

# Second assignment Description

## 1. Theoretical basis

In this assignment we need to implement the two mode which called CBC mode and CTR mode when using ASE encrypt. However when using library of Crypto in python, we need to assign the AES\_MODE to the init function. So that, in order to implement two mode by ourself, **i decide to pass ECB\_MODE for each blocks**, in which method, it is just like we only call the basis AES algorithm with mode, and then finish the mode by myself, There is the basic though below can describe my design in my code

- CBC\_MODE :

In this mode, we need to extract the IV in the cipher text ,and do the following step:

```
IV XOR decrypte(C[0], k) = m[0]
c[0] XOR decrypte(C[1], k) = m[1]
c[1] XOR decrypte(C[2], k) = m[2]
...
```

By using this step, we can design the algorithm, and finish it

- CTR\_MODE :

in this mode, we also need to extract the IV, and operator IV to increase 1 every roll

```
c[0] XOR encrypte(IV, k) = m[0]
c[1] XOR encrypte(IV+1, k) = m[1]
c[2] XOR encrypte(IV+2, k) = m[2]
...
```

this is the core design though in my code, and knowing the step we can get the result easily

## 2.code

In my project, i use the oop design method, and divide my project into two file,

- my\_decrypt.py
- Main\_for\_AES.py

my\_decrypt.py is used to finish the core algorithm

main\_for\_AES.py is used to be as a entrance to new a class and just pass the key and ciphertext, then we can get the message

In overall angle, i divide my program into ### parts, next i will introduce them respectively.

### 2.1 part #1 my\_decrypt's init fuction

I define a decrypt class called MyDecrypt, and this class is used to implement the decrypt algorithm, and in the init function show below

```
class MyDecrypt:
    def __init__(self, key, cipher_text, IV_len = 16):
        """CBC模式的下的AES
        此时知道 key
        知道 cipher_text
        要求以CBC格式的解码
        IV_len为在密文首添加的IV长度，单位为byte
        """
        self.key = key
        self.byte_key = bytes.fromhex(key)
        self.AES_mode = AES.MODE_ECB
        self.cipher_test = cipher_text
        # 加密
        self.AES = AES.new(self.byte_key, self.AES_mode)
        self.key_len = len(key)
        self.IV_len = IV_len
        # 以十六进制表示的时候，1byte = 2 位十六进制
        self.IV_len_hex = self.IV_len * 2
        self.IV_text = self.cipher_test[:self.IV_len_hex]
        # real_text为不加IV的密文部分
        self.real_text = self.cipher_test[self.IV_len_hex:]
        # block_times为密文分块的数量
        # self.block_times = int(len(self.real_text) / len(self.key))
        if float(len(self.real_text)) / len(self.key) >
int(len(self.real_text) / len(self.key)):
            self.block_times = int(len(self.real_text) / len(self.key)) + 1
        else:
            self.block_times = int(len(self.real_text) / len(self.key))
        self.blocks = []
        #加密
        self.aes = AES.new(str.encode(self.key), AES.MODE_ECB)
        # 一个block的长度等于key的长度
        for i in range(0, self.block_times):
            temp_block = self.real_text[i * self.key_len: (i+1) *
self.key_len]
            self.blocks.append(temp_block)
```

in this init function, we get two pass parameters called **key** and **cipher\_text**, then we should do something based on this two parameters.

Firstly, we need call the lib of crypto and import the AES class to generate an AES object by using new function. However in AES.new() function, the type of str cannot be passed, which means that **we need to transfer the key into the type of bytes** by using the function bytes.fromhex(key), which can transfer the hex in string form into the hex in bytes form, in which type we can use as the pass parameters to construct a AES object. Secondly. we need to extract IV from the cipher text, and devide the cipher text into IV and the read text. then we

should stored the read text into a datastruct called list by the same length as key (in this project is 16 bytes), and we can get several blocks with the same length

Till now, we complete the process of dealing with the data into which we can use in our later function

## 2.2 part #2 XOR in String level

in this part is order to accomplish two functon, one is we need to XOR two hex string such as

```
str1 = "abc"
str2 = "000"
printf(function_one(str1,str2)) == "abc"
```

