Hgame 2019 Writeup Week4

Author: Rainbow

IV MISC

1. Warmup

Question

Description

提交管理员密码的sha256，自己补上格式hgame{}

URL

http://pmsw8b6r5.bkt.clouddn.com//ea72c72ba8808ccc22ca7f0fac63cfcb/1.zip

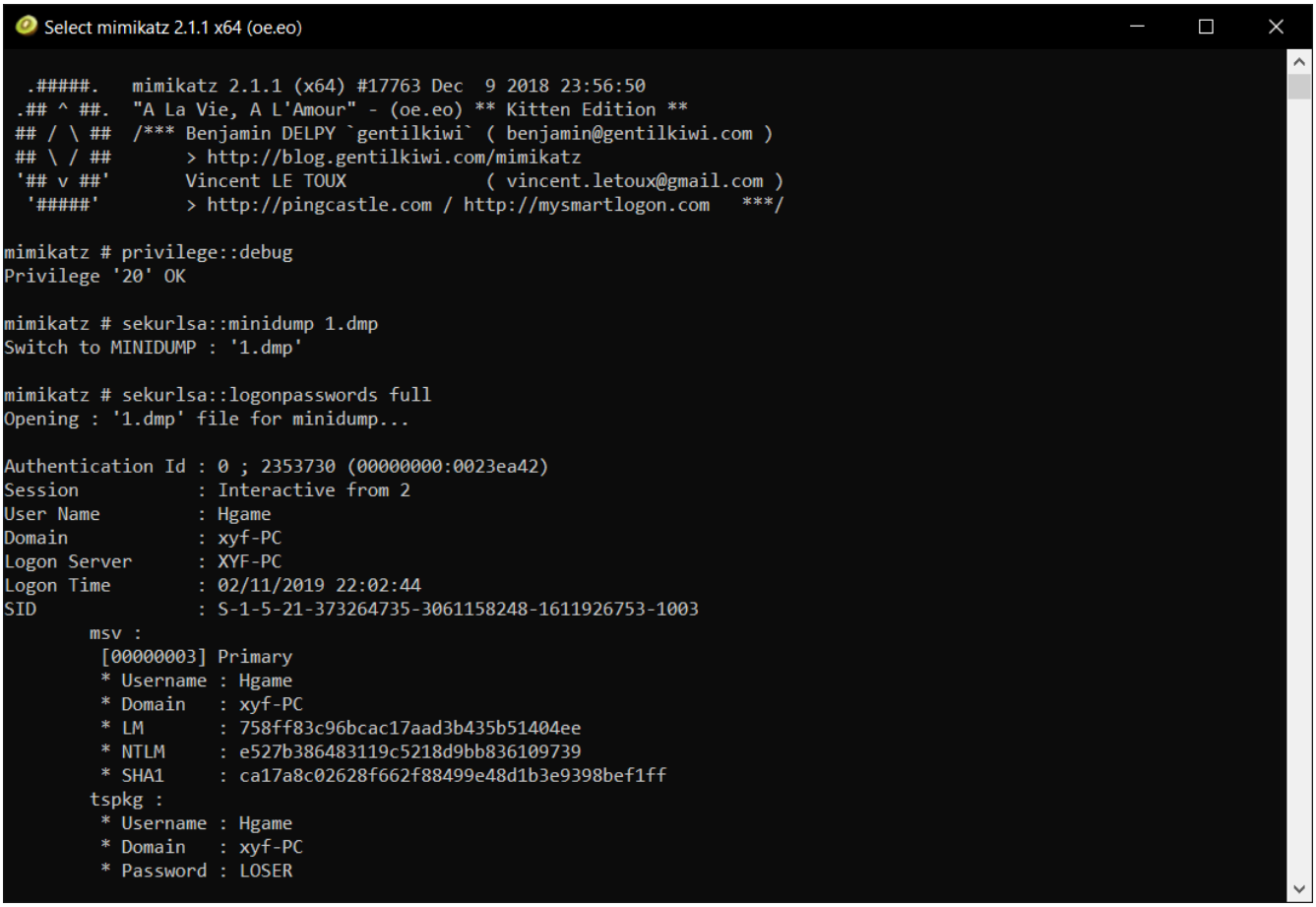
Base Score

100

Answer

这题主要就是用到了mimikatz

我把解压出来的1.gif改名成1.dmp



所以可以看到密码就是LOSER

SHA256就是

`dd6dffcd56b77597157ac6c1beb514aa4c59d033098f806d88df89245824d3f5`

所以flag就是

`hgame{dd6dffcd56b77597157ac6c1beb514aa4c59d033098f806d88df89245824d3f5}`

3. 暗藏玄机

Question

Description

要开学了，要开学了（悲）

URL

http://plqfgjy5a.bkt.clouddn.com/%E5%BC%80%E5%AD%A6.zip

Base Score

150

Answer

打开压缩包，里面是两张看上去一模一样的图片。

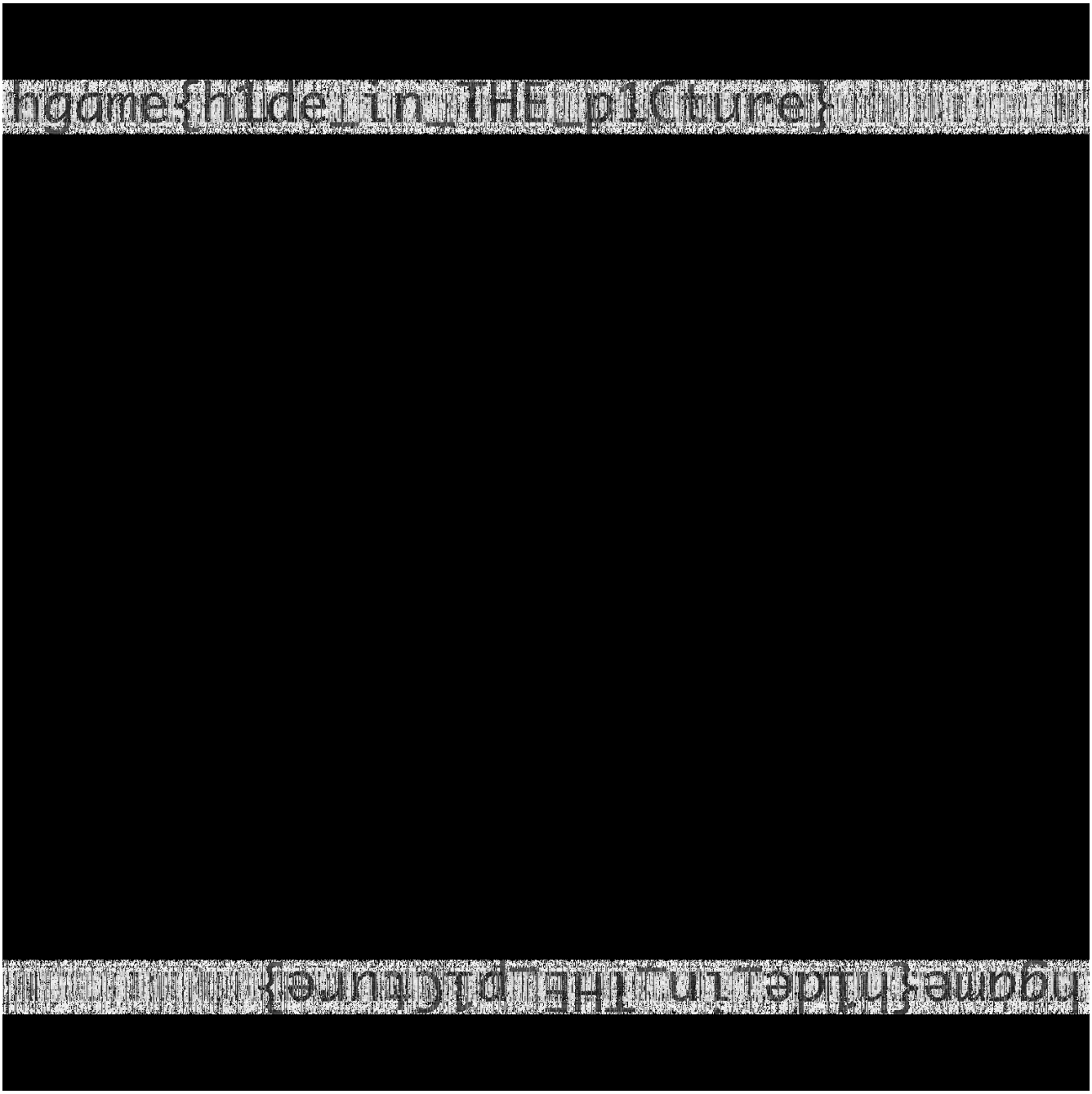
一开始我看了半天，并没有发现什么。

最后在py了学长之后拿到了hint：`盲水印`

网上有现成地代码(https://github.com/YvesZHI/BlindWaterMark/blob/master/bwm.py)

然后执行`python bwm.py decode 开学了.png 开学啦.png flag.png`

就会得到一张图片



所以flag就是`hgame{h1de_in_THE_p1Cture}`

