密码学实验报告 实验五

2019 年 4 月 15 日

1 AES 加解密算法

1.1 算法原理

AES 是一种分组加密方法，明文分组的长度为 128 位即 16 字节，密钥长度可以为 16 字节，密钥长度可以为 16 字节，24 字节或 32 字节（128 位，192 位或 256 位）。根据密钥的长度，算法被称为 AES-128，AES-192 或 AES-256，本次实验我们针对 AES-128 进行实现。

加密和解密算法的输入是一个 128 位分组。在 FIPS PUB 197 里，这个分组被描述为4×4的字节方阵。这个分组被复制到 **state** 数组，并在加密或解密的各个阶段被修改。同样地，密钥也被描述为字节的方阵。这个密钥接着被扩展为密钥字阵列。每个字是 4 字节，128 位的密钥最终扩展为 44 字的阵列。注意在矩阵中字节是按照列进行排序的。所以，加密算法的 128 位明文分组输入的前四个字节被按顺序放在了 **in** 矩阵的第一列，接着的四个字节被放在了第二列，等等。相似地，扩展密钥的前四个字节（形成一个字）被放在 **w** 矩阵的第一列。

密码由 N 轮组成，其中轮数依赖于密钥长度，AES-128 中密钥是 10 轮。前 N-1 轮由 4 个不同的变换组成：字节代替，行移位，列混淆和轮密钥加。最后一轮仅包含三个变换，而在第一轮的前面有一个起始的单变换（轮密钥加），可以看成是第 0 轮。每一个变换输入一个或多个4×4矩阵，并输出一个4×4的矩阵，最后一轮的输出为密文。同样，密钥扩展函数产生 N+1 轮密钥，它们是互不相同的4×4矩阵。每一个轮密钥作为每轮的轮密钥加变换的一种输入。

