AES 加密算法实现

张境文 16337305

一、AES

AES 是最常见的对称加密算法，对称加密算法，就是加密和解密过程中使用相同的密钥。AES 为分组密码，分组密码也就是把明文分成一组一组的，每组长度相等，每次加密一组数据，直到加密完整个明文。在 AES 标准规范中，分组长度只能是 128 位，也就是说，每个分组为 16 个字节（每个字节 8 位)。密钥的长度可以使用 128 位、192 位或 256 位。密钥的长度不同，推荐加密轮数也不同。

二、AES 的基本结构

AES 是分组密码，它把明文分为一组一组，每一组 128 个 bits。每一次 AES 加密一个分组知道把所有的分组加密完。AES 的加密过程分为 10 轮，每一轮除了第 10 轮都有四个过程：1.字节替代；2.行移位；3.列混淆；4.轮密钥相加。而最后一轮只有 3 个过程，缺少了列混淆，在开始加密在前，都要先进行一次出事的轮密钥加。

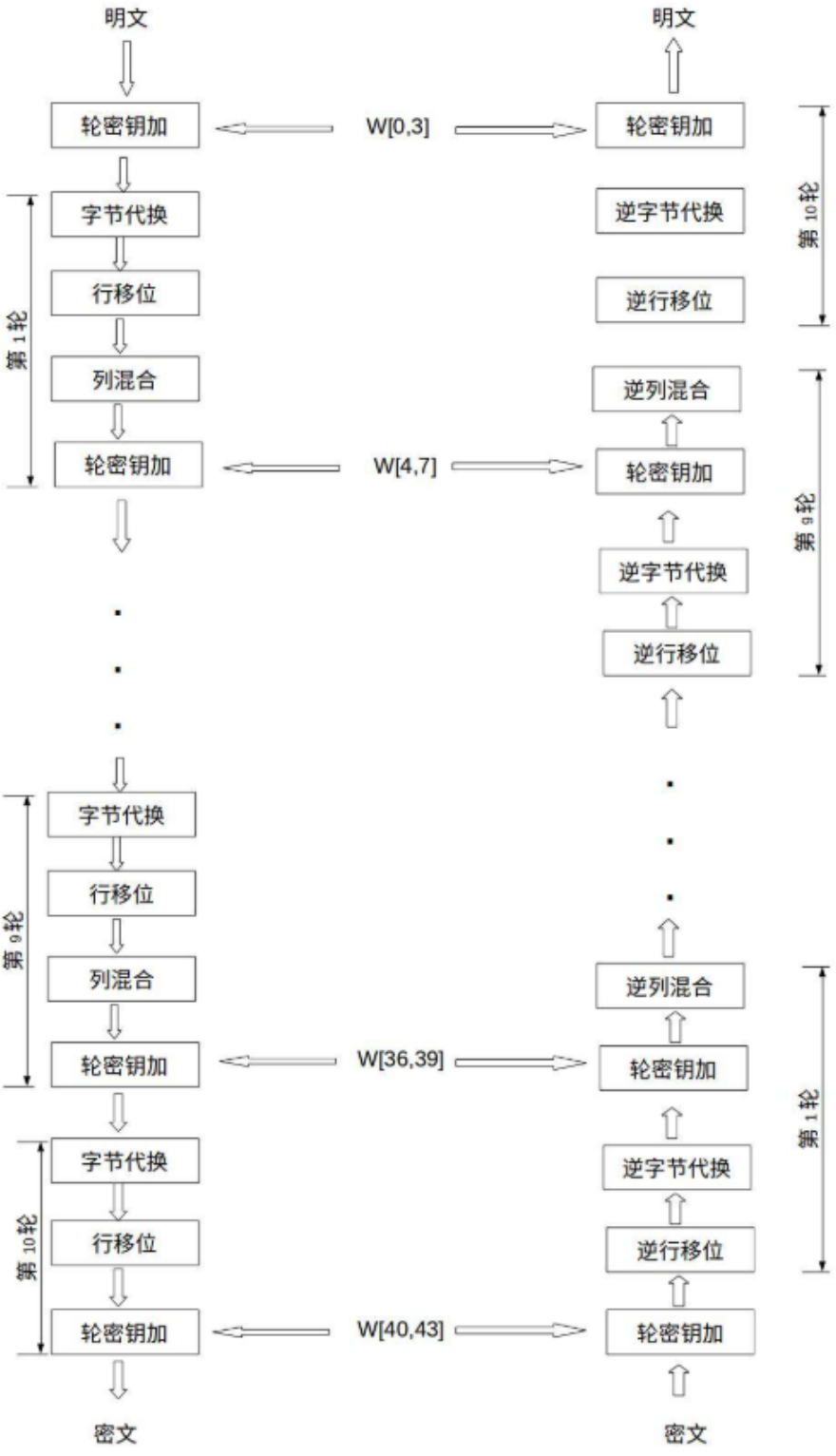
字节替换，就是一个查表的过程，AES 定义了一个 S 和和一个逆 S 盒。替换的规则是：把该字节的高 4 位作为行值，低 4 位作为列值，取出 S 盒或者逆 S 盒中对应的行的元素作为输出。

