CTF Crypto中涉及的AES题目

原创

M3ng@L 于 2022-04-20 17:04:47 发布 195 收藏

分类专栏： CTF比赛复现 密码学知识总结 文章标签： Crypto AES CBC模式 ECB模式 CTR模式

本文链接：https://blog.csdn.net/qq_51999772/article/details/124301715

CTF比赛复现 同时被 2 个专栏收录

31 篇文章 0 订阅

订阅专栏

密码学知识总结

18 篇文章 2 订阅

订阅专栏

CTF Crypto中涉及的AES题目

单独涉及AES_ECB模式
单独涉及AES_CBC模式
Problem
Analysis
Solving code
AES_CBC与AES_ECB的联合求解
Problem
Analysis
Solving Code
单独涉及AES_CTR模式
Problem
Analysis
Solving code
Conclusion

keywords: `AES_CBC模式`， `AES_ECB模式`， `AES_CTR模式`

单独涉及AES_ECB模式

最常见的是`RSA`中套了一个简单的`AES_ECB`加密（`AES`加密`flag`），`RSA`加密的是`AES_ECB`模式的密钥`key`，解`RSA`得到密钥`key`进而求得`flag`

单独涉及AES_CBC模式

来源于陇原战疫2021网络安全大赛 `Civet cat for Prince`

keywords: `AES_CBC`， `xor`， `狸猫换太子`

Problem

```
from Crypto.Cipher import AES
import os
from hashlib import sha256
import socketserver
import signal
import string
import random

table = string.ascii_letters + string.digits
BANNER = br'''
 .d8888b.  d8b                   888                            888
d88P  Y88b Y8P                   888                            888
888    888                       888                            888
888        888 888  888  .d88b.  888888        .d8888b  8888b.  888888
888        888 888  888 d8P  Y8b 888          d88P"        "88b 888
888    888 888 Y88  88P 88888888 888          888      .d888888 888
Y88b  d88P 888  Y8bd8P  Y8b.     Y88b.        Y88b.    888  888 Y88b.
 "Y8888P"  888   Y88P    "Y8888   "Y888        "Y8888P "Y888888  "Y888
```

```
 .d888                         8888888b.          d8b
d88P"                          888   Y88b         Y8P
888                            888    888
888888  .d88b.  888d888        888   d88P 888d888 888 88888b.   .d8888b  .d88b.
888    d88""88b 888P"          8888888P"  888P"   888 888 "88b d88P"    d8P  Y8b
888    888  888 888            888        888     888 888  888 888      88888888
888    Y88..88P 888            888        888     888 888  888 Y88b.    Y8b.
888     "Y88P"  888            888        888     888 888  888  "Y8888P  "Y8888
'''

guard_menu = br'''
1.Tell the guard my name
2.Go away
'''

cat_menu = br'''1.getpermission
2.getmessage
3.say Goodbye
'''



def Pad(msg):
    return msg + os.urandom((16 - len(msg) % 16) % 16)


class Task(socketserver.BaseRequestHandler):
    def _recvall(self):
        BUFF_SIZE = 2048
        data = b''
        while True:
            part = self.request.recv(BUFF_SIZE)
            data += part
            if len(part) < BUFF_SIZE:
                break
        return data.strip()

    def send(self, msg, newline=True):
        try:
            if newline:
                msg += b'\n'
            self.request.sendall(msg)
        except:
            pass

    def recv(self, prompt=b'[-] '):
        self.send(prompt, newline=False)
        return self._recvall()

    def proof_of_work(self):
        proof = (''.join([random.choice(table) for _ in range(12)])).encode()
        sha = sha256(proof).hexdigest().encode()
        self.send(b"[+] sha256(XXXX+" + proof[4:] + b") == " + sha)
        XXXX = self.recv(prompt=b'[+] Give Me XXXX :')
        if len(XXXX) != 4 or sha256(XXXX + proof[4:]).hexdigest().encode() != sha:
            return False
        return True

    def register(self):
        self.send(b'')
```

```
        self.send(b" ")
        username = self.recv()
        return username

    def getpermission(self, name, iv, key):
        aes = AES.new(key, AES.MODE_CBC, iv)
        plain = Pad(name)+b"a_cat_permission"
        return aes.encrypt(plain)

    def getmessage(self, iv, key, permission):
        aes = AES.new(key, AES.MODE_CBC, iv)
        return aes.decrypt(permission)

    def handle(self):
        signal.alarm(50)
        if not self.proof_of_work():
            return
        self.send(BANNER, newline=False)
        self.key = os.urandom(16)
        self.iv = os.urandom(16)
        self.send(b"I'm the guard, responsible for protecting the prince's safety.")
        self.send(b"You shall not pass, unless you have the permission of the prince.")
        self.send(b"You have two choices now. Tell me who you are or leave now!")
        self.send(guard_menu, newline=False)
        option = self.recv()
        if option == b'1':
            try:
                self.name = self.register()
                self.send(b"Hello " + self.name)
                self.send(b"Nice to meet you. But I can't let you pass. I can give you a cat. She will play with
you")
                self.send(b'Miao~ ' + self.iv)
                for i in range(3):
                    self.send(b"I'm a magic cat. What can I help you")
                    self.send(cat_menu, newline=False)
                    op = self.recv()
                    if op == b'1':
                        self.send(b"Looks like you want permission. Here you are~")
                        permission = self.getpermission(self.name, self.iv, self.key)
                        self.send(b"Permission:" + permission)
                    elif op == b'2':
                        self.send(b"Looks like you want to know something. Give me your permission:")
                        permission = self.recv()
                        self.send(b"Miao~ ")
                        iv = self.recv()
                        plain = self.getmessage(iv, self.key, permission)
                        self.send(b"The message is " + plain)
                    elif op == b'3':
                        self.send(b"I'm leaving. Bye~")
                        break
                self.send(b"Oh, you're here again. Let me check your permission.")
                self.send(b"Give me your permission:")
                cipher = self.recv()
                self.send(b"What's the cat tell you?")
                iv = self.recv()
                plain = self.getmessage(iv, self.key, cipher)
                prs, uid = plain[16:],plain[:16]
                if prs != b'Princepermission' or uid != self.name:
                    self.send(b"You don't have the Prince Permission. Go away!")
                    return
```

```
                else:
                    self.send(b"Unbelievable! How did you get it!")
                    self.send(b"The prince asked me to tell you this:")
                    f = open('flag.txt', 'rb')
                    flag = f.read()
                    f.close()
                    self.send(flag)
            except:
                self.request.close()
        if option == b'2':
            self.send(b"Stay away from here!")
        self.request.close()


class ThreadedServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    pass


class ForkedServer(socketserver.ForkingMixIn, socketserver.TCPServer):
    pass


if __name__ == "__main__":
    HOST, PORT = '0.0.0.0', 10005
    print("HOST:POST " + HOST + ":" + str(PORT))
    server = ForkedServer((HOST, PORT), Task)
    server.allow_reuse_address = True
    server.serve_forever()
```

