密码学实验报告 5

张天辰 17377321

2019 年 4 月 17 日

1 AES 算法

1.1 算法简介

这里只讨论十轮 AES。

加密算法输入为 16 字节的分组，组成一个 4×4 的矩阵 S。针对 S 有以下几种操作：

1 字节代替。将 S 中的每个字节代替为 S 盒中的某个字节。代替规则为原始字节的前四位为行数，后四位为列数。考虑到算法实现的时间与空间的取舍，字节代替使用打表实现，因此算法毋需构造 S 盒。同理可以实现逆 S 盒用于解密。

2 行移位。将 S 中每一行分别循环左移 0、1、2、3 个字节，实现置换。类似地可以实现逆行移位用于解密。

3 列混淆。实际上可以写成如下的运算：

$$\mathbf{S'} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \mathbf{S} \tag{1}$$

这里的所有运算都是基于 $GF(2^8)$ 有限域上的运算，其中使用的 8 次不可约多项式为 $x^8+x^4+x^3+x+1$。

逆变换可以用相似的方法实现：

$$\mathbf{S'} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \mathbf{S} \tag{2}$$

4 轮密钥加。每个字节与轮密钥中的对应字节异或。

密钥扩展算法中，将四个字节组成一个字，原始密钥生成初始四个字 w_0, w_1, w_2, w_3。对 $i \geqslant 4$，有：

$$w_i = \begin{cases} w_{i-1} \oplus w_{i-4} & 4 \nmid i \\ g(w_{i-1}) \oplus wi - 4 & 4 \mid i \end{cases} \tag{3}$$

每次取四个字组成轮密钥。

1.2 算法实现

算法都是类方法，所以可能没有参数。

1 密钥扩展算法

(1) g 函数

Algorithm 1 g 函数

输入: 轮数 $round$，待操作字 $word$

输出: 操作后字 $tempWord$

```
function G(word, round)
    tempWord ← word[1 :] + [word[0]]
    SBOX(tempWord)
    tempWord[0] ← tempWord[0] ⊕ RC[round]
    return tempWord
end function
```

(2) 密钥扩展

Algorithm 2 密钥扩展

输入: 字列表 $wordList$

输出: 操作后字列表 $wordList$

```
function GENERATEKEY(wordList)
    for flag ∈ [4, 43] do
        if 4|flag then
            wordList[flag] ← wordList[flag − 1]⊕ G(wordList[flag − 1], [flag/4])
        else
            wordList[flag] ← wordList[flag − 1] ⊕ wordList[flag − 1]
        end if
    end for
end function
```

2 AES 加密部件算法

(a) 字节代替。解密算法只需把算法中的 SBOX 替换成别的表即可。

Algorithm 3 字节代替

```
function SUBSTITUTE
    for i ∈ [0, 3] do
        for j ∈ [0, 3] do
            temp ← S[i][j]
            S[i][j] ← SBOX[temp//16][temp%16]
        end for
    end for
```

```
8: end function
```

(b) 行移位。解密算法只不过是左移改成右移，不再赘述。

Algorithm 4 行移位

```
1: function shiftRows
2:     S[1] ← S[1][1 :] + [S[1][0]]
3:     S[2] ← S[2][2 :] + S[2][: 2]
4:     S[3] ← [S[3][3]] + S[3][: 3]
5: end function
```

(c) 列混淆。解密算法就是把 mixColumnTable 换成另一个表。算法中的 + 和 ∗ 是有限域运算。

Algorithm 5 列混淆

```
1: function mixColumn
2:     temp ← [[0 × 4] × 4]
3:     for i ∈ [0, 3] do
4:         for j ∈ [0, 3] do
5:             for k ∈ [0, 3] do
6:                 temp[i][j] ← temp[i][j] + mixColumnTable[i][k] ∗ S[k][j]
7:             end for
8:         end for
9:     end for
10:    S ← temp
11: end function
```

(d) 轮密钥加，逆算法考虑到交换顺序，一并给出。算法中的 + 和 ∗ 是有限域运算。

Algorithm 6 轮密钥加

输入: key

```
1: function addRoundKey(key)
2:     for i ∈ [0, 3] do
3:         for j ∈ [0, 3] do
4:             S[i][j] ← S[i][j] ⊕ key[i][j]
5:         end for
6:     end for
7: end function
8: function revAddRoundKey(key)
9:     temp ← [[0 × 4] × 4]
10:    for i ∈ [0, 3] do
11:        for j ∈ [0, 3] do
12:            for k ∈ [0, 3] do
13:                temp[i][j] ← temp[i][j] + revMixColumnTable[i][k] ∗ key[k][j]
14:            end for
```

```
        end for
    end for
    for i ∈ [0, 3] do
        for j ∈ [0, 3] do
            S[i][j] ← S[i][j] ⊕ temp[i][j]
        end for
    end for
end function
```

3 AES 算法

Algorithm 7 AES

输入: *key* 扩展列表，*plain*（加密）*cipher*（解密）

输出: *cipher*（加密），*plain*（解密）

```
function AES_Encrypt(plain, key)
    state ← plain → matrix
    for round ∈ [0, 10] do
        if round == 0 then
            state.addRoundKey(key.getRoundKey(round))
        else
            state.substitute()
            state.shiftRows()
            if round ≠ 10 then
                state.mixColumn()
            end if
            state.addRoundKey(key.getRoundKey(round))
        end if
    end for
    return cipher ← state → 十六进制字符串
end function
function AES_Decrypt(cipher, key)
    state ← copher → matrix
    for round ∈ [10 → 0] do
        if round == 10 then
            state.addRoundKey(key.getRoundKey(round))
        else
            state.revSubstitute()
            state.revShiftRows()
            if round ≠ 0 then
                state.revMixColumn()
```

```
            state.revAddRoundKey(key.getRoundKey(round))
        else state.addRoundKey(key.getRoundKey(round))
        end if
    end if
  end for
  return plain ← state → 十六进制字符串
end function
```

