密码学lab3

1813540 陈鸿运

2020 年 12 月 8 日

目录

1 AES加密原理

AES算法以字节为基本单位进行操作，各个主要加密的层级有密钥加法层、字节代换层、行位移层、列混淆层。下面是AES进行加密的流程图*：

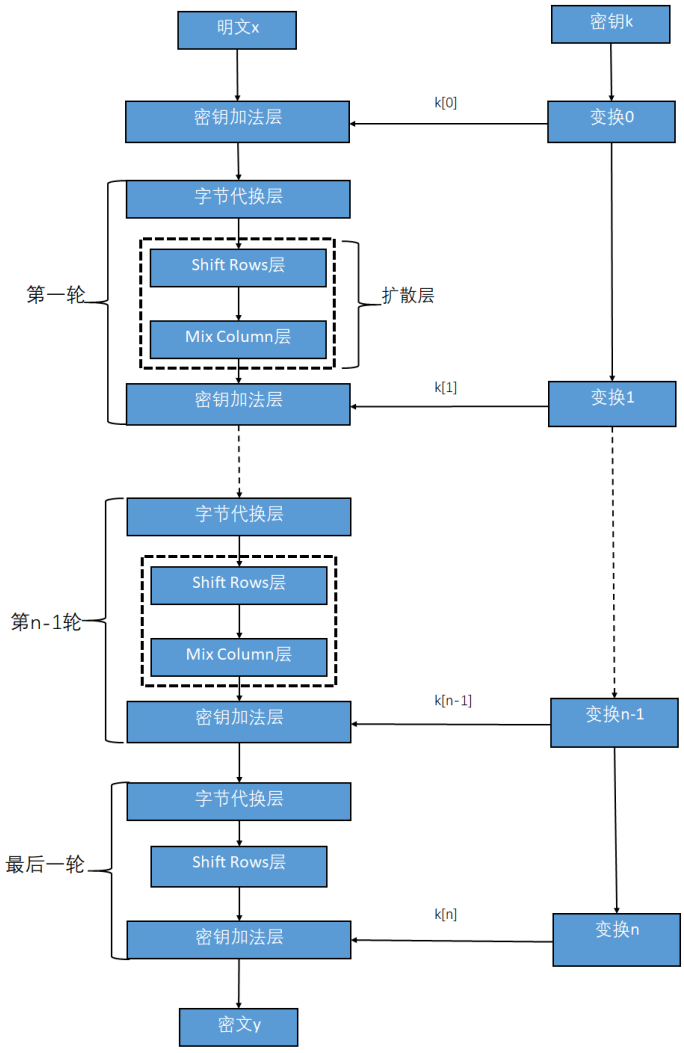


图 1: AES加密算法整体流程图

*图片源自网络

下面我们不妨考虑其具体的实现：

1.1 矩阵转化

在AES中，我们将输入的明文与密钥转化为相应的矩阵进行处理。因此我们需要实现这一函数：

```
1         void input_to_array(const unsigned char *input,unsigned char (*
    array)[4])
2         {
3           for (int i = 0; i < 16; i++)
4           {
5             array[i & 0x03][i >> 2] = input[i];
6           }
7         }
8
```

这里我们将输入的数据转化为一个$4*4$的矩阵，我们分别取i的高四位与低四位作为列号与行号，最终将数据存到一个二维数组中。

1.2 密钥扩展

我们初始的密钥是128比特，但是AES需要进行十轮加密，在第一轮加密之前有一个字节代换层，AES要求每一轮都需要不同的密钥加密。因此，我们需要对密钥进行扩展。

```
1       void get_keys(const unsigned char(*key_array)[4], unsigned char(*
    extend_key_array)[44])
2       {
3         for (int i = 0; i < 16; i++)
4         {
5           extend_key_array[i & 0x03][i >> 2] = key_array[i & 0x03][i >>
    2];
6         }
7
8         for (int i = 1; i < 11; i++)
9         {
10          G_Function(extend_key_array, 4 * i);
11
12          for (int k = 0; k < 4; k++)
13          {
14            extend_key_array[k][4 * i] = extend_key_array[k][4 * i] ^
    extend_key_array[k][4 * (i - 1)];
15          }
16
17          for (int j = 1; j < 4; j++)
18          {
19            for (int k = 0; k < 4; k++)
20            {
21              extend_key_array[k][4 * i + j] = extend_key_array[k][4 * i +
     j - 1] ^ extend_key_array[k][4 * (i - 1) + j];
```

```
            }
          }

        }

      }