行移位，就是状态矩阵的第一行不移动，第二行循环左移一位，第三行左移 2 位，第四行左移 3 位。

列混合变换是通过矩阵相乘来实现的，经行移位后的状态矩阵与固定的矩阵相乘，得到混淆后的状态矩阵。使用的乘法和加法是 GF（2^8）上定义的乘法

和加法。

轮密钥加是将 128 位轮密钥 Ki 同状态矩阵中的数据进行逐位异或操作。加解密的过程如图：



三、算法实现

（1）对整个文本进行加密

```
    def onEncryptionClicked(self):
        text = self.textEdit_1.toPlainText()
        if text and len(self.subKey):
            ints = self.string2int(text)
            if len(ints) % 16 != 0:
                ints.extend([0]*(16-len(ints)%16))
            blocks = [ints[i: i+16] for i in range(0, len(ints), 16)]#分
成 128 位的块
            result = []
            for i in range(len(blocks)):
                result.extend(self.encryption(blocks[i]))#字节列表
            enText = self.int2hex(result)
            self.textBrowser_1.setText(enText)
```

（2）对每一个分组进行加密

```
def encryption(self, block):
        block = (np.array(block) ^ np.array(self.subKey[0])).tolist()#初
始的轮密钥加
        for i in range(10):
            s_block = self.s_premutate(block)#16 字节
            s_block_piece = [s_block[j:j+4] for j in range(0,
len(s_block), 4)]
            for j in range(4):#行移位
                for k in range(j):
                    s_block_piece[j].append(s_block_piece[j].pop(0))
            #列混淆
            s_block_piece = np.array(s_block_piece)#s_block_piece 是矩阵形
式
            if i != 9:
                s_block_piece = self.colConfusion(s_block_piece,
self.ConfusionMat)
            s_block_piece = s_block_piece.reshape(1,
s_block_piece.size)[0]#矩阵形式 reshape
            block = (s_block_piece ^ np.array(self.subKey[i+1])).tolist()
        return block
```

（3）产生轮密钥

```
    def saveKey(self):
        key = self.key.text()
        if key:
```

```
            self.subKey.clear()
            initKey = self.string2int(key)
            if len(initKey) > 16:
                initKey = initKey[:16]
            elif len(initKey) < 16:
                initKey.extend([0] * (16 - len(initKey)))
            self.subKey.append(initKey)

            # i+1 #每一轮的 key 是是个独立的 list
            for i in range(10):
                preKey = self.subKey[i]#preKey 是 16 个字节
                curKey = []#也应该是 16 个字节
                for j in range(4):
                    if j == 0:
                        t_preKey_4th = self.T(preKey[12:16], i)#是一个字，4
字节

curKey.extend((np.array(preKey[0:4])^np.array(t_preKey_4th)).tolist())
                    else:#字与字异或得到下一个字
                        xor_result = (np.array(curKey[(j-
1)*4:j*4])^np.array(preKey[j*4:(j+1)*4])).tolist()
                        curKey.extend(xor_result)
                self.subKey.append(curKey)
```