Analysis

首先是执行`proof of work`函数，简单的爆破四个字符使得其连接上剩余已知字符的`sha256`等于函数给出的`sha256`值

然后进入正题，第一段守卫处没有有用信息，需要输入你的`name`（之后称为`m1`），然后在`magic cat`处可以获得

- `getpermission()`

得到对`m1+b'a_cat_permission'`的`AES_CBC`加密之后得到的密文，其中`iv`是系统生成的已知量，`key`未知

```
    def getpermission(self, name, iv, key):
        aes = AES.new(key, AES.MODE_CBC, iv)
        plain = Pad(name)+b"a_cat_permission"
        return aes.encrypt(plain)
```

- `getmessage()`

输入一段密文进行`AES_CBC`解密，其中`iv`由我们给出，`key`是之前使用的，得到解密之后给出的明文

```
    def getmessage(self, iv, key, permission):
        aes = AES.new(key, AES.MODE_CBC, iv)
        return aes.decrypt(permission)
```

总共有三次询问的机会（循环次数为3），循环完成之后由守卫进行验证

提交给守卫某个`密文`和自己给出的偏移量`iv`，使得进行相同的`key`的AES_CBC解密之后得到的明文是`m1+b'Princeperimission'`

```
self.send(b"Give me your permission:")
cipher = self.recv()
self.send(b"What's the cat tell you?")
iv = self.recv()
plain = self.getmessage(iv, self.key, cipher)
prs, uid = plain[16:],plain[:16]
if prs != b'Princepermission' or uid != self.name:
    ...
```

以上就是对题目代码的分析

简单来说，我们需要想办法把从`magic cat`处得到的`perimission = m1 + b'a_cat_perimission'`的密文换成`m1 + b'Princepermission'`的密文

其中可以用到的是`magic cat`会提供一次或两次`解密`的机会（用自己给出的`iv`）

显然没有办法知道`key`进而直接求解，但是我们不需要老老实实地直接找
$need\_m + {}_{need\_m}$ 的密文，因为我们可以构造合适的`iv`代入最后的AES_CBC解密

先令

$m = input\_name$

$m = a\_cat\_permission$

$need\_m = {}_{m}$

$need\_m = Princepermission$

由于AES_CBC加密是分块加密，而每一块的大小是`128bits`也就是`16字节`，而由于题目要求，${}_{m}$ 和${}_{m}$ 以及$need_m$ 都是16字节大小的

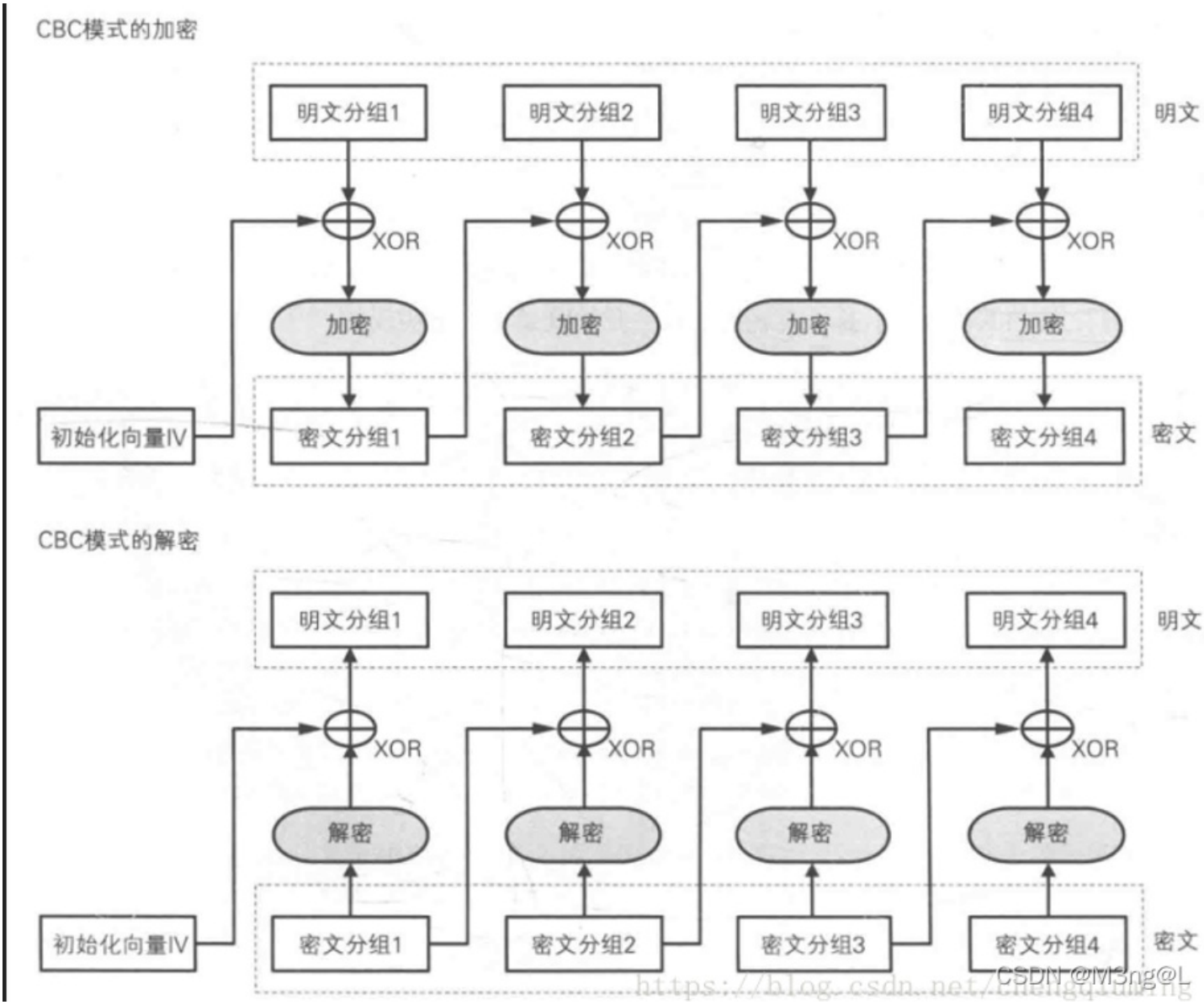
这就意味着加密得到的密文块会有特殊的性质

令

通过图片明确一下AES_CBC加解密过程

加密过程是，先将明文按`16字节`一分组拆开，然后进行如图加密（先进行异或，非明文分组1的异或对象是前一个密文分组或者是明文分组1的异或对象初始化向量`iv`），再分别投入加密器，得到各个密文分组后拼接起来成为最后的密文