```

我们最终得到的扩展密钥中，共有44列，前4列就是初始的密钥。而之后的40列，各列之间经过G函数或者异或操作不断迭代，最终得到了需要的扩展密钥。我们在处理过程中用到的G函数以及S盒代换的代码如下：

```
      void Key_S(unsigned char(*extend_key_array)[44], unsigned int n_col)
      {
        for (int i = 0; i < 4; i++)
        {
          extend_key_array[i][n_col] = S[(extend_key_array[i][n_col]) >> 4][(extend_key_array[i][n_col]) & 0x0F];
        }
      }

      void G_Function(unsigned char(*extend_key_array)[44], unsigned int n_col)
      {
        for (int i = 0; i < 4; i++)
        {
          extend_key_array[i][n_col] = extend_key_array[(i + 1) % 4][n_col - 1];
        }

        Key_S(extend_key_array, n_col);

        extend_key_array[0][n_col] ^= Rcon[n_col / 4];

      }


```

1.3 密钥加法层

下面我们就来查看加密过程中用到的各个层函数。首先来看密钥加法层，其原理十分简单，就是异或操作，实现也十分简单，代码如下：

```
      void round_key_add(unsigned char(*plain_array)[4], unsigned char(*extend_key_array)[44],unsigned int min_col)
      {
        for (int i = 0; i < 4; i++)
        {
          for (int j = 0; j < 4; j++)
          {
            plain_array[i][j] ^= extend_key_array[i][j + min_col];
```

```
        }
      }
    }

```

1.4 字节代换层

该层使用S盒将输入的矩阵中的每个元素进行代换，操作也并不复杂，代码如下：

```
    static const int S[16][16] =
    {
      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
    };

    void plain_S_Sub(unsigned char *plain_array)
    {
      for (int i = 0; i < 16; i++)
      {
        plain_array[i] = S[plain_array[i] >> 4][plain_array[i] & 0x0F];
      }
    }

```

1.5 行位移层

在该层AES对数据矩阵的处理是这样的：第一行不做处理，第二行左移8bit，第三行左移16bit，第三行左移24 bit，那么我们不难得到如下代码：

```
void left_shift_row(unsigned int *plain_array)
{
  plain_array[1] = (plain_array[1] >> 8) | (plain_array[1] << 24);

  plain_array[2] = (plain_array[2] >> 16) | (plain_array[2] << 16);

  plain_array[3] = (plain_array[3] >> 24) | (plain_array[3] << 8);

}

```

1.6 列混淆层

该层是AES中最为复杂的一部分，但同时也是这一部分大大增强了AES破解的难度，由于在附件的代码中我们加入了较为详尽的注释，因此不再此做过多说明。

```
const unsigned char mix_array[4][4] =
{
  0x02, 0x03, 0x01, 0x01,
  0x01, 0x02, 0x03, 0x01,
  0x01, 0x01, 0x02, 0x03,
  0x03, 0x01, 0x01, 0x02
};

void mix_col(unsigned char(*plain_array)[4])
{
  unsigned char temp_array[4][4];

  memcpy(temp_array, plain_array, 16);


  for (int i = 0; i < 4; i++)
  {
    for (int j = 0; j < 4; j++)
    {
      plain_array[i][j] =
        Gal_Mul(mix_array[i][0], temp_array[0][j]) ^
        Gal_Mul(mix_array[i][1], temp_array[1][j]) ^
        Gal_Mul(mix_array[i][2], temp_array[2][j]) ^
        Gal_Mul(mix_array[i][3], temp_array[3][j]);
    }
  }

}

```

值得注意的是，这里我们使用的矩阵乘法是在伽罗瓦域内的乘法，所以我们需要自行实现该乘法：

```
char Gal_Mul(unsigned char L_num, unsigned char R_num)
{
  unsigned char result = 0;

  while (L_num)
  {
    if (L_num & 0x01)
    {
      result ^= R_num;
    }

    L_num = L_num >> 1;

    if (R_num & 0x80)
    {
      R_num = R_num << 1;

      R_num ^= 0x1B;
    }
    else
    {
      R_num = R_num << 1;
    }
  }

  return result;

}