(4) 解密真个文本

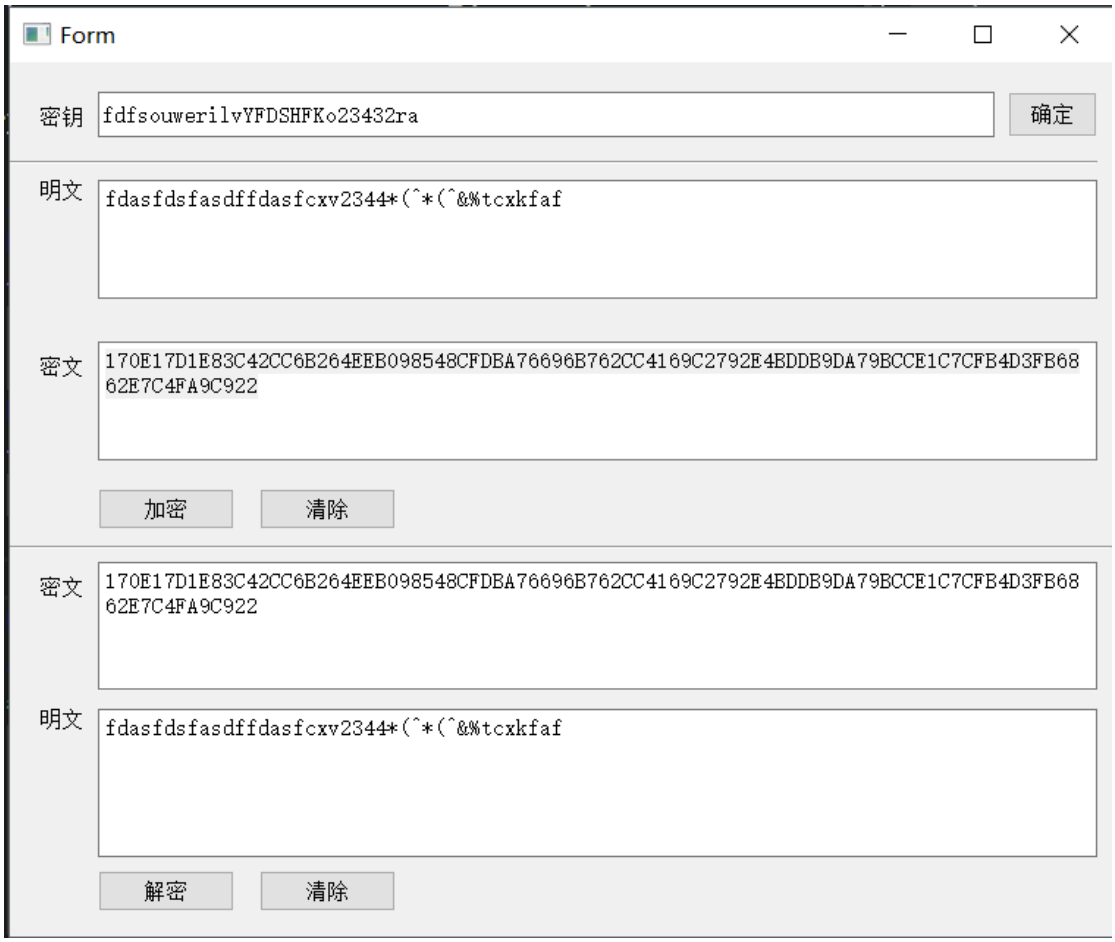
```
    def onDecryptionClicked(self):
        enText = self.textEdit_2.toPlainText()
        if enText and len(self.subKey):
            enText = str.upper(enText)
            if len(enText) % 32 != 0:
                enText += '0' * 5(32 - len(enText)%32)
            ints = self.hex2int(enText)

            blocks = [ints[i:i+16] for i in range(0, len(ints), 16)]
            result = []
            for block in blocks:
                result.extend(self.decryption(block))
            plainText = self.int2string(result)
            self.textBrowser_2.setText(plainText)
```