解密过程是，将密文按`16字节`一分组拆开，然后分别投入解密器，得到的结果与前一个密文分组或者初始化向量进行异或，最后得到各个明文分组再直接拼接在一起组成最后的明文



其中加密器和解密器在本题算是**黑盒**，只需要知道进去的数据和原来是不一样的即可

我们构造的密文要先经过一次解密器，这就意味着非经过AES_CBC加密产生的密文（比如 c ,c ）都会产生一个未知的数据；那么这时就需要用到`magic cat`提供的`解密`的机会

经过解密器之后会进行异或，那么这就是本题的关键之一了，异或的性质就是两个一样的数异或均为0，也就是说对其他一起的数不会有影响，利用这个性质我们构造

$$fake\_c = c \oplus m \oplus$$

使用系统给出的`iv`代入`magic cat`提供的一次解密的机会

先经过解密器得到

$$m \oplus iv \oplus decrypted\_m \oplus$$

在与初始化向量`iv`异或得到

$$m= m \oplus$$

接着构造

$$fake\_iv= m \oplus m \oplus$$

那么最后传入的密文是$fake\ c +$，传入的`iv`也就是$fake_iv$

推导一下最后的解密过程，（AES_CBC是分块解密的）

$fake_c$单独进行解密得到$m \oplus iv \oplus$

然后再与$fake_iv$异或得到

$$m \oplus iv \oplus decrypted\ m \oplus$$

得到明文分组1，符合最后的判断条件

然后c单独进行解密，得到$c \oplus$

再与$fake_c$进行异或得到

$$c \oplus m \oplus m \oplus$$

这就是明文分组2，符合最后的判断条件

Solving code

```
from re import L
from pwn import *
import hashlib,string,random
from Crypto.Cipher import AES

io = remote("node4.buuoj.cn","27370")
temp = io.recvline()
# print(temp)
temp1 = temp.split(b"==")
# print(temp1)
part_proof = bytes.decode(temp1[0].split(b"XXXX")[1])[1:-2]
sha = bytes.decode(temp1[1]).strip()
table = string.ascii_letters + string.digits
while True:
    XXXX = "".join([random.choice(table)for _ in range(4)])
    temp_proof = XXXX + part_proof
    temp_sha = hashlib.sha256(temp_proof.encode()).hexdigest()
    if sha == temp_sha:
        io.recvuntil(b"[+] Give Me XXXX :")
        io.sendline(XXXX.encode())
        break
io.recvuntil(b"[-] ")
io.sendline(b"1")
io.sendline(b"1" * 16)
```

```
io.recvuntil(b'Miao~ ')
iv = io.recvuntil(b"\n")[:-1]
# print(iv)
io.recvuntil(b'[-]')
io.sendline(b'1')
io.recvuntil(b'Permission:')
cat_permission = io.recvline()[:-1]
# print(cat_permission)
io.recvuntil(b"[-]")
io.sendline(b'2')
io.recvuntil(b"Looks like you want to know something. Give me your permission:")

m2 = b"a_cat_permission"
m1 = b'1' * 16 # your name,直接16字节，不会进行pad()函数，如果那样的话，m1也会成为一个未知量
need_m2 = b"Princepermission"
need_m1 = m1
c1 = cat_permission[:16]
c2 = cat_permission[16:]
fake_c1 = xor(xor(c1,m2),need_m2)

io.sendline(fake_c1)
io.recvuntil(b"Miao~ ")
io.sendline(iv)
io.recvuntil(b"The message is ")
m = io.recvuntil(b"\n")[:-1]
# print(plain)
io.recvuntil(b"[-]")
io.sendline(b'3')
io.recvuntil(b"Give me your permission:")

fake_permission = fake_c1 + c2
fake_iv = xor(xor(m,m1),iv)

io.sendline(fake_permission)
io.recvuntil(b"What's the cat tell you?")
io.sendline(fake_iv)
io.interactive()
```

AES_CBC与AES_ECB的联合求解

来源于安洵杯2020 `easyAES`

keywords: `AES_CBC`， `AES_CBC所使用的加解密器和AES_ECB模式所使用的相同`

Problem

```
#!/usr/bin/python
from Crypto.Cipher import AES
import binascii
from Crypto.Util.number import bytes_to_long
from flag import flag
from key import key

iv = flag.strip(b'd0g3{').strip(b'}')

LENGTH = len(key)
assert LENGTH == 16

hint = os.urandom(4) * 8
print(bytes_to_long(hint)^bytes_to_long(key))

msg = b'Welcome to this competition, I hope you can have fun today!!!!!!'

def encrypto(message):
    aes = AES.new(key,AES.MODE_CBC,iv)
    return aes.encrypt(message)

print(binascii.hexlify(encrypto(msg))[-32:])

'''
566312332923254122055287547981339707836332169363020498931302204611391606827777
b'3c976c92aff4095a23e885b195077b66'
'''
```

Analysis

`flag`赋值给了`iv`，用作`AES_CTR`加密模式；其中也用到了`key`，而关于`key`的提示是`key`与一串重复的字节进行异或

`len(key) == 16`，`hint = os.urandom(4) * 8`，很显然两者是不等长的，所以很容易恢复出`key`，因为并没有都异或；也可以这样来验证

```
>>> from Crypto.Util.number import *
>>> hint_xor_key = 566312332923254122055287547981339707836332169363020498931302204611391606827777
>>> long_to_bytes(hint_xor_key)
b'}4$d}4$d}4$d}4$d\x19\x04CW\x06CA\x08\x1e[I\x01\x04[Q\x19'
```

