

# HGAME2020 Week3 Writeup

原创

wh201906 于 2020-02-08 01:50:16 发布 545 收藏 1

分类专栏: [CTF试水](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/wh201906/article/details/104218119>

版权



[CTF试水](#) 专栏收录该内容

4 篇文章 0 订阅

订阅专栏

签到成功

只求week4不要交白卷

Crypto - Exchange

题目:

```
Our admin hijacked a secret channel and it looks like there are two teams doing some unspeakable transactions.
nc 47.98.192.231 25258
```

一开始先是考了个week2的Crypto签到题, 直接用上次的解题脚本即可  
之后的题目是“我”截获了Alice和Bob的通信, 并且“我”可以篡改当中的部分内容  
然而双方使用了非对称加密, 使得“我”无法直接获得解密用的密钥

一段正常的、消息未被篡改的对话记录如下:

```
root@tesla:~# nc 47.98.192.231 25258
sha256(XXXXxAkaaFN1enC5UQm1) == a196d23933aaa95a960505f51b0bbb80e4d5e171c650ee77e5f690db0da49f9d
Give me XXXX: hhA9
Bob: Hi Alice, I got the second half of the flag.
Alice: Really? I happen to have the first half of the flag.
Bob: So let's exchange flags, :-)
Alice: Ok,
Alice: Ah, wait, maybe this channel is not secure,
Alice: Let's do Key Exchange first.

[INFO] : ***** STEP.1 *****
[INFO] : Alice and Bob publicly agree to use a modulus `p` and base `g`

Alice: p = 15355689228302638107794080400839486113151574198965882167917541684969851758757593898150445443277393656
8307359774473745489589317350169237690857181940569391633955685290014611082866737102235415668265604660796650340393
980508711731525896209747614702453521502602117643570282460093254535065811953844942144919365256521
Alice: g = 12533226630211123267480634698107636517645717206048415151467364959257674379888339001890561804433849050
919145849131053263972695583418788336181686631504927023

[INFO] : ***** STEP.2 *****
[INFO] : Alice generates her private key `a` which is a random number from the set {2, 3, ... p-2}.
[INFO] : Bob does exactly the same: he compute his private key `b` which is taken randomly from the same set of
integers.
```

```

[INFO] : Now, Alice computes her public key `A` which is simply `g` raises to the `a`'s power modulo `p`, a.k.a,
`A = pow(g, a, p)`.
[INFO] : Bob does the same: Bob computes the `B = pow(g, b, p)`.

[INFO] : ***** STEP.3 *****
[INFO] : Alice and Bob exchange their public key.
[WARNING] : You intercepted Alice's message, which is
[WARNING] : A = 706475118629741676057181269330195341998995537813607425265729582411293548387589806502690800311713
8641338315836461348474051252929399423120186795645956190435499788156130101697925071764086182680050093223119778291
8763790608092952036935588386452098820212616532542944466948129452067049514243474150423655883359222625
[WARNING] : Do you want to modify this message? (yes/no)
> no

Alice: A = 70647511862974167605718126933019534199899553781360742526572958241129354838758980650269080031171386413
3831583646134847405125292939942312018679564595619043549978815613010169792507176408618268005009322311977829187637
90608092952036935588386452098820212616532542944466948129452067049514243474150423655883359222625

[WARNING] : Again, you intercepted Bob's message, which is
[WARNING] : B = 550852498751127238569250711436854048550373572026931252676279478727174791502269736445821453738655
5453200635468341157114126561288998847697935504479025455406860059600269360503605276391541812195158293280901192522
3738451952524155809034538713880059778382169632171663072947110450799562961709324788795954190769715233
[WARNING] : Do you want to modify this message? (yes/no)
> no

Bob: B = 5508524987511272385692507114368540485503735720269312526762794787271747915022697364458214537386555453200
6354683411571141265612889988476979355044790254554068600596002693605036052763915418121951582932809011925223738451
952524155809034538713880059778382169632171663072947110450799562961709324788795954190769715233

[INFO] : ***** STEP.4 *****
[INFO] : Alice takes public key of Bob (`B`), raises to, again, to the `a`'s power modulo `p`, the she gets the s
ecret value `S_a = pow(B, a, p)`.
[INFO] : Bob also gets the secret value `S_b = pow(A, b, p)`.
[INFO] : Right now, they have done the key exchange.
[WARNING] : hint: does Alice and Bob share the same key?

Alice: Now, we have securely shared the secret value.
Bob: Right, let's exchange the encrypted flag!

[INFO] : For the encryption, Alice compute `C_a = (m * S_a) % p`
[INFO] : Bob does the same, `C_b = (m * S_b) % p`

[WARNING] : Bob is trying to send a message to Alice,
[WARNING] : C_b = 11091334160372016479689181521057422435002115326690332084322306805900112196757073369179632452520
2094785906639422350854888191815074216368455818691383364061826587141528725929442688841090153110271122789180372616
5091368041974003571462338237395576033657204061415069107115364368082057002132107630185756527268217377247
[WARNING] : Do you want to modify this message? (yes/no)
> no

Alice: Great, I get the flag.
Alice: C_a = 124986611840107028357373332961427876017685009748795981788067670242988099919086562972017332366455079
2942111321318124130226281179525563567168583748928048761915095401892418405382431595376883618790320016767348369179
46408430037028348731342744428672358594885114018261862941151688600653201876316592113600143479932888

Happy cooperation. :)