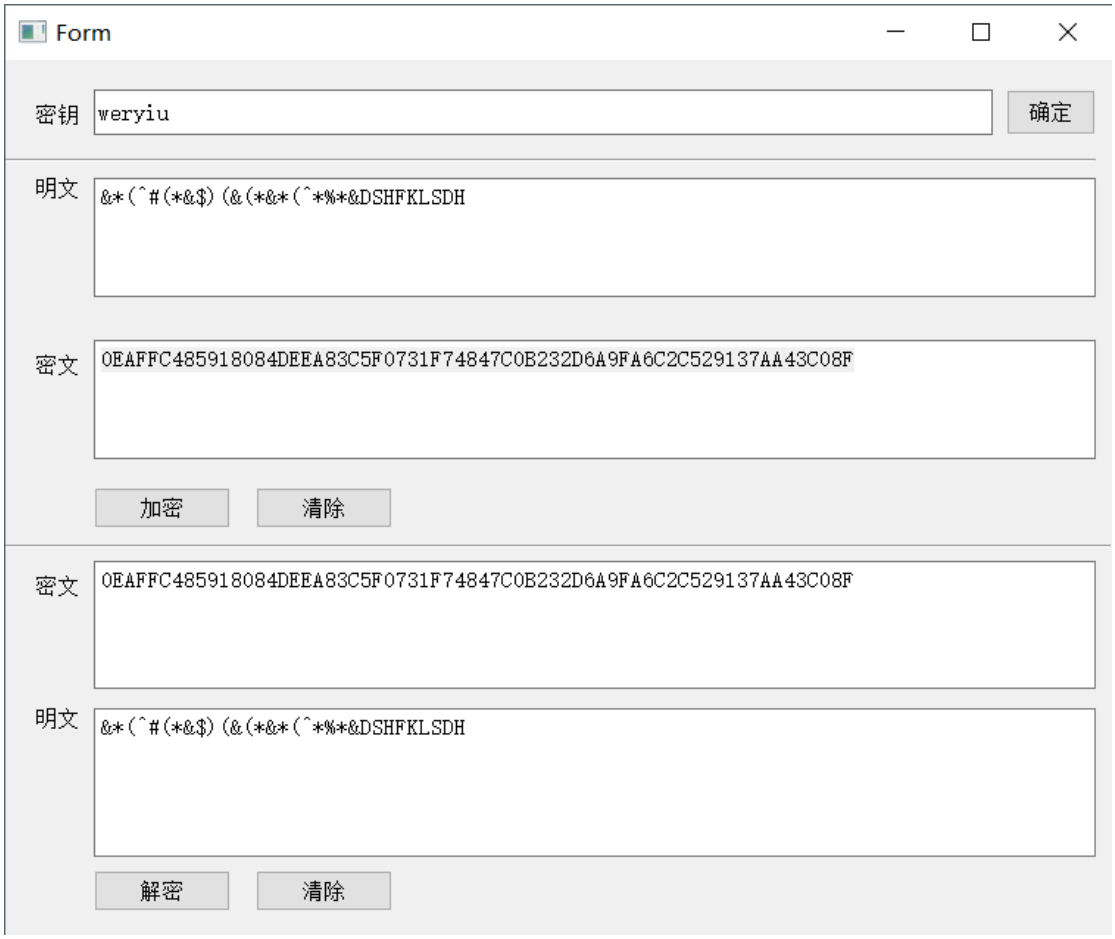
(5) 解密一个分组

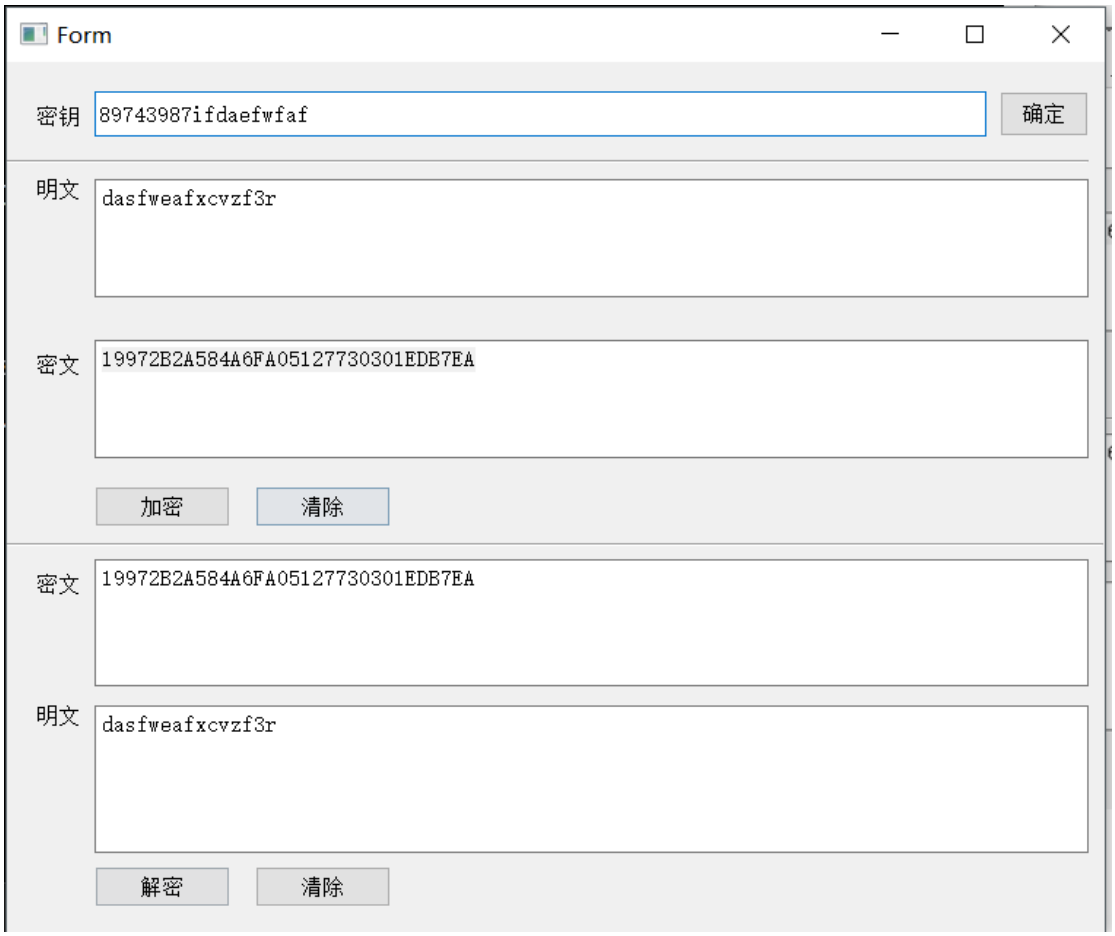
```
    def decryption(self, block):
        block = (np.array(block) ^ np.array(self.subKey[10])).tolist()#
初始的轮密钥加
```

```
        for i in range(9, -1, -1):
            block_piece = [block[j:j+4] for j in range(0, len(block), 4)]
            for j in range(4):#逆向行移位
                for k in range(j):
                    block_piece[j].insert(0, block_piece[j].pop())
            block = []
            for piece in block_piece:
                block.extend(piece)
            s_block = self.s_premutate(block, type=2)#16 字节#逆向字节替代
            s_block = (np.array(s_block) ^
np.array(self.subKey[i])).tolist()#轮密相加
            block = s_block
            if i != 0:
            #列混淆
                s_block_piece = [s_block[j:j+4] for j in range(0,
len(block), 4)]
                s_block_piece = np.array(s_block_piece)#s_block_piece 是矩
阵形式