which means that we can XOR two hex number in string level

another is to transform the str hex into ASCII character, which can make our XOR resualt readable, here are the two functon below:

```
@staticmethod
def XOR_Two_String(StringOne, StringTwo):
    """异或两段字符串,返回十六进制的字符串结果"""
    length1 = len(StringOne)
    length2 = len(StringTwo)
    if length1 > length2:
        length = length2
    else:
        length = length1
    res = ""
    index = 0
    while index < length:
        res += MyDecrypt.XOR_Two_Bit(StringOne[index],
StringTwo[index])
        index += 1
    return res

@staticmethod
def Hex_to_readable(StringText):
    """传入一个十六进制形式的ASCII码, 返回一段可读的文字"""
    length = len(StringText)
    index = 0
    AsciiString = ""
    tmp = ""
    HexFlag = 0
    while index < length:
        if not HexFlag:
            tmp = ""
            tmp += StringText[index]
            HexFlag += 1
        else:
```

```

        tmp += StringText[index]
        AsciiString += MyDecrypt.TwoBit_To_Hex(tmp)
        HexFlag = 0
        index += 1
    return AsciiString

    @staticmethod
    def Bit_To_Hex(Bit):
        """ 'a'==> 0xA """
        if Bit == 'a':
            return 0xa
        elif Bit == 'b':
            return 0xb
        elif Bit == 'c':
            return 0xc
        elif Bit == 'd':
            return 0xd
        elif Bit == 'e':
            return 0xe
        elif Bit == 'f':
            return 0xf
        else:
            return int(Bit)

    @staticmethod
    def XOR_Two_Bit(BitOne, BitTwo):
        """异或两十六进制位 eg 'a' XOR '2' """
        return hex(MyDecrypt.Bit_To_Hex(BitOne) ^
MyDecrypt.Bit_To_Hex(BitTwo))[2:]

    @staticmethod
    def TwoBit_To_Hex(stringBit):
        """将stringbit转为十六进制 'ab'==>0xab """
        FirstBit = stringBit[0]
        SecondBit = stringBit[1]
        firstHex = MyDecrypt.Bit_To_Hex(FirstBit) * 0x10
        SecondBit = MyDecrypt.Bit_To_Hex(SecondBit)
        return chr(firstHex+SecondBit)

```

## 2.2 part #3 decrypt based on CBC mode

here is the core code of decryption in CBC mode

```

def get_CBC_message(self):
    """以CBC的方式解密
    IV XOR C[]
    """

```

```

message = ""
index = 0
xor_text = self.IV_text
while index < self.block_times:
    # print(xor_text)
    # print(self.ASE_TO_BE_FINSHED(self.blocks[index]))
    message +=
self.XOR_Two_String(self.ASE_TO_BE_FINSHED_CBC(self.blocks[index]),
xor_text)
    xor_text = self.blocks[index]
    index += 1
return self.Hex_to_readable(message)

def ASE_TO_BE_FINSHED_CBC(self, message):
    """
    此为待实现的AES加密函数
    传入一个key, message
    此时传入的message为str类型的, 需要以byte的格式进行解密
    且解密之后返回一个十六进制类型的str
    返回一个加密的映射
    """
    decrypte_message = self.AES.decrypt(bytes.fromhex(message)).hex()
    return decrypte_message

```

we define function get\_CBC\_message() to simulate the CBC\_mode, we can see that the core code is

```

message +=
self.XOR_Two_String(self.ASE_TO_BE_FINSHED_CBC(self.blocks[index]),
xor_text)

```

it accomplishes that  $\text{XOR}(F(k, c[i]), c[i-1])$  ( $c[-1] = \text{IV}$ )

code : `self.ASE_TO_BE_FINSHED_CBC(self.blocks[index])` pass a string a the cipher text's block which has the same length of key, and return a string which is the decryption of AES, **furthermore the return value is the type of string of hex(not bytes)**, then use the static method `XOR_Two_String(str1, str2)`, we can XOR two hex in string types and get a string types of the resualt hex. **finally, we use the function `Hex_to_readable()`, which can transfer the string type of hex into it corresponds ACSII character.**

## 2.4 part#4 decrypt based on CTR mode

```

def get_CTR_message(self):
    """
    以CTR的方式进行解密
    IV 隔一个加一
    :return:

```

```

    """
    message = ""
    index = 0
    Xor_text = self.IV_text
    while index < self.block_times:
        message +=
self.XOR_Two_String(self.ASE_TO_BE_FINSHEDED_CTR(Xor_text),
self.blocks[index])
        Xor_text = self.increasement_IV(Xor_text)
        index += 1
    return self.Hex_to_readable(message)

def ASE_TO_BE_FINSHEDED_CTR(self, message):
    """
    CTR模式为对message进行加密操作，而非解密
    :param message:
    :return:
    """

    decrypte_message = self.AES.encrypt(bytes.fromhex(message)).hex()
    return decrypte_message

@staticmethod
def increasement_IV(string):
    """
    :param string: 十六进制的字符串
    :return: 该字符串加一之后的结果
    """

    length = len(string)
    index = length - 1
    while index >= 0:
        if string[index] != 'f':
            #直接返回加一的结果
            tmp_list = list(string)
            tmp_list[index] = hex(MyDecrypt.Bit_To_Hex(string[index]) +
0x1)[2:]

            string = ''.join(tmp_list)
            return string
        else:
            tmp_list = list(string)
            tmp_list[index] = hex(MyDecrypt.Bit_To_Hex(string[index]) +
0x1)[2:]

            string = ''.join(tmp_list)
            index -= 1
    return string