```

一开始先是根据p、g两个名字去找相关的加密算法，在StackOverflow上面找到一个[问答](#)，似乎与SSH2有关。之后搜索发现找不到相关加密算法。。。后来又去CTF Wiki上面翻RSA相关内容（目前就只知道这个了），发现还是没有思路之后由于整个week3都没啥会做的，而这道题在Crypto当中完成人数最多，决定还是来啃一下

因为找不到类似的加密算法，而且最后求密文的时候用的是乘法（似乎在现代加密算法当中不太常见），因此猜测可能是要从加密过程本身来进行分析

分析两人在对话过程中所获得的信息如下：

```
A:p,g
B:p,g
A:p,g,a,A
B:p,g,b,B
A:p,g,a,A,B
B:p,g,b,B,A
A:p,g,a,A,B,S_a
B:p,g,b,B,A,S_b
A:p,g,a,A,B,S_a,C_b
B:p,g,b,B,A,S_b,C_a
```

整个过程中 $a,b$ 是私钥， $A,B$ 是传给对方的公钥， $A,B$ 应该是用于解密的（题目中没给具体解密算法所以未知）

尝试分析一下密文，一个非常不严谨的分析过程如下（还没开始学数论）：

```
Ca=m%p*S_a%p
=m%p*S_a
=m%p*B^a%p
=m*B^a%p
=m*g^b%p^a%p
=m*g^b^a%p
=m*g^a^b%p
=m*g^a%p^b%p
=m*A^b%p
=m%p*A^b%p
=m%p*S_b
```

意外发现 $S_a$ 与 $S_b$ 好像是相同的

怀疑自己推导过程中出了问题，但看到聊天记录中有"hint: does Alice and Bob share the same key?"，考虑到两人公钥私钥不同，所以题目应该是在说 $S_a$ 和 $S_b$ 相同

之后又卡壳了几天

后来冷静分析一波，这个加密的安全性就体现在 $a$ 与 $b$ 不需要发送给对方（也就是无法被截获），而目前似乎没有方法用其它信息推出 $a$ 与 $b$ 。加密密钥 $S$ 受 $a$ 与 $b$ 共同影响，因此解密所需的 $S^{-1}$ 也变成了未知状态。一种思路是让 $S$ 变为定值或者用其它信息推出 $S$ ，那么密文就可以解出来了

因为 $S$ 与 $a,b$ 两个量有关，结合这几周Crypto题当中多次出现的逆元，很自然就想到了构造公钥让 $S$ 当中的 $a、b$ 互相抵消。经过各种尝试（因为不会证明），存在以下破解流程：

```
A = g^a%p (不变)
B = A^-1 (篡改)
S_a = B^a%p = g
```

这样 $S$ 就变成了 $g$ ，就可以解出 $m$ 了

然而具体实施的过程并没有这么顺利。由于两人特定的通信顺序，出现了两种情况：

1.篡改 $B$ 是为了使 $S_a=g$ ，进而通过 $C_a$ 解出 $m$ 。而如果 $B$ 被篡改，则Alice无法用错误的 $B$ 解出正确的flag，因此不会提供 $C_a$

2.截获公钥的顺序永远是先A后B，因此无法篡改A使得 $A=B^{-1}$

破解失败

强烈怀疑两人之间ABBA的通信顺序就是用来堵这种方法的。不过这也提供了破解的方向：只能通过篡改A密钥来破解 $C_b$ 中的内容

之后想到了一个更简单的方法，对于指数运算，1也是一个很好的工具。令 $A=1$ ，则 $S_b=1$ ，那么 $C_b$ 就可以破解了。兴高采烈地把 $A$ 改为1，结果如下：

```
[WARNING] : Select a decimal number
> 1
Invalid A
Rua!!!
```