（说实话，真的很简单，对我来说，主要的困难就是python2和3之间不兼容的这个硬伤，以及所依赖的库安装不上去。。。）

V CRYPTO

1. easy_rsa

Question

Description

m为17位十进制数，提交格式hgame{m}

URL

http://pmuyzdinx.bkt.clouddn.com/1.txt

Base Score

200

Answer

打开给的题目如下：

```
e1:0x33240

e2:0x3e4f

n:0x9439682bf1b4ab48c43c524778c579cc844b60872275725c1dc893b
5bcb358b9f136e4dab2a06318bb0c80e202a14bc54ea334519bec023934
e01e9378abf329893f3870979e9f2f2be8fff4df931216a77007a2509f4
9f697bf286285e97fac5dc6e4a164b5c2cc430887b18136437ba67777bd
a05aafdeaf918221c812b4c7d1665238f84ab0fab7a77fcae92a0596e58
343be7a8e6e75a5017c63a67eb11964970659cd6110e9ec6502288e9e44
3d86229ef2364dfecb63e2d90993a75356854eb874797340eece1b19974
e86bee07019610467d44ec595e04af02b574a97fa98bdb2e779871c8042
19cab715f4a80fef7f8fb52251d86077560b39c1c2a1

c1:0x7c7f315a3ebbe305c1ad8bd2f73b1bb8e300912b6b8ba1b331ac24
19d3da5a9a605fd62915c11f8921c450525d2efda7d48f1e503041498f4
f0676760b43c770ff2968bd942c7ef95e401dd7facbd4e5404a0ed3ad96