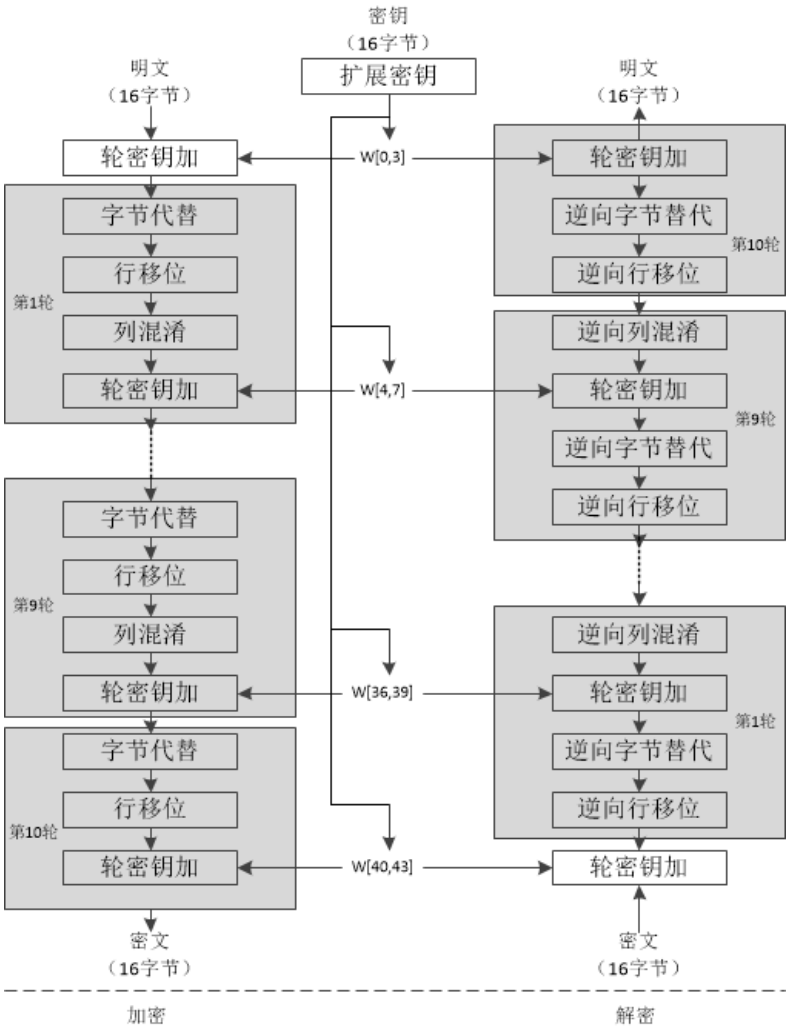

图 1 AES 加解密过程

1.2 算法实现的伪代码

AES 加密算法 利用 AES 加密

输入： 128 位明文P，128 位密钥K

输出： 128 位密文C

function $aes\_encrypt(K, P)$

$keys \leftarrow ExpandKey(K)$

$state \leftarrow P$

$AddRoundKey(state, keys[0])$

for $i \leftarrow 1$ to 9 **do**

$SubBytes(state)$

$ShiftRows(state)$

$MixColumns(state)$

$AddRoundKey(state, keys[i])$

end for

$SubBytes(state)$

$ShiftRows(state)$

$AddRoundKey(state, keys[10])$
$C \leftarrow state$
return C
end function

AES 解密算法 利用 AES 解密

输入： 128 位密文C，128 位密钥K
输出： 128 位明文P

function $aes\_decrypt(K, C)$
$keys \leftarrow ExpandKey(K)$
$state \leftarrow C$
$AddRoundKey(state, keys[10])$
for $i \leftarrow 9$ to 1 **do**
$InvShiftRows(state)$
$InvSubBytes(state)$
$AddRoundKey(state, keys[i])$
$InvMixColumns(state)$
end for
$InvShiftRows(state)$
$InvSubBytes(state)$
$AddRoundKey(state, keys[0])$
$P \leftarrow state$
return P
end function

1.3 测试样例及运行结果

我们假设密钥

$$K = (9E51760954E9F8890DC0EDEAAF70C65E)_{16}$$

尝试对明文

$$P = (0123456789ABCDEFFEDCBA9876543210)_{16}$$

进行加密，得到密文

$$C = (B341081BBE57F3E99D20AE4F0C7C043B)_{16}$$

再使用相同的密钥对密文解密，得到相同的明文，证明算法正确。

2 AES 大批量报文加解密算法

2.1 算法原理

分组密钥的输入为具有 b 位固定长度的明文分组和密钥，输出为 b 位的密文。明文长度若大于 b 位，则可简单将其分为 b 位一组的块。每次使用相同的密钥对多个分组加密，则会引发许多安全问题。为了将分组密码应用于各种实际应用，NIST 定义了五种工作模式。从本质上讲，工作模式是一项增强密码算法或者使算法适应具体应用的技术。在这里我们编程对电码本（ECB）、密文分组链接（CBC）和密文反馈（CFB）三种模式进行实现。

2.1.1 ECB 模式

电码本（ECB）模式一次处理一组明文分块，每次使用相同的密钥加密。明文若长于 b 位，则可简单将其分为 b 位一组的块，有必要则可对最后一块进行填充。解密也是一次执行一块，且使用相同的密钥。

2.1.2 CBC 模式

密文反馈（CBC）模式的输入是当前的明文组和上一个密文组的异或，而使用的密钥是相同的。这就相当于将所有的明文组链接起来了。加密算法的每次输入与本明文组没有固定的关系。和 ECB 方式类似，CBC 方式也要求如果最后的分组不是完整的分组，则需要填充至 b 位的满分组。

解密时，每个密文组分别进行解密，再与上一块密文异或就可恢复出明文。第一块明文可以和一个初始向量（IV）异或后再加密，以产生第一个明文分组。解密时将第一块密文解密的结果与 IV 异或而恢复出第一块明文。IV 是和密文具有相同的长度的数据分组。

2.1.3 CFB 模式

利用密文反馈模式（CFB）可以将分组密码当成流密码使用。流密码不需要将明文填充到长度是分组长度的整数倍，且可以实时操作。所以，待发送的字符流中任何一个字符都可以用面向字符的流密码加密后立即发送。

假设传输单位是 s 位，s 通常为 8。同使用 CBC 模式一样，明文的各个单元要链接起来，所以任意个明文单元的密文都是前面所有明文的函数。此时明文被

分为 s 位的片段而不是 b 位的单元。

加密函数的输入是 b 位的移位寄存器，它的值为初始向量 IV。加密函数输出最左边的 s 位与明文的第一个分段P_1异或得到密文的第一个单元C_1，然后将C_1发送出去，接着，移位寄存器左移 s 位，C_1填入移位寄存器的最后边 s 位。就这样直到所有的明文单元被加密完。

解密使用相同的办法，只是有一点不同：将收到的密文单元与加密函数的输出异或得到明文单元。注意这里使用的是加密函数而非解密函数。

2.2 算法实现的伪代码

ECB 模式加密 利用 ECB 模式对数据进行分组加密

输入： b 位的明文分组P，b 位密钥K
输出： b 位的密文分组C

function $ecb\_encrypt(K, P)$
    for $j \leftarrow 1$ to N **do**
        $C_j \leftarrow E(K, P_j)$
    end for
    return C
end function