```

in this part i accomplish the CTR mode's decryption, due to the feature of CTR mode, we need to increase IV every block.so we add function increasement\_IV() compared to CBC mode, **function increasement\_IV(IV) is used to increase a hex form string IV into IV+1.** comparing with the CBC mode , CTR mode just change the XOR method, From XOR(F(c[i],k), c[i-1]) into

XOR(F[IV+L, k], c[L]), so the core code becomes below

```
message += self.XOR_Two_String(self.ASE_TO_BE_FINSHED_CTR(Xor_text),
self.blocks[index])
```

## part #5 main\_file

In this file we call the MyDecrypt as a decrypt class, and get the result

```
from my_decrypt import MyDecrypt

# 第一段CBC
key_one = "140b41b22a29beb4061bda66b6747e14"
cipher_text_one = '4ca0ff4c898d61e1edbf1800618fb2828a226d160dad07883d04e0' \
\
'08a7897ee2e4b7465d5290d0c0e6c6822236e1daafb94ffe0c5da05d9476be028ad7c1d81' \
,
the_CBC_one = MyDecrypt(key_one, cipher_text_one)
print(the_CBC_one.get_CBC_message())

# 第二段CBC
key_two = "140b41b22a29beb4061bda66b6747e14"
cipher_text_two = "5b68629feb8606f9a6667670b75b38a5b4832d0f26e1ab7da33249" \
\
"de7d4afc48e713ac646ace36e872ad5fb8a512428a6e21364b0c374df45503473c5242a253"
the_CBC_two = MyDecrypt(key_two, cipher_text_two)
print(the_CBC_two.get_CBC_message())

# 第三段CTR
key_three = "36f18357be4dbd77f050515c73fcf9f2"
cipher_text_three =
"69dda8455c7dd4254bf353b773304eec0ec7702330098ce7f7520d1" \
\
"cbbb20fc388d1b0adb5054dbd7370849dbf0b88d393f252e764f1f5f7ad97ef79d59ce29f5f51eeca32eabedd9afa9329"
the_CBC_three = MyDecrypt(key_three, cipher_text_three)
print(the_CBC_three.get_CTR_message())

# 第四段CTR
key_four = "36f18357be4dbd77f050515c73fcf9f2"
```

```

cipher_text_four =
"770b80259ec33beb2561358a9f2dc617e46218c0a53cbeca695ae45faa8952aa0e311bde9d
4e01726d3184c34451"
the_CBC_four = MyDecrypt(key_four, cipher_text_four)
print(the_CBC_four.get_CTR_message())

```

Seeing from the code, we can know that just need to pass two information called key and cipher\_text and call different method, then we can get the message(result)

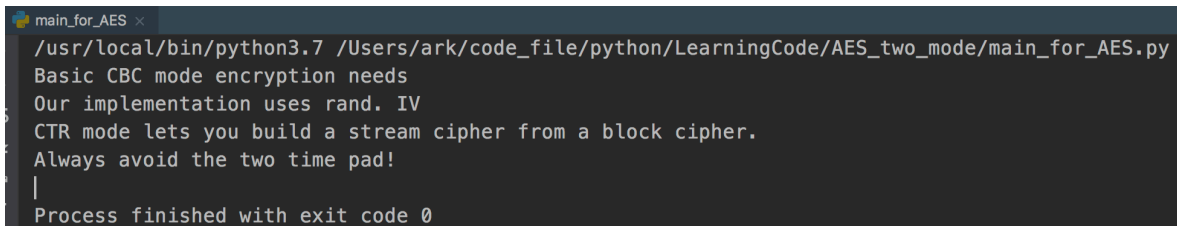
### 3. result

```

#CBC
m1 = "Basic CBC mode encryption needs "
m2 = "Our implementation uses rand. IV"

#CTR
m3 = "CTR mode lets you build a stream cipher from a block cipher."
m4 = "Always avoid the two time pad!"

```



```

main_for_AES x
/usr/local/bin/python3.7 /Users/ark/code_file/python/LearningCode/AES_two_mode/main_for_AES.py
Basic CBC mode encryption needs
Our implementation uses rand. IV
CTR mode lets you build a stream cipher from a block cipher.
Always avoid the two time pad!
|
Process finished with exit code 0

```

### 4. conclusion

In this project, I learn how to implement the two mode of AES which is CBC\_MODE and CTR\_MODE, and get a deep understanding about how it works of AES encryption. Furthermore we use the knowledge which we learn in the class, and apply it into the program, which can increase our skill in programming and also give us second thinking about the knowledge, and give large improvement about our ability!