很显然发现前`4`个字节是重复出现了很多次，这就是`hint`的重复字节

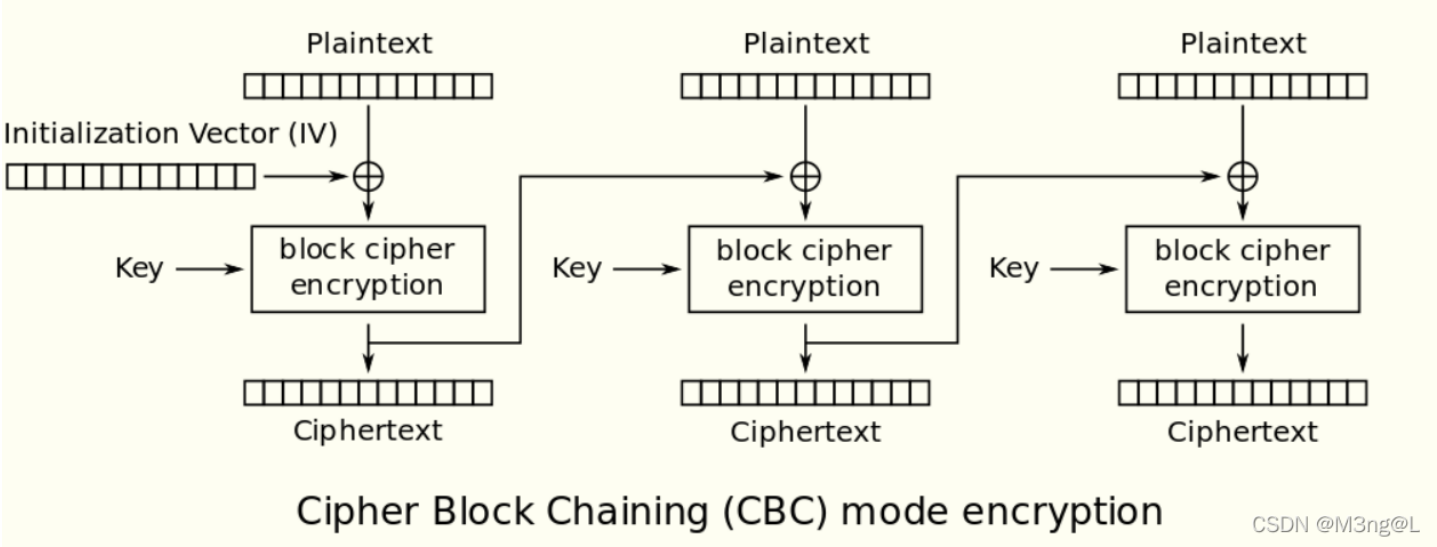
那么

```
hint_xor_key = 566312332923254122055287547981339707836332169363020498931302204611391606827777
hint = long_to_bytes(hint_xor_key)[:4]
key = hint_xor_key ^ bytes_to_long(hint * 8)
key = long_to_bytes(key)
```

看到脚本最后给出的`enc`是`binascii.hexlify(encrypto(msg))[-32:]`，转换回来得到`16`字节长的密文（实际上就是最后一组密文分组）

```
>>> import binascii
>>> part_enc = b'3c976c92aff4095a23e885b195077b66'
>>> part_enc = binascii.unhexlify(part_enc)
>>> part_enc
b'<\x97l\x92\xaf\xf4\tZ#\xe8\x85\xb1\x95\x07{f'
```

现在`key,msg`已知，回到`CBC`模式实现的细节



以上是加密过程，那么反过来，当我们已知整个`msg`和最后一组密文分组，由于**CBC模式使用的加解密器与`ECB`模式的是一样的**，那么我们可以应用`ECB`模式的解密器将最后一组密文分组进行**解密器解密**，得到的结果是**上一个密文分组与最后一个明文分组的异或结果**；由于明文分组已知，与之异或得到上一个密文分组的值，以此类推，我们可以知道所有密文分组和偏移量`iv`（也即是`flag`）

Solving Code

```
from Crypto.Cipher import AES
from Crypto.Util.number import *
from pwn import *
import binascii
hint_xor_key = 566312332923254122055287547981339707836332169363020498931302204611391606827777
part_enc = b'3c976c92aff4095a23e885b195077b66'
msg = b'Welcome to this competition, I hope you can have fun today!!!!!!'
# print(long_to_bytes(hint_xor_key))
hint = long_to_bytes(hint_xor_key)[:4]
key = hint_xor_key ^ bytes_to_long(hint * 8)
key = long_to_bytes(key)
# print(len(msg))
aes = AES.new(key,AES.MODE_ECB)
part_enc = binascii.unhexlify(part_enc)
enc1 = part_enc
m1 = aes.decrypt(enc1)
enc2 = xor(msg[-16:],m1)
m2 = aes.decrypt(enc2)
enc3 = xor(msg[-32:-16],m2)
m3 = aes.decrypt(enc3)
enc4 = xor(msg[-48:-32],m3)
m4 = aes.decrypt(enc4)
enc5 = xor(msg[-64:-48],m4)
iv = enc5
print(iv)
```