ECB 模式解密 利用 ECB 模式对数据进行分组解密

输入： b 位的密文分组C，b 位密钥K
输出： b 位的明文分组P

function $ecb\_decrypt(K, C)$
    for $j \leftarrow 1$ to N **do**
        $P_j \leftarrow D(K, C_j)$
    end for
    return P
end function

CBC 模式加密 利用 CBC 模式对数据进行分组加密

输入： b 位的明文分组P，b 位密钥K，初始向量IV
输出： b 位的密文分组C

function $cbc\_encrypt(K, P, IV)$
    $C_1 \leftarrow E(K, [P_1 \oplus IV])$
    for $j \leftarrow 2$ to N **do**

$C_j \leftarrow E\left(K, \left[P_j \oplus C_{j-1}\right]\right)$

end for

return C

end function

CBC 模式解密 利用 CBC 模式对数据进行分组解密

输入： b 位的密文分组C，b 位密钥K，初始向量IV

输出： b 位的明文分组P

function $cbc\_decrypt(K, C, IV)$

$P_1 \leftarrow D(K, C_1) \oplus IV$

for $j \leftarrow 2$ to N **do**

$P_j \leftarrow D\left(K, C_j\right) \oplus C_{j-1}$

end for

return P

end function

CFB 模式加密 利用 CFB 模式对数据进行分组加密

输入： s 位的明文片段P，b 位密钥K，初始向量IV

输出： s 位的密文片段C

function $cfb\_encrypt(K, P, IV)$

$I_1 \leftarrow IV$

for $j \leftarrow 2$ to N **do**

$I_j \leftarrow LSB_{b-s}\left(I_{j-1}\right) || C_{j-1}$

end for

for $j \leftarrow 1$ to N **do**

$O_j \leftarrow E\left(K, I_j\right)$

$C_j \leftarrow P_j \oplus MSB_s\left(O_j\right)$

end for

return C

end function

CFB 模式解密 利用 CFB 模式对数据进行分组解密

输入： s 位的密文片段C，b 位密钥K，初始向量IV

输出： s 位的明文片段P

function $cfb\_decrypt(K, C, IV)$

$I_1 \leftarrow IV$

for $j \leftarrow 2$ to N **do**

$I_j \leftarrow LSB_{b-s}(I_{j-1})||C_{j-1}$

end for

for $j \leftarrow 1$ to N **do**

$O_j \leftarrow E(K, I_j)$

$P_j \leftarrow C_j \oplus MSB_s(O_j)$

end for

return P

end function

2.3 测试样例及运行结果

我们假设密钥

$$K = (0F1571C947D9E8590CB7ADD6AF7F6798)_{16}$$

初始向量

$$IV = (EA50C189A1BC6154E1ABF87091504321)_{16}$$

使用 ECB 模式加密后的结果如图 2 所示，从上至下依次为原文件、加密后文件和解密后文件。

图 2 ECB 模式

使用 CBC 模式加密后的结果如图 3 所示。

图 3 CBC 模式

使用 CFB 模式加密后的结果如图 4 所示。

图 4 CFB 模式

可以看出三种不同加密方式下得到的密文是不同的，除此之外，ECB 模式和 CBC 模式都会在解密后的文本末尾添加一些空格以补足分组大小，而 CFB 模式无需添加空格，解密出的文本与原文本相同。以上结果证明算法正确。