如此简单的方法被ban也是情理之中

接着尝试令 $A=2$ ，或许能从尾数当中看出点东西。然而经过 $m * S_b$ 的运算之后也看不出什么了

因为之前构造逆元的时候是从 $g$ 出发，于是这次准备换个角度，从 $p$ 出发找找思路。对 $p$ 进行简单拼凑之后突然回想起二项式定理， $(p+1)^a$ 展开后的 $a+1$ 项中前 $a$ 项都含有因式 $p$ ，而最后一项为 $1^a=1$ ，也就是有 $(p+1)^a = kp+1$ ，因此 $((p+1)^a) \% p = 1$ 。通过令 $A=p+1$ 也可以使得 $S_b$ 变为1。

兴高采烈地根据 $p$ 把 $A$ 改为 $p+1$ ，结果如下：

```
Invalid A
Rua!!!
```

接着又尝试把 $A$ 改为 $2p, (p^2)+1$ 和 $2p+1$ ，都失败了

因为摸不透出题人对于合法 $A$ 的验证方式，于是又卡壳了几天

后来想到计算 $A$ 的时候也涉及到模 $p$ 的操作，因此一个正常的 $A$ 一定小于 $p$ ，而前几次用二项式定理构造 $A$ 的时候都不满足这一条件。为了使得 $A < p$ ，可以构造 $A=p-1$ 使得 $S_b$ 变为1或-1（取决于 $b$ 是否为偶数），这样也满足让 $S$ 变为定值的攻击条件。

令 $A=p-1$ ，直接得到以下 $C_b$

```
[WARNING] : Bob is trying to send a message to Alice,
[WARNING] : C_b = 4328638271619280595085272697247544492929485359891054205
```

可以发现这次得到的 $C_b$ 长度明显小于正常 $C_b$ 的长度。而且经过多次尝试， $C_b$ 较短时永远是

4328638271619280595085272697247544492929485359891054205（ $b$ 为偶数），而当 $C_b$ 较长时（ $b$ 为奇数），将得到的 $C_b$ 乘上-1关于 $p$ 的逆元（用`gmpy2.invert()`求）并模 $p$ ，也可以得到

4328638271619280595085272697247544492929485359891054205，说明刚才的思路应该是正确的

直接在python中运行

```
print((4328638271619280595085272697247544492929485359891054205).to_bytes(50, 'big'))
```

得到结果

```
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00-1n~ThE+miDd!3_4TtAck~}'
```

那么flag的后半段就是

```
-1n~ThE+miDd!3_4TtAck~}
```

由于flag里面是有意义的内容，接下来暴力猜测flag前半段即可



```

import string, sys, re
from socket import socket
from telnetlib import Telnet
from hashlib import sha256
from multiprocessing import Pool, Pipe
from time import sleep

table = (string.ascii_letters + string.digits).encode()
prefix = [string.ascii_lowercase.encode()[0:13], string.ascii_lowercase.encode()[13:26], string.ascii_uppercase.encode()[0:13], string.ascii_uppercase.encode()[13:26], string.digits.encode()]

def task(pipe, index, c, part):
    print('task', index, "is running")
    for i in prefix[index]:
        for j in table:
            for k in table:
                for l in table:
                    raw = i.to_bytes(1, 'big')
                    raw += j.to_bytes(1, 'big')
                    raw += k.to_bytes(1, 'big')
                    raw += l.to_bytes(1, 'big')
                    raw += part
                    if sha256(raw).hexdigest().encode() == c:
                        pipe.send(raw.decode()[:4].encode())

if __name__ == '__main__':
    sock = socket()
    sock.connect(("47.98.192.231", 25258))
    text = sock.recv(1024).decode()
    sleep(0.02)
    sock.recv(1024)
    print(text)
    part = re.search(r'(?<=\+).{16,}(?=\))', text,).group(0).encode()
    c = re.search(r'(?<=\= \= ).+', text,).group(0).encode()
    (root, sub)=Pipe()
    pool = Pool(processes=5)
    for i in range(5):
        pool.apply_async(func=task, args=(sub, i, c, part))
    print("all process started")
    pool.close()
    pool.join()
    print("all process stopped")
    res = root.recv()
    print(res)
    sock.send(res)
    sleep(0.02)
    tel = Telnet()
    tel.sock = sock
    tel.interact()

```

---

## Crypto - Feedback

题目:

听说上周Classic\_CrackMe的CBC很简单? 来耍个CFB试试 XD  
nc 47.98.192.231 25147

然后是服务端的脚本:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import os, random
import string, binascii
import signal
import socketserver
from hashlib import sha256
from Crypto.Cipher import AES

from secret import MESSAGE
assert len(MESSAGE) == 48

class Task(socketserver.BaseRequestHandler):
    def __init__(self, *args, **kwargs):
        self.KEY = b""
        self.IV = b""
        super().__init__(*args, **kwargs)

    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 encrypt(self, data):
        assert len(data) % 16 == 0
        aes = AES.new(self.KEY, AES.MODE_CFB, self.IV, segment_size=128)
        return aes.encrypt(data)

    def decrypt(self, data):
        assert len(data) % 16 == 0
        aes = AES.new(self.KEY, AES.MODE_CFB, self.IV, segment_size=128)
        return aes.decrypt(data)

    def handle(self):
        signal.alarm(60)
        self.KEY = os.urandom(32)
        self.IV = os.urandom(16)

        self.send(b"You have only 3 times to decrypt sth, then I'll give u the FLAG.")
        try:
            for _ in range(3):
                self.send(b"Give me sth(hex) to decrypt")
                hex_input = self.recv()
                if not hex_input:

```

```

    if not hex_input:
        break
    ciphertext = binascii.unhexlify(hex_input)
    plaintext = self.decrypt(ciphertext)
    self.send( binascii.hexlify(plaintext) )
except:
    self.send(b"Rua!!!")
    self.request.close()

enc_msg = self.encrypt(MESSAGE)
self.send(b"Here is your encrypted FLAG(hex): ", newline=False)
self.send( binascii.hexlify(enc_msg) )
self.request.close()

```

```

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

```

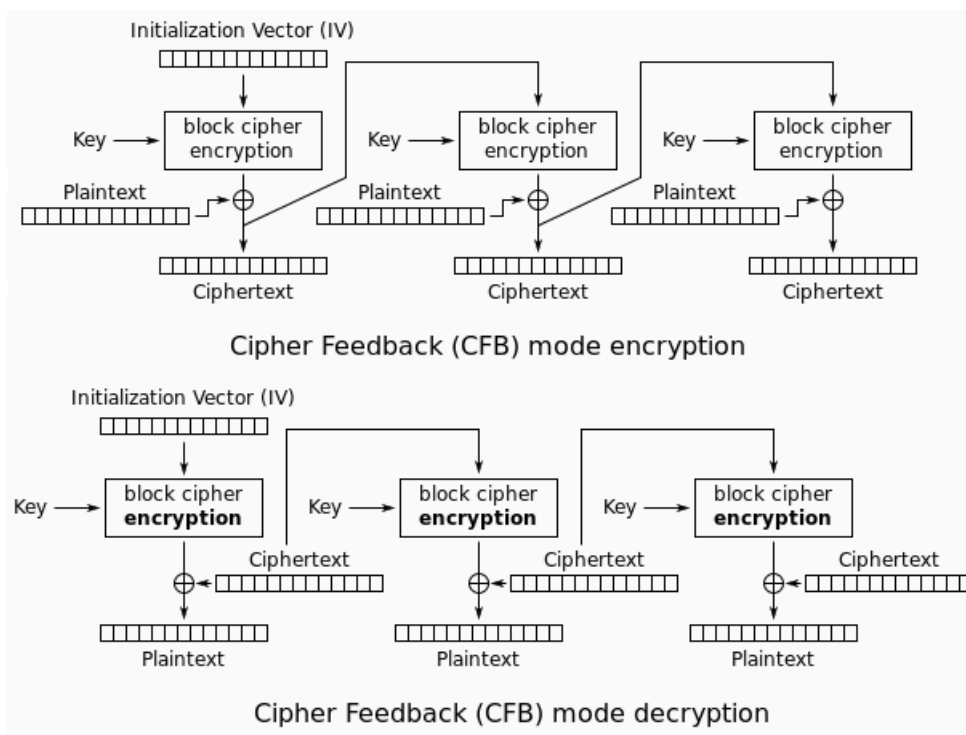
```