单独涉及AES_CTR模式

来源于安洵杯2021 `air encryption`

keywords: AES_CTR， xor， 交互， 欧拉定理， 数据处理

Problem

```python
#!/usr/bin/python
import socketserver
import random
import os
import string
import binascii
import hashlib
from Crypto.Cipher import AES
from Crypto.Util import Counter
from Crypto.Util.number import getPrime
from hashlib import sha256
import gmpy2
from flag import flag

def init():
    q = getPrime(512)
    p = getPrime(512)
    e = getPrime(64)
    n = q*p
    phi = (q-1) * (p-1)
    d = gmpy2.invert(e, phi)
    hint = 2 * d + random.randint(0, 2**16) * e * phi
    mac = random.randint(0, 2**64)
    c = pow(mac, e, n)
    counter = random.randint(0, 2**128)
    key = os.urandom(16)
    score = 0
    return n, hint, c, counter, key, mac, score

class task(socketserver.BaseRequestHandler):

    def POW(self):
        random.seed(os.urandom(8))
        proof = ''.join([random.choice(string.ascii_letters+string.digits) for _ in range(20)])
        result = hashlib.sha256(proof.encode('utf-8')).hexdigest()
        self.request.sendall(("sha256(XXXX+%s) == %s\n" % (proof[4:],result)).encode())
        self.request.sendall(b'Give me XXXX:\n')
        x = self.recv()

        if len(x) != 4 or hashlib.sha256((x+proof[4:].encode())).hexdigest() != result:
            return False
        return True

    def recv(self):
        BUFF_SIZE = 2048
        data = b''
        while True:
            part = self.request.recv(BUFF_SIZE)
            data += part
            if len(part) < BUFF_SIZE:
                break
        return data.strip()

    def padding(self, msg):
        return  msg + chr((16 - len(msg)%16)).encode() * (16 - len(msg)%16)
```

```python
    def encrypt(self, msg):
        msg = self.padding(msg)
        if self.r != -1:
            self.r += 1
            aes = AES.new(self.key, AES.MODE_CTR, counter = Counter.new(128, initial_value=self.r))
            return aes.encrypt(msg)
        else:
            return msg

    def send(self, msg, enc=True):
        print(msg, end= '    ')
        if enc:
            msg = self.encrypt(msg)
        print(msg, self.r)
        self.request.sendall(binascii.hexlify(msg) + b'\n')

    def set_key(self, rec):
        if self.mac == int(rec[8:]):
            self.r = self.counter

    def guess_num(self, rec):
        num = random.randint(0, 2**128)
        if num == int(rec[10:]):
            self.send(b'right')
            self.score += 1
        else:
            self.send(b'wrong')

    def get_flag(self, rec):
        assert self.r != -1
        if self.score ==  5:
            self.send(flag, enc=False)
        else:
            self.send(os.urandom(32) +  flag)

    def handle(self):
        self.r = -1

        if not self.POW():
            self.send(b'Error Hash!', enc= False)
            return

        self.n, self.hint, self.c ,self.counter, self.key, self.mac, self.score = init()

        self.send(str(self.n).encode(), enc = False)
        self.send(str(self.hint).encode(), enc = False)
        self.send(str(self.c).encode(), enc = False)

        for _ in range(6):
            rec = self.recv()
            if rec[:8] == b'set key:':
                self.set_key(rec)
            elif rec[:10] == b'guess num:':
                self.guess_num(rec)
            elif rec[:8] == b'get flag':
                self.get_flag(rec)
            else:
                self.send(b'something wrong, check your input')
```

```
class ForkedServer(socketserver.ForkingMixIn, socketserver.TCPServer):
    pass

def main():
    HOST, PORT = '127.0.0.1', 10086
    server = ForkedServer((HOST, PORT), task)
    server.allow_reuse_address = True
    server.serve_forever()

if __name__ == '__main__':
    main()
```

Analysis

经典的密码nc连接题，用了套接字模块，那么重点函数就是`handle()`

```
    def handle(self):
        self.r = -1

        if not self.POW():
            self.send(b'Error Hash!', enc= False)
            return

        self.n, self.hint, self.c ,self.counter, self.key, self.mac, self.score = init()

        self.send(str(self.n).encode(), enc = False)
        self.send(str(self.hint).encode(), enc = False)
        self.send(str(self.c).encode(), enc = False)

        for _ in range(6):
            rec = self.recv()
            if rec[:8] == b'set key:':
                self.set_key(rec)
            elif rec[:10] == b'guess num:':
                self.guess_num(rec)
            elif rec[:8] == b'get flag':
                self.get_flag(rec)
            else:
                self.send(b'something wrong, check your input')
```

先进行`POW`函数验证（相当于拖慢脚本运行速度），就是判断`hash`值，爆破即可

按照`RSA`加密的方式给出了`n,c`和`hint`（注意服务端给出的这几个值是通过字节转十六进制了的，需要转回正常的字节，而原来的字节是十进制数，所以使用`binascii.unhexlify()`），且加密的明文是`mac`；先把ta求出来

$$hint = 2 \cdot d + random$$

再来看到之后提供三种选择，`set key`，`guess num`，`get flag`

分别查看函数内容

```
    def set_key(self, rec):
        if self.mac == int(rec[8:]):
            self.r = self.counter
```

似乎是判定输入的内容是否和`mac`一致，若一致，则进行一个赋值操作，看到是将`counter`赋值给`r`

寻找`counter`的生成

```
def init():
    ...
 counter = random.randint(0, 2**128)
```

就是一个随机数，而`self.r`初始值是`-1`

为什么要进行赋值操作呢，可以在`encrypt()`函数中看到

```
    def encrypt(self, msg):
        msg = self.padding(msg)
        if self.r != -1:
            self.r += 1
            aes = AES.new(self.key, AES.MODE_CTR, counter = Counter.new(128, initial_value=self.r))
            return aes.encrypt(msg)
        else:
            return msg
```

如果`self.r`的值没有改变的话，也就是说没有进行`set key`操作的话，`AES_CRT`加密过程就不会生效；加密不生效就可以直接拿`flag`吗，再来看到`get flag`操作

```
    def get_flag(self, rec):
        assert self.r != -1
        if self.score ==  5:
            self.send(flag, enc=False)
        else:
            self.send(os.urandom(32) +  flag)
```

有个`assert self.r != -1`，意味着必须`self.r`被随机生成的`counter`赋值才能进行`get flag`，否则会直接`assert`报错

三种选择中还剩下一个`guess num`操作

```
    def guess_num(self, rec):
        num = random.randint(0, 2**128)
        if num == int(rec[10:]):
            self.send(b'right')
            self.score += 1
        else:
            self.send(b'wrong')
```