3 加解密程序的 UI 实现

3.1 算法原理

我们使用 python 的 Tkinter 库对程序进行了简单的 UI 设计与实现，由于 UI 无法用伪代码描述，所以贴出相关的实现代码。

3.2 UI 实现的代码

```
''' UI '''
self.root = tk.Tk()
self.mainFrame = tk.Frame(self.root)
''' TOP '''
self.topFrame = tk.Frame(self.mainFrame)
```

```
''' LEFT TOP '''
self.leftTopFrame = tk.LabelFrame(self.topFrame, text='Method')
self.method = tk.StringVar()
self.method.set('encrypt')
self.encryptRadio = tk.Radiobutton(self.leftTopFrame,
text='Encrypt', variable=self.method, value='encrypt',
indicatoron=0)
self.encryptRadio.grid(row=0, column=0)
self.decryptRadio = tk.Radiobutton(self.leftTopFrame,
text='Decrypt', variable=self.method, value='decrypt',
indicatoron=0)
self.decryptRadio.grid(row=0, column=1)
self.leftTopFrame.grid(row=0, column=0,
sticky=tk.E+tk.W+tk.N+tk.S)
''' LEFT TOP END '''
''' LEFT DOWN '''
self.leftDownFrame = tk.LabelFrame(self.topFrame, text='Mode')
modes = ('ECB', 'CBC', 'CFB')
self.modeList = tk.Listbox(self.leftDownFrame, height=len(modes),
width=15, selectmode=tk.SINGLE,
listvariable=tk.StringVar(value=modes))
self.modeList.grid(row=0, column=0)
self.leftDownFrame.grid(row=1, column=0,
sticky=tk.E+tk.W+tk.N+tk.S)
''' LEFT DOWN END '''
''' RIGHT TOP '''
self.rightTopFrame = tk.LabelFrame(self.topFrame,
text='Parameters')
tk.Label(self.rightTopFrame, text='Key').grid(row=0, column=0,
sticky=tk.E)
self.keyEntry = tk.Entry(self.rightTopFrame)
self.keyEntry.grid(row=0, column=1)
tk.Label(self.rightTopFrame, text='IV').grid(row=1, column=0,
sticky=tk.E)
self.ivEntry = tk.Entry(self.rightTopFrame)
self.ivEntry.grid(row=1, column=1)
self.rightTopFrame.grid(row=0, column=1,
sticky=tk.E+tk.W+tk.N+tk.S)
''' RIGHT TOP END '''
''' RIGHT DOWN '''
self.rightDownFrame = tk.LabelFrame(self.topFrame, text='Input
File')
tk.Button(self.rightDownFrame, text='Choose File',
command=self.chooseFile).grid(row=0, sticky=tk.W)
```

```
self.fileLabel = tk.Label(self.rightDownFrame, text='No file')
self.fileLabel.grid(row=1, sticky=tk.W)
self.rightDownFrame.grid(row=1, column=1,
sticky=tk.E+tk.W+tk.N+tk.S)
''' RIGHT DOWN END '''
self.topFrame.grid(row=0, sticky=tk.E+tk.W)
''' TOP END '''
tk.Button(self.mainFrame, text='Run',
command=self.run).grid(row=1)
self.statusLabel = tk.Label(self.mainFrame, text='Ready')
self.statusLabel.grid(row=2, sticky=tk.W)
''' DOWN END '''
self.mainFrame.grid()
self.root.title('My cipher')
```

3.3 测试样例及运行结果

软件的界面如图 5 所示。

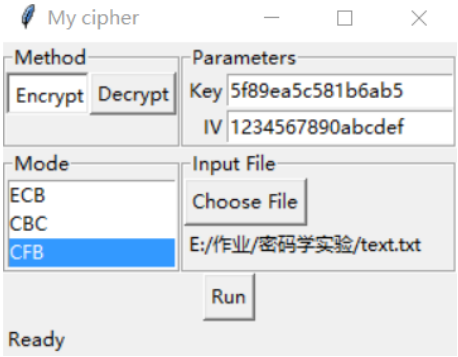

图 5 UI 界面展示

我们可以任选一种模式，通过文件选择对话框选择待加密或解密的文件，输入密钥和初始向量，对文件进行加密或解密，加解密结果如图 6 所示。

图 6 加解密结果

4 总结

在本次的实验中，我们实现了难度较高的 AES 加解密算法，在实现中，我充分利用了之前编写过的一些封装好的类，例如有限域类、矩阵类等，在具体操作中体现了面向对象编程中代码重用的便利特性。AES 中的密钥生成算法较之前的 DES 有较大难度提升，对编程实现造成了一定的挑战性，每一轮的具体步骤也是需要格外注意的，例如最后一轮的不同。总之这次 AES 的实现进一步提升了我的实现能力，使我对 AES 算法的理解进一步加深。

在实现 AES 算法之后，我还针对大批量报文的加解密，实现了三种不同的分组加密工作模式，ECB、CBC 和 CFB 模式难度依次提高，各自有不同的特性，其中 ECB 适合对单组密文的加密，CBC 比 CFB 的效率更高，但是 CFB 可以实现实时加密，类似于流密码的工作方式。

最后，我将之前的所有功能利用 Tkinter 库设计了 UI 界面，使程序的操作更加易于上手，在经过 UI 设计的学习之后，我们可以针对之后的一些软件级算法设计具体的 UI，使程序的表现形式更加多样化。