if __name__ == "__main__":
    HOST, PORT = '0.0.0.0', 1234
    server = ForkedServer((HOST, PORT), Task)
    server.allow_reuse_address = True
    server.serve_forever()

```

连接指定地址后，服务器会提供3次机会解密任意文本，并将解密结果以及加密后的flag显示出来  
脚本中有os.urandom()，因此每一次连接中服务器使用的加解密参数都是不一样的

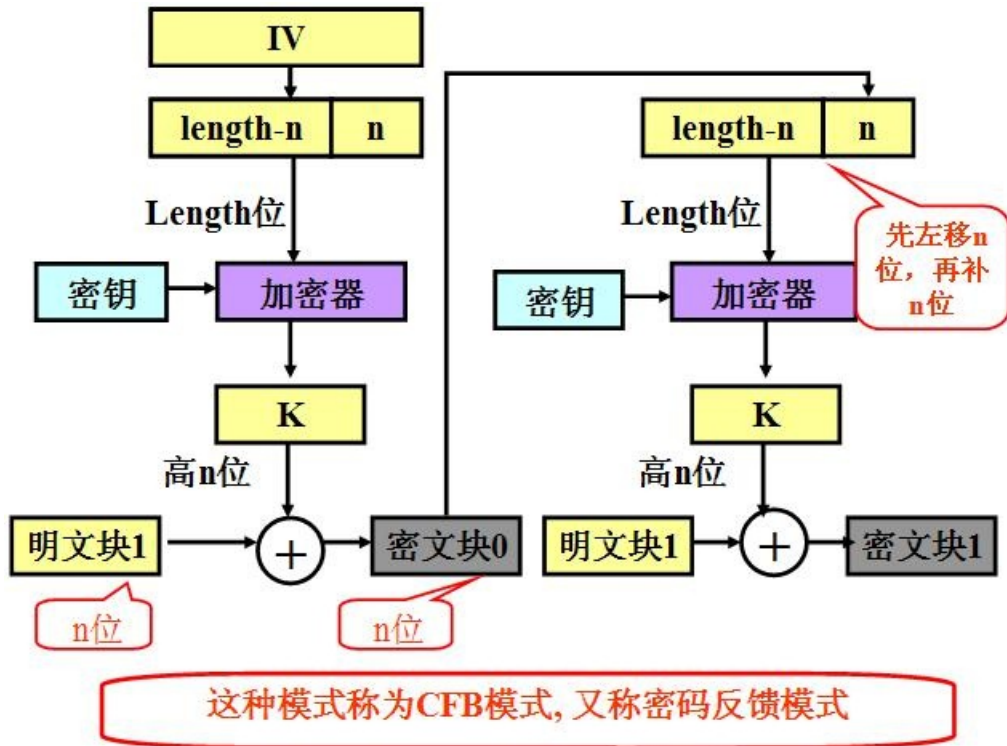
题目里面明确提到了CFB，经过百度，发现这是AES的一种加密模式（坑）。原理图如下：



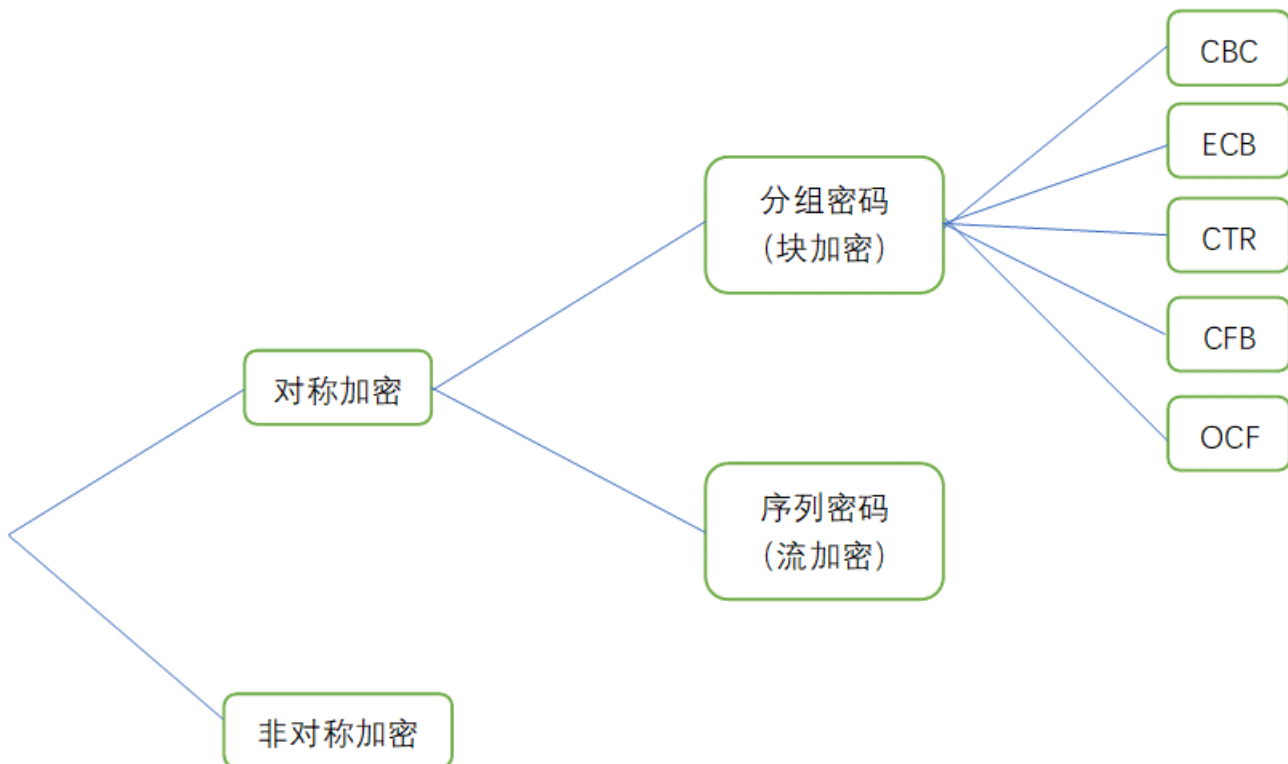
对图上的各个部分进行搜索，得知图中的 $\oplus$ 表示异或，IV是一个用于首轮加密的初始参数（初始化向量），然而中间的block cipher encryption就不知道是什么意思了

之后找到了用中文标注的图片，但还是没弄清楚加密器是怎么生成K的





之后直接百度AES加密，得知这是一种对称加密的方式，而且其原理很复杂（对我来说），并且之前查到的几种加密模式（CBC、ECB、CTR、CFB和OCF）并不只是针对于AES，而是所有分组加密（块加密）都可以使用的整个关系图应该是这个样子



而之前图片中的block cipher encryption以及“加密器”就是所选用的具体加密算法，在这里是AES

后来查到服务端脚本中的segment\_size=128表明每处理128位调用一次加密器，而因为len(MESSAGE)==48，所以整个加密器只被调用了一次（坑）。结合之前的CFB原理图，我只需要提供一串长度为48的文本C，将其提供给服务器解密得到m，之后计算 $K=C^m$ ，然后将加密后的flag与K异或即可获得明文。题目给的3次机会明显多余（坑）

然而题目很明显不会这么简单。实际操作结果如下：