                s_block_piece = self.colConfusion(s_block_piece,
self.invConfusionMat)
                s_block_piece = (s_block_piece.reshape(1,
s_block_piece.size)[0]).tolist()#矩阵形式 reshape
                block = s_block_piece
        return block
```

四、运行结果


Form
密钥
fdfsouwerilvYFDSHFKo23432ra
确定
明文
fdasfdsfasdffdasfcxv2344*(^*(^&%tcxkfaf
密文
170E17D1E83C42CC6B264EEB098548CFDBA76696B762CC4169C2792E4BDDB9DA79BCCE1C7CFB4D3FB6862E7C4FA9C922
加密
清除
密文
170E17D1E83C42CC6B264EEB098548CFDBA76696B762CC4169C2792E4BDDB9DA79BCCE1C7CFB4D3FB6862E7C4FA9C922
明文
fdasfdsfasdffdasfcxv2344*(^*(^&%tcxkfaf
解密
清除


Form
密钥
weryiu
确定
明文
&*(^#(*&$)(&(*&*(^*%*&DSHFKLSDH
密文
0EAFFC485918084DEEA83C5F0731F74847C0B232D6A9FA6C2C529137AA43C08F
加密
清除
密文
0EAFFC485918084DEEA83C5F0731F74847C0B232D6A9FA6C2C529137AA43C08F
明文
&*(^#(*&$)(&(*&*(^*%*&DSHFKLSDH
解密
清除


Form
密钥
89743987ifdaefwfaf
确定
明文
dasfweafxcvzf3r
密文
19972B2A584A6FA05127730301EDB7EA
加密
清除
密文
19972B2A584A6FA05127730301EDB7EA
明文
dasfweafxcvzf3r
解密
清除