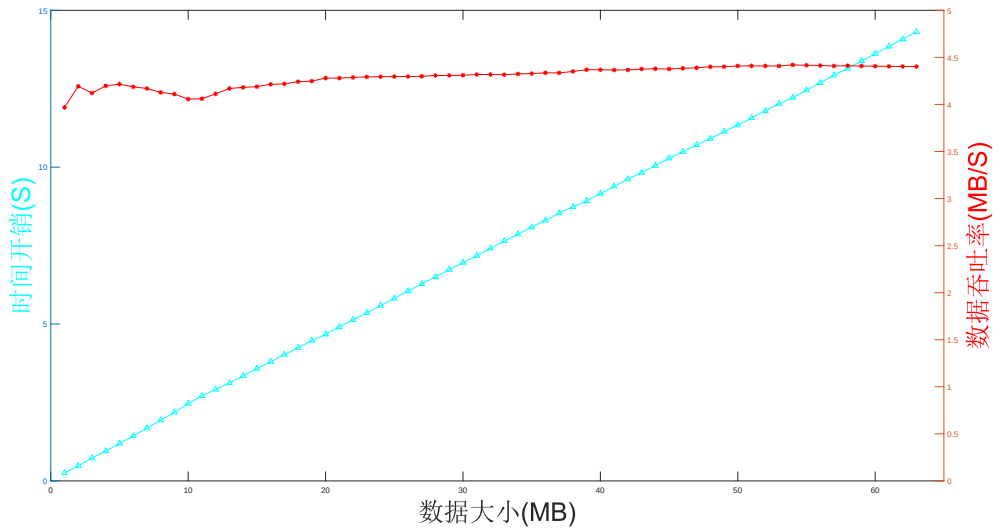

图 6: AES implement benchmark

可观察到随着加密数据量的增加时间开销不断增大, 基本呈线性趋势增长. 而数据吞吐率在数据量较小时有一定波动, 随着数据量的增大逐步趋于稳定, 基本稳定在 4.4MB/S 左右.

为使软件实现效率更高, 可进行如下方式的算法效率优化: (1) 减少函数调用, 在单个函数内实现完整的加密过程. 但会减弱程序可读性和拓展性. (2) 进行循环展开, 提高运行效率. (3) 调用 CPU 指令集运行 AES 加密.

参考文献

[1] https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.