1.3 测试样例

这里只测试加密，解密测试可以从此后大批量加密文件的测试中得到正确性的验证。

```
Plain:0123456789abcdeffedcba9876543210
Key:0f1571c947d9e8590cb7add6af7f6798
Cipher:ff0b844a0853bf7c6934ab4364148fb9
[Finished in 0.1s]
```

图 1: AES

2 AES 工作模式

2.1 算法简介

1 密文分组链接（CBC）这种工作模式需要初始向量。将明文第一个分组加密后与初始向量异或得到第一组密文。对此后的每一组，将其加密后与前一组密文异或得到密文。这样的工作模式使得密文的每一个分组都与其之前的所有明文有关。解密时，将第一个密文分组解密后与初始向量异或，对此后每一组，将其解密后与前一组密文异或得到明文。

此外，这种工作模式需要对明文进行填充，以保证明文是整数个 128bit 分组。本方案采用 PKCS5 填充方式，即在最后一个分组为 x 个字节时：若 $x < 16$，则在 x 后填充 $16 - x$ 个字节，每个字节值为 $16 - x$；若 $x = 16$，则在 x 后填充 16 个字节，每个字节值为 16。

2 密文反馈模式（CFB）这种工作模式也需要初始向量，但是不需要进行填充。使用移位寄存器，最初时为初始向量，并对其进行加密，只取第一个字节，然后与明文第一个字节异或得到密文第一个字节。对此后的每一组，将移位寄存器左移一个字节，并将上次加密得到的一个字节密文填充在最右侧，再进行加密即可。解密算法与加密算法几乎相同，差别存在于加密的移位寄存器填充的是加密得到的密文字节，而解密的移位寄存器填充的是解密前的密文字节。

2.2 算法实现

算法都是类方法，所以可能没有参数。

1. 密文分组链接 CBC

Algorithm 8 CBC

function CBC_Encrypt
    $length \leftarrow len(plain)$
    $cipher \leftarrow []$
    if $16|length$ **then**
        $plain.extend([16] * 16)$
        $length += 16$
    else
        $plain.extend([16 - length\%16] * (16 - length\%16)$
        $length += 16 - length\%16$
    end if
    $partEncrypt \leftarrow AES()$
    for $i = 0; i < length; i += 16$ **do**
        $partPlain \leftarrow plain[i : i + 16] \oplus initVector$
        $partEncrypt.encrypt(partPlain, key)$
        $cipher.append(partEncrypt.cipher)$
        $initVector \leftarrow partEncrypt.cipher$
    end for
end function
function CBC_Decrypt
    $plain \leftarrow []$
    $partDecrypt \leftarrow AES()$
    for $i = 0; i < length; i += 16$ **do**
        $partDecrypt.decrypt(cipher[i : i + 16], key)$
        $partPlain \leftarrow partDecrypt.plain \oplus initVector$
        $plain.append(partPlain)$
        $initVector \leftarrow cipher[i : i + 16]$
    end for
    $paddle \leftarrow plain[-1][-2 :]$
    $plain[-1] \leftarrow plain[-1][: -2 * paddle]$
end function

2. 密文反馈模式 CFB

Algorithm 9 CFB

function CFB($mode$)
    $shiftReg \leftarrow initVector$
    $source \leftarrow AES()$
    $targets \leftarrow []$

```
    for p ∈ sources do
        source.encrypt(shiftReg, key)
        target ← source.cipher[: 2] ⊕ p
        if mode == "ENCRYPT" then
            shiftReg ← shiftReg[2 :] + target
        else
            shiftReg ← shiftReg[2 :] + p
        end if
        targets.append(target)
    end for
end function
```

2.3 测试样例

本方案实现了 GUI 界面。通过二进制读写文件，可以实现大小不太大的任意文件的加密。

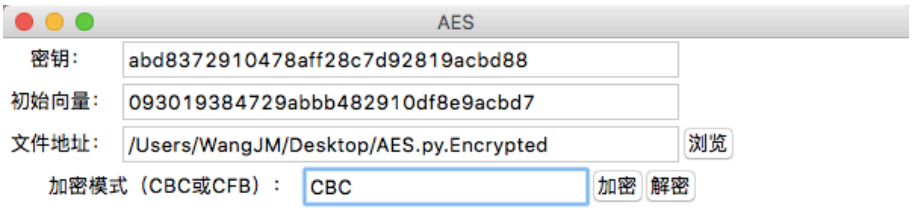

图 2: 加密

图 3: 解密

图 4: 加密后文件和解密出的文件

3 感想

AES 的实现使用面向对象的编程方式能大大简化操作，让逻辑变得非常清晰。本次实验对我的锻炼更多地来自于编程方面，因为 AES 算法本身并不复杂。我熟悉了 python 的 GUI 界面设计和面向对象的各种方法。然而 python 的运行速度给我带来了很大的困扰，其加密文件的大小只能限于几 kb。这一方面是算法优化不足的问题，更大的还是语言问题。如果使用 c++ 等编译型语言甚至是硬件语言，能大大加快速度。因此我并没有很纠结于速度的优化，而是更加重视实现本身。如果要让用 python 实现的 AES 算法去加密比较大批量的文件，实在有点勉为其难。

这次实验依然花了我很多时间，估计以后也将一直如此。无论这是好是坏，我只能接受然后面对。