如果猜对随即生成的数字，则使得`self.score += 1`，而`self.score`的作用是在`get flag`中

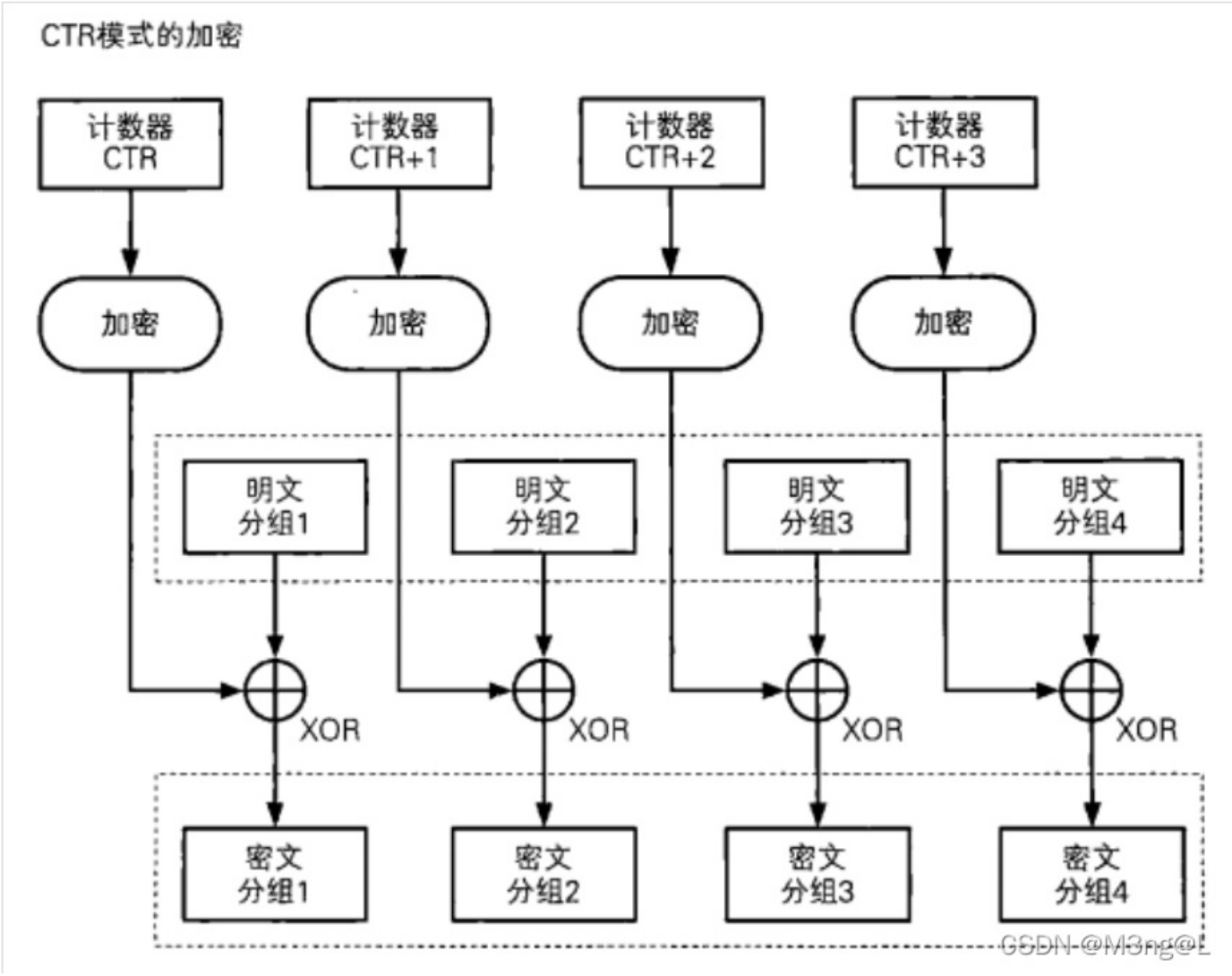
```
        if self.score ==  5:
            self.send(flag, enc=False)
```

意味着连续猜中`5`次则能使得`get flag`操作直接返回`flag`

但是细看一下，首先随机数生成的范围很大，不可能爆破；只能猜测一次，否则会在`6`次选择中浪费一次机会；最重要的是，`6`次选择中有一次机会必须用来`set key`（否则无法进行`get flag`操作），有一次机会用来`get flag`，只剩下`4`次机会，而我们需要连续猜中`5`次随机数；所以`guess num`操作在现在看来就是一个幌子

那么关于能取得`flag`的机会只有落在`encrypt()`函数上，是`AES_CTR`加密，关于这个模式

- 加密过程



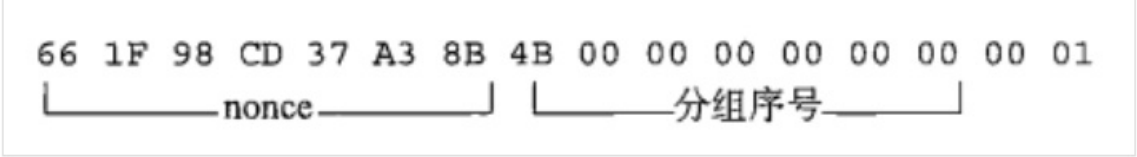
如图所示，引入了一个计数器`CTR`，使得在第二步中与各个明文分组进行异或的**加密流**不同，因为每加密一个明文分组（`16 bytes`），计数器`CTR`就会自动`+1`

题目当中是如何生成`CTR`的

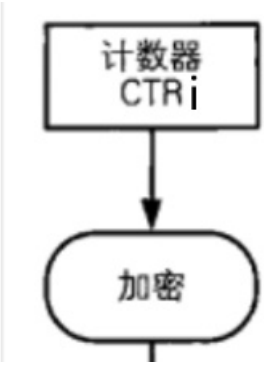
```
from Crypto.Util import Counter
aes = AES.new(self.key, AES.MODE_CTR, counter = Counter.new(128, initial_value=self.r))
```