```

1.7 加密流程代码

在上文中我们说明了*AES*加密过程需要的函数，下面我们就可以依据在文章一开始所展示的流程图进行加密过程的实现了：

```
    void encrypt(const unsigned char *plain_text, const unsigned char *
key,unsigned char *cipher_text)
    {
      unsigned char plain_array[4][4];
      unsigned char key_array[4][4];
      unsigned char extend_key_array[4][44];

      memset(plain_array, 0, 16);
      memset(key_array, 0, 16);
      memset(extend_key_array, 0, 176);

      input_to_array(plain_text, plain_array);

      input_to_array(key, key_array);

      get_keys(key_array, extend_key_array);

      round_key_add(plain_array, extend_key_array, 0);
```

```
18
19
20            for (int i = 1; i < 10; i++)
21            {
22              plain_S_Sub((unsigned char *)plain_array);
23
24              left_shift_row((unsigned int *)plain_array);
25
26              mix_col(plain_array);
27
28              round_key_add(plain_array, extend_key_array, 4 * i);
29
30            }
31
32            plain_S_Sub((unsigned char *)plain_array);
33
34            left_shift_row((unsigned int *)plain_array);
35
36            round_key_add(plain_array, extend_key_array, 4 * 10);
37
38
39
40            array_to_output(plain_array, cipher_text);
41
42
43          }
44
```

2 AES解密原理

*AES*的解密过程与加密过程是完全相反的，而不是像*DES*那样只需逆向使用密钥即可。因此，我们在实现解密过程时，需要进行更多的考虑。

2.1 逆向移位函数

在解密过程中不再是左移而是右移，因此我们需要对原有的移位函数进行修改，如下：

```
1           void right_shift_row(unsigned int *cipher_array)
2           {
3             cipher_array[1] = (cipher_array[1] << 8) | (cipher_array[1] >> 24)
      ;
4
5             cipher_array[2] = (cipher_array[2] << 16) | (cipher_array[2] >>
      16);
6
7             cipher_array[3] = (cipher_array[3] << 24) | (cipher_array[3] >> 8)
      ;
8
9           }
10
```

2.2 逆S盒代换

解密过程使用的是逆S盒而非S盒，因此此处也需要修改，只需将S换为逆S盒即可：

```
        void cipher_S_Sub(unsigned char *cipher_array)
        {
          for (int i = 0; i < 16; i++)
          {
            cipher_array[i] = S_1[cipher_array[i] >> 4][cipher_array[i] & 0
    x0F];
          }
        }

```

2.3 列混淆层

该层与加密过程的列混淆层的主要区别就是使用了逆混淆矩阵：

```
        const unsigned char mix_array_1[4][4] =
        {
          0x0E, 0x0B, 0x0D, 0x09,
          0x09, 0x0E, 0x0B, 0x0D,
          0x0D, 0x09, 0x0E, 0x0B,
          0x0B, 0x0D, 0x09, 0x0E
        };

        void re_mix_col(unsigned char(*cipher_array)[4])
        {
          unsigned char temp_array[4][4];

          memcpy(temp_array, cipher_array, 16);


          for (int i = 0; i < 4; i++)
          {
            for (int j = 0; j < 4; j++)
            {
              cipher_array[i][j] =
                Gal_Mul(mix_array_1[i][0], temp_array[0][j]) ^
                Gal_Mul(mix_array_1[i][1], temp_array[1][j]) ^
                Gal_Mul(mix_array_1[i][2], temp_array[2][j]) ^
                Gal_Mul(mix_array_1[i][3], temp_array[3][j]);
            }
          }

        }

```

2.4 解密流程代码

下面我们就可以实现解密函数的代码，可以看到，它与加密函数完全相反：

```
1         void decrypt(const unsigned char *cipher_text, const unsigned char *
      key, unsigned char *plain_text)
2         {
3           unsigned char cipher_array[4][4];
4           unsigned char key_array[4][4];
5           unsigned char extend_key_array[4][44];
6
7           memset(cipher_array, 0, 16);
8           memset(key_array, 0, 16);
9           memset(extend_key_array, 0, 176);
10
11          input_to_array(cipher_text, cipher_array);
12          input_to_array(key, key_array);
13
14
15          get_keys(key_array, extend_key_array);
16
17
18
19          round_key_add(cipher_array, extend_key_array, 4 * 10);
20
21          right_shift_row((unsigned int *)cipher_array);
22
23          cipher_S_Sub((unsigned char *)cipher_array);
24
25
26          for (int i = 9; i > 0; i--)
27          {
28            round_key_add(cipher_array, extend_key_array, 4 * i);
29
30            re_mix_col(cipher_array);
31
32            right_shift_row((unsigned int *)cipher_array);
33
34            cipher_S_Sub((unsigned char *)cipher_array);
35
36          }
37
38          round_key_add(cipher_array, extend_key_array, 0);
39
40
41          array_to_output(cipher_array, plain_text);
42        }
43
```