```
C(编码前, 自定义)=b'123456789012345678901234567890123456789012345678'
C(编码后)=313233343536373839303132333435363738393031323334353637383930313233343536373839303132333435363738
m(16进制编码)=f8a940949cbce4fa99c87070055517fd926d8107c69aa64675d188fc9461400352cac50ad9af1660e9def81c1bbccd70
m(解码后)=b'\xf8\xa9@\x94\x9c\xbc\xe4\xfa\x99\xc8pp\x05U\x17\xfd\x92m\x81\x07\xc6\x9a\xa6Fu\xd1\x88\xfc\x94a@\x0
3R\xca\xc5\n\xd9\xaf\x16` \xe9\xde\xf8\x1c\x1b\xbc\xcdp'
K=b'\xc9\x9bs\xa0\xa9\x8a\xd3\xc2\xa0\xf8AB6a"\xcb\xa5U\xb87\xf7\xa8\x95r@\xe7\xbf\xc4\xadQq1a\xfe\xf0<\xee\x97/
P\xd8\xec\xcb(. \x8a\xfaH'
flag(16进制编码, 解密前)=8fd732e789e3a0e2c89f202f531a17facd049273a075ec70f393376473f833a7486269c7076d3341cad2c4041
2009150
flag(解密前)=b'\x8f\xd72\xe7\x89\xe3\xa0\xe2\xc8\x9f /S\x1a\x17\xfa\xcd\x04\x92s\xa0u\xecp\xf3\x937ds\xf83\xa7Hb
i\xc7\x07m3A\xca\xd2\xc4\x04\x12\x00\x91P'
flag(解密后)=b'FLAG is hgame{51hQ*DW\xddy\x02\xb3t\x88\xa0\xde\xa9B\x96)\x9c\x99\xfb\xe9\xfa\x1c\x11\x12>\x0f,<\
x8ak\x18'
```

发现flag的后半段是无法打印的字符，并且最后也没有出现}

怀疑要对flag后半段进行异或。因为 $0x7d('}' \text{ 对应编码})^0x18=0x65=101$ ，所以将刚才的flag整体与101异或，得到以下结果