看得出来是`Counter.new(128, initial_value=self.r))`中`initial_value`变量在设置`CTR`的初始值

而`CTR`所表示的值也不是简单的`00`，`01`，而是一组由**随机数nonce**和**分组序号**组成的初始值

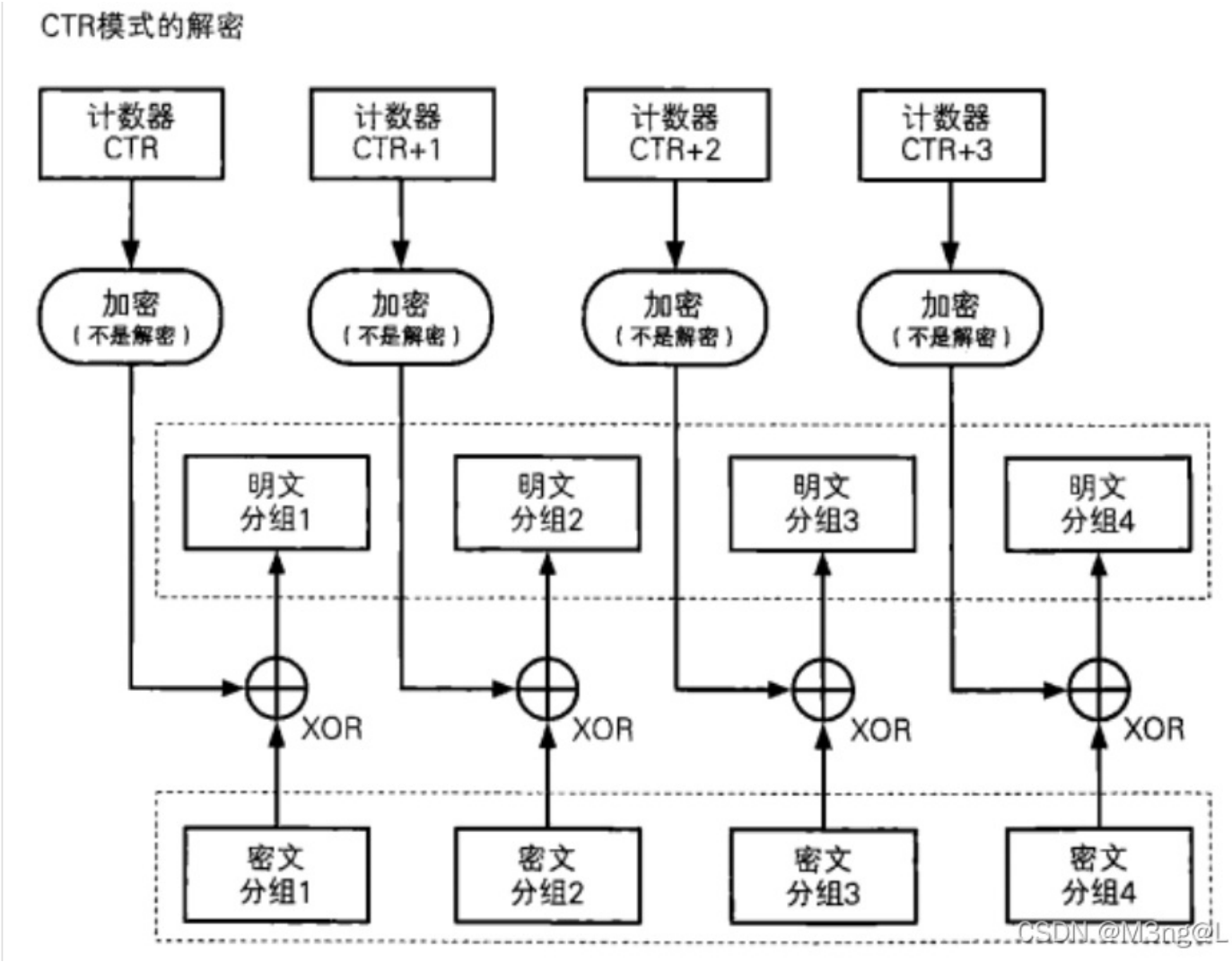


每加密一个明文分组，`CTR`的分组序号则会`+1`，使得进行**加密**操作之后的加密流与之前的加密流完全不同，达到与各个明文分组进行异或的字节串互不相同



PS：这里以及之后提到的**加密流**都是
这一部分的结果；并且**加密操作**特指`CTR`生成后马上进行的加密器的操作

- 解密过程



可以发现就是重置计数器`CTR`使其在对应的密文分组上产生与加密时一样的`CTR`，再将其进行**加密**操作，生成与加密过程中一样的加密流，与密文分组异或即得到明文分组

分析完`AES_CTR`的加解密过程，发现实际上加解密过程都可以分为两大块，`CTR加密`和`异或`；

再回到题目脚本，这里容易发现，在每次服务器脚本`send`的时候，函数中很多地方都是`send(x,enc = False)`，可以找到脚本中自己定义的`send`函数

```
    def send(self, msg, enc=True):
        print(msg, end= '    ')
        if enc:
            msg = self.encrypt(msg)
        print(msg, self.r)
        self.request.sendall(binascii.hexlify(msg) + b'\n')
```

作用就是写出`enc = False`的`send`函数里面，会直接输出内容，也就是正常交互过程中的`send`函数；而当`enc = True`，或者说`send`函数里面没有对`enc`进行再赋值，那么输出的内容会进行`encrypt()`加密，也就是`AES_CTR`加密

仔细观察可以发现

```
        self.send(b'something wrong, check your input')
# 以及
        if num == int(rec[10:]):
            self.send(b'right')
            self.score += 1
        else:
            self.send(b'wrong')
```

这里的`send`函数返回的实际上不是明文`right`，`something wrong...`；而是`AES_CTR`加密后的密文（也可以在本地测试`debug`的时候发现这个特征）

不可能无缘无故地把这些返回内容设置为加密之后再返回，所以这里一定是**突破点**

现在我们可以得到一**对明密文**，`AES_CTR`加密后的`flag`；加密密钥`key`未知也无法知道，显然不能通过正常求密钥`key`来解密

总共给予了`6次`选择的机会，那如果我们多得到几组明密文，有什么用呢

在`AES_CTR`加密中，如果我们已知明文以及对应的密文，将两者异或即可得到**加密流**，因为加密过程中

$$\oplus\ enc\_stream =$$

那么最终的问题就是我们需要哪些加密流，也就是说哪些加密流是加密`flag`的时候真正使用的

加密流当然是越长越好，所以我们使用`something wrong...`作为明密文组求得加密流，默认经过`padding`函数后明文长度为`48`

正常情况下（本题不是这样的正常情况，之后会提到，也是关键点之一）是得到的加密流应该是由`CTR`，`CTR+1`，`CTR+2`生成的，那么很可能不够长足以使得与加密之后的`flag`异或可以得到完整的`flag`（因为`flag`加密过程中是作为`padding(os.urandom(32)+flag)`进行加密的，所以密文会比较长）

那么现在有一个疑惑就是如果连续加密两次及以上的话，`counter`会不会自动继续`+1`，使得我们确实可以将对不同的加密过程（虽然明文确实是相同的）中生成的多个加密流按照`CTR+i`的顺序拼接在一起，作为一整个加密流