3 AES实现效果

下面我们来查看所实现的AES加密解密算法代码执行的效果，我们对两例测试数据进行测试，得到如下结果：

可以看到，程序对输入的数据进行了正确的加密与解密。

4 AES雪崩效应分析

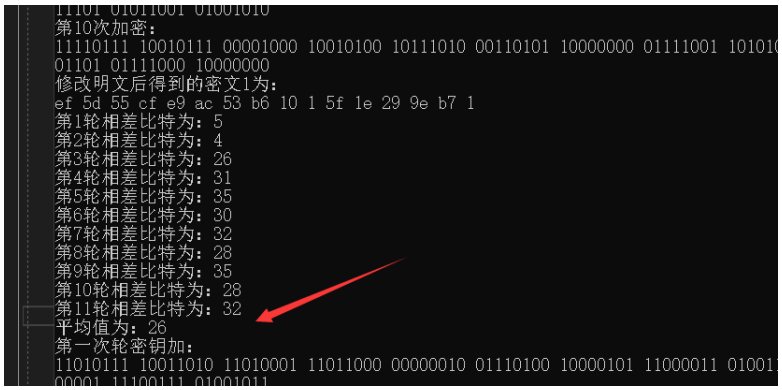
下面我们尝试分析AES的雪崩效应，我们对之前实现的程序稍作修改，便可以进行统计分析。

我们首先修改明文的数据，不妨以第一个测试数据作试验：

```
1        //the plain_text before change
2        unsigned char plain_text[] = { 0x00, 0x01, 0x00, 0x01, 0x01, 0xa1, 0
     x98, 0xaf, 0xda, 0x78, 0x17, 0x34, 0x86, 0x15, 0x35, 0x66 };
3
4        //the plain_text after change
5        unsigned char change_plain_text_1[] = { 0x00, 0x01, 0x00, 0x01, 0x01
     , 0xa1, 0x98, 0xaf, 0xda, 0x78, 0x17, 0x34, 0x3f, 0x15, 0x35, 0x66 };
6
```

我们修改了第13个数据(倒数第4个)，即从$0x86$变为$0x3f$。

执行我们的统计程序，得到结果如下：

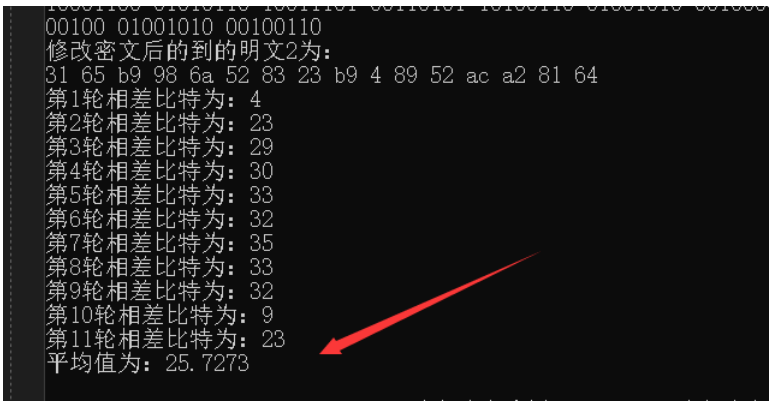


由于数据繁多，我们不在此全部列出。但是可以看到，仅仅修改了明文的8位数据，就使得加密后的密文完全不一样。由此可见AES是具有雪崩效应的。

第二组数据样例测试如下：

```
1        //the cipher_text before change
2        unsigned char cipher_text_2[] = { 0x39, 0x25, 0x84, 0x1d, 0x02, 0xdc
     , 0x09, 0xfb, 0xdc, 0x11, 0x85, 0x97, 0x19, 0x6a, 0x0b, 0x32 };
3
4        //the cipher_text after change
5        unsigned char change_cipher_text_2[] = { 0x43, 0x25, 0x84, 0x1d, 0
    x02, 0xdc, 0x09, 0xfb, 0xdc, 0x11, 0x85, 0x97, 0x19, 0x6a, 0x0b, 0x32 };
6
```

这里我们将第一个数据$0x39$换成了$0x43$，我们查看结果：



同样的，解密的数据得到了完全不同的明文，足以验证AES的雪崩效应。

参考文献

[1] 现代密码学/杨波编著.— 4版.— 北京：清华大学出版社，2017（2018.8重印）

[2] https://bbs.pediy.com/thread-253884.htm