```
b'#)$"E\x0c\x16E\r\x02\x04\x08\x00\x1ePT\r40!2\xb8\x1cg\xd6\x11\xed\xc5\xbb\xcc'\xf3L\xf9\xfc\x9e\x8c\x9fytw[ji
Y\xef\x0e}'
```

妥妥陷入了僵局。。。

后来才发现segment\_size=128指的是128位（16个字节），而len(MESSAGE)==48是指48个字节，因此整个加密器被使用了3次。上面的操作只能解出MESSAGE的前16个字节

（回想一下也觉得自己挺傻的，既然考点是CFB，那么一次简单的异或肯定不能解决问题）

之后进一步分析CFB，发现在这种模式下，某个块的加密结果和前一个块的加密结果有关。举个例子，在segment\_size=8（1个字节）的情况下，使用相同的key和初始向量IV加密b'12345'和b'02345'，即使明文只有第1个字节不同，之后相同的'2345'加密之后的结果也不一样。通过这种方式加密，即使明文中出现了多个完全相同的块，攻击者也无法从密文当中直接看出来

（与其形成鲜明对比的是ECB模式，块与块的加密过程完全独立（类似于week1的仿射密码，在指定A、B的情况下单个明文字符和密文字符的对应关系是唯一的），虽然方便并行计算，但是在攻击者拥有大量明文和对应密文的情况下更容易受攻击）

而从本题的角度来分析，如果我提交的待解密密文是任意的，那么刚才的方法只能确保加密器第一次生成的K（K用于与明文/密文异或，图中有说明）与加密flag的第一个K相同（因为都是用IV和Key生成的），而之后加密器生成的K就与加密flag的K不一样了（因为参数中只有Key相同，而上一个块的密文不同），因此前者的K不能用于解密后者。这也就是刚才的解密过程只能获得MESSAGE前16个字节（128位）的原因

不过目前已知第一个块的明文m1，在获取了第一个块的K1之后可以算出第一个块的密文 $c1=m1^k1$ ，将第一个块的密文与任意字符串S2+X拼接得到密文 $C'=c1+S2+X$ （仍然需要确保len(C')==48，且S2=16），将其提交给服务器解密得到 $M'=m1+s2+X$ ，此时计算 $K2=S2^s2$ 即可得到正确的K2，并且用K2可以解出flag的第二部分m2

同理，K3,m3也可以求出来，最后flag=m1+m2+m3

当前思路用符号表述如下（括号内为字符长度）：

其中S是提交的密文，s是服务器返回的明文，K用于与明文/密文异或（CFB中的K），C是flag的密文，m是flag的明文，X为任意字符串

整个过程中m全程不变，其余参数在单次连接内不变

第一次连接：

任意构造 $S=S1(16)+X(32)$ ，提交一次得到 $s=s1(16)+X(16)$ ，计算 $K1(16)=S1^s1$

之后服务器提供 $C=C1(16)+X(32)$ ，计算 $m1(16)=C1^k1$

第二次连接:

任意构造 $S=S1(16)+X(32)$ , 提交一次得到 $s=s1(16)+X(16)$ , 计算 $K1(16)=S1 \wedge s1$ ,  $C1(16)=m1 \wedge K1$

之后构造 $S'=C1(16)+S2(16)+X(16)$ , 再提交一次得到 $s'=m1+s2(16)+X$ , 计算 $K2(16)=S2 \wedge s2$

之后服务器提供 $C'=C1(16)+C2(16)+X(16)$ , 计算 $m2(16)=C2 \wedge K2$

第三次连接:

任意构造 $S=S1(16)+X(32)$ , 提交一次得到 $s=s1(16)+X(16)$ , 计算 $K1(16)=S1 \wedge s1$ ,  $C1(16)=m1 \wedge K1$

之后构造 $S'=C1(16)+S2(16)+X(16)$ , 再提交一次得到 $s'=m1+s2(16)+X$ , 计算 $K2(16)=S2 \wedge s2$ ,  $C2(16)=m2 \wedge K2$

之后构造 $S''=C1(16)+C2(16)+S3(16)$ , 再提交一次得到 $s''=m1+m2+s2(16)$ , 计算 $K3(16)=S3 \wedge s3$

之后服务器提供 $C''=C1(16)+C2(16)+C3(16)$ , 计算 $m3(16)=C3 \wedge K3$

最后得到 $flag=m1+m2+m3$

由于每一次连接服务器时的Key、IV都不同, 因此每一次连接服务器时取得的Kx都不一样, 所以每次连接的关键在于获得一部分flag (明文块) mx, 这样才可以进行下一步

具体实现过程中还有个问题, 因为服务端脚本中有`signal.alarm(60)`, 所以每次连接之后的所有操作要在60秒内完成, 否则连接会自动断开。只能写脚本了

最后编写的半自动脚本如下:

```

from socket import socket
from telnetlib import Telnet
from time import sleep
from binascii import hexlify,unhexlify
import re

XOR = lambda s1, s2: bytes([x ^ y for x, y in zip(s1, s2)])

base = '123456789012345678901234567890123456789012345678'.encode()
flag = 'FLAG is hgame{51b72d4cd23b2fe672a874cb44020868}'.encode()

sock = socket()
sock.connect(("47.98.192.231", 25147))
tel = Telnet()
tel.sock = sock
sleep(0.1)

text = tel.read_until(b'>').decode()
print(text, end='\n-----\n')

tel.write(hexlify(base))
sleep(0.1)
text = tel.read_until(b'>').decode()
print(text, end='\n-----\n')

c = unhexlify(re.search(r'(?<= ).+(?=\n)', text).group(0))
tmp1 = c[:16]
tmp2 = base[:16]
key = XOR(tmp1, tmp2)
print(key)

base = XOR(key, flag[:16]) + base[16:]
tel.write(hexlify(base))
sleep(0.1)
text = tel.read_until(b'>').decode()
print(text, end='\n-----\n')

c = unhexlify(re.search(r'(?<= ).+(?=\n)', text).group(0))
tmp1 = c[16:32]
tmp2 = base[16:32]
key = XOR(tmp1, tmp2)
print(key)

base = base[:16]+ XOR(key, flag[16:32]) + base[32:]
tel.write(hexlify(base))
sleep(0.1)
text = tel.read_until(b'>').decode()
print(text, end='\n-----\n')

c = unhexlify(re.search(r'(?<= ).+(?=\n)', text).group(0))
tmp1 = c[32:]
tmp2 = base[32:]
key = XOR(tmp1, tmp2)
print(key)

tel.interact()

```

最终flag: hgame{51b72d4cd23b2fe672a874cb44020868}

后记：  
刚看到初始化向量(IV)的时候不知道其用途，以为解密方不需要IV只通过Key就能解密，于是猜测IV可以增强破解难度  
后来看资料和服务端代码发现解密时也需要提供IV  
经过后续分析，IV只是CFB加密的递归过程中的一个初始条件

---

## Misc - 美人鲸

题目：

我们受过严格的训练，无论在什么环境里，都不会乱放东西，  
除非忍不住。

后面是Docker镜像的地址

<https://hub.docker.com/r/zhouweitong/hgame2020-misc>

先是在Kali上安装Docker并下载镜像，然后百度到docker有个export命令可以将容器的文件系统提取为一个压缩包，提取并解压后直接搜索hgame无果，搜flag发现/usr/share/nginx/html/index.html里面有以下内容

```
You want flag? See $FLAG.
```

之后在Docker镜像中以交互模式运行sh，输入env  
查看环境变量，发现以下内容

```
FLAG=Find flag.tar.gz!
```

而刚刚的搜索结果当中就有flag.tar.gz!  
用压缩包管理器打开，里面含有flag.zip和README文件。flag.zip是被加密的压缩包，而README当中有以下内容

```
See sh history.
```

之后在镜像的sh中输入sh history，出现以下错误

```
sh: can't open 'history': No such file or directory
```

于是百度如何查看sh的历史记录，得知直接运行history命令即可。运行后发现一条记录

```
/ # history  
0 echo -e "Zip password is somewhere else in /etc.\nFind it!"
```

到头来还是要从文件搜索入手。直接搜password，发现在/etc/issue文件中有以下内容

```
Welcome to Alpine Linux 3.10  
Kernel \r on an \m (\l)  
  
Zip Password: cfuzQ3Gd6gqKG@$N
```

用该密码解压flag.zip，得到flag.db。Kali系统中可以直接打开这个数据库文件，得到flag  
最终flag：hgame{v3RWI3qSpckZhp^xv\$kaBhNjVqXk##3e}

后记：  
该题时间分配：5%搜相关命令 5%解题 90%安装Docker（雾  
（写了篇安装Docker的博文不到3天访问破千）  
（网上有现成的安装方法，因为想修改一点点所以装了很久）

---

看完week4的题，感觉自己要凉

2020.02.08