实践一下

```
>>> from Crypto.Cipher import AES
>>> from Crypto.Util import Counter
>>> from pwn import *
>>> key = b'\x01' * 16
>>> msg = b"flag1111111111111111111111111111111111111111111111111111111111"
>>> t = Counter.new(128, initial_value=123)
>>> aes = AES.new(key, AES.MODE_CTR, counter = t)
>>> enc1 = aes.encrypt(msg)
>>> enc1
b"0=J\xe6\x87\xec\x07r\xf1{L\x94\xf7\xf2\xfcp|-\xf0\xca\xba\xe4:\x89\x7f\rI\xb5\x84\xb8\x00\xe2%(\x02o\xa5\x8c\x
a3\x93\x18'\xb0\xa2\xe2\xd4]\xb1*\xb5\x8fH7\xd8\xfa\x8f\x08\xe7X"
>>> enc2 = aes.encrypt(msg)
>>> enc2
b'\x97\xf5\xe3_\xdc\xb3\x1c\x11\x1fM\n\x16\x06}:hvU\x1d\x93"\xf6\x89\xc3\x05\x94\x8b>6ha\xce\'\x9f\xb5$\x07Hm\xa
5\xd1]y)P\xff\xd3e\xea\x7f\xa9\xb3\xe9\xcd\x88\x97>\x8d\xad'
```

实践证明确实可以将不同加密过程中得到的加密流拼接在一起，但是如何确保对 `flag` 加密的加密流和我们求得的加密流一致呢，那就是重复使用 `set key` 操作

大致重演一下重复使用 `set key` 的作用

```
>>> from Crypto.Cipher import AES
>>> from Crypto.Util import Counter
>>> from pwn import *
>>> key = b'\x01' * 16
>>> msg = b"flag1111111111111111111111111111111111111111111111111111111111"
>>> t = Counter.new(128, initial_value=123)
>>> aes = AES.new(key, AES.MODE_CTR, counter = t)
>>> enc1 = aes.encrypt(msg)
>>> enc2 = aes.encrypt(msg)
>>> t = Counter.new(128, initial_value=123)
>>> aes = AES.new(key, AES.MODE_CTR, counter = t)
>>> enc3 = aes.encrypt(msg)
>>> enc4 = aes.encrypt(msg)
>>> assert xor(enc1,msg) == xor(enc3,msg)
>>> assert xor(enc2,msg) == xor(enc4,msg)
>>>
```

意思就是当我们重复对 `counter` 进行赋相同的值时，整个 `CTR` 初始值会被刷新

那么我们是不是只需要两到三个 `something wrong...` 明密文对得到的加密流拼接在一起与 `flag` 进行异或就好了呢

实践发现只能得到一半的 `flag`，刚好就是前 `48` 位的 `padding(os.urandom(32)+flag)`；之后的加密流为什么不是正确对应的呢

发现作者在 `self.r` 和 `aes` 上面动了手脚，在每次加密整个明文之前 `self.r` 会自己首先 `+1`，也就是 `CTR + 1`；并且在每次加密整个明文之前都会重新定义一遍 `aes = AES.new(self.key, AES.MODE_CTR, counter = Counter.new(128, initial_value=self.r))`，这就意味着对两次明文（不是明文分组）加密中所使用两个的 `CTR` 生成的加密流中，上一个明文的最后一组明文分组使用的**是** `CTR + i`，下一个明文的第一组明文分组的**不是** `CTR + i + 1`

```
    def encrypt(self, msg):
        msg = self.padding(msg)
        if self.r != -1:
            self.r += 1
            aes = AES.new(self.key, AES.MODE_CTR, counter = Counter.new(128, initial_value=self.r))
            return aes.encrypt(msg)
        else:
            return msg
```

所以加密过程中的加密流实际上是`CTR + 1`，`CTR + 2`，`CTR + 3`等等通过**加密**操作生成的

并且上一个明文（不是明文分组）加密所涉及的`CTR`初始值是`CTR + 1`，下一个明文加密所涉及的`CTR`初始值是`CTR + 2`（因为每次加密整个明文之前`self.r += 1`）

所以

$$m: CTR+1 \quad CTR+2 \quad CTR+$$

为了使得`flag`使用的是已知的加密流，我们使用`set key`操作把`CTR`刷新，使得

$$flag: CTR+1 \quad CTR+2 \quad CTR+$$

Solving code

```python
from pwn import *
from Crypto.Util.number import *
import gmpy2
import hashlib
import string
import random
import binascii

context.log_level='debug'
io = remote("192.168.109.129","9998")
io.recvuntil(b"+")
tmp = io.recvuntil(b"\n")
part_proof = tmp.split(b") == ")[0]
result = tmp.split(b") == ")[1][:-1]
table = string.ascii_letters + string.digits
while True:
    XXXX = ("".join(random.sample(table,4))).encode()
    if hashlib.sha256(XXXX + part_proof).hexdigest() == result.decode():
        io.recvuntil(b"\n")
        io.sendline(XXXX)
        break

n = int(binascii.unhexlify(io.recvline()[:-1]))
hint = int(binascii.unhexlify(io.recvline()[:-1]))
c =  int(binascii.unhexlify(io.recvline()[:-1]))
mac = gmpy2.iroot(pow(c,hint,n) % n,2)[0]


io.sendline(b"set key:" + str(mac).encode())
enc = []
for _ in range(3):
    io.sendline(b"I_want_flag")
    time.sleep(0.5)
    tmp = io.recvline()[:-1].decode()
    if _ != 2:
        enc.append(long_to_bytes(int(str(tmp),16))[:16])
    else:
        enc.append(long_to_bytes(int(str(tmp),16)))
# print(enc)
io.sendline(b"set key:" + str(mac).encode())
time.sleep(0.5)
io.sendline(b"get flag")
time.sleep(0.5)
tmp = io.recvline()
enc_flag = binascii.unhexlify(tmp[:-1])

msg = b'something wrong, check your input'
msg = msg + chr((16 - len(msg)%16)).encode() * (16 - len(msg)%16)
key_stream = b""
for i in enc:
    key_stream += xor(i,msg[:len(i)])
print(xor(enc_flag,key_stream[:len(enc_flag)]))
```

Conclusion

关于`AES`的深入一点的题目都是交互式的，正常情况不会要求求密钥`key`，而是利用`xor`的特性以及**加密器**（相当于黑盒加密）来从服务器脚本处**骗取**可以求得`flag`的数值

关于交互式的题目没有头绪的时候更倾向于直接与服务端进行交互查看`debug`返回，而不单是查看服务器所使用的脚本

`Crypto`的交互式题目需要注意服务端返回的内容是什么类型的，很可能脚本会对十进制数值转字节再转十六进制，可能要进行一定的数据处理