ae505f87c4e12439a2da636f047d84b1256c0e363f63373732cbaf24bda
22d931d001dcca124f5a19f9e28608ebd90161e728b782eb67deeba4cc8
1b6df4e7ee29a156f51a0e5148618c6e81c31a91036c982debd1897e6f3
c1e5e248789c933a4bf30d0721a18ab8708d827858b77c1a020764550a7
fe2ebd48b6848d9c4d211fd853b7a02a859fa0c72160675d832c94e0e43
355363a2166b3d41b8137100c18841e34ff52786867d

c2:0xf3a8b9b739196ba270c8896bd3806e9907fca2592d28385ef24afa
dc2a408b7942214dad5b9e14808ab988fb15fbd93e725edcc0509ab0dd1
656557019ae93c38031d2a7c84895ee3da1150eda04cd2815ee3debaa7c
2651b62639f785f6cabf83f93bf3cce7778ab369631ea6145438c3cd4d9
3d6f2759be3cc187651a33b3cc4c3b477604477143c32dfff62461fdfd9
f8aa879257489bbf977417ce0fbe89e3f2464475624aafef57dd9ea6033
9793c69b53ca71d745d626f45e6a7beb9fcbd9d1a259433d36139345b7b
b4f392e78f1b5be0d2c56ad50767ee851fac670946356b3c05d0605bf24
3b89c7e683cc75030b71633632fb95c84075201352d6
```

```
c1=pow(m, e1, n)
c2=pow(m, e2, n)
```

可以看到这里对m进行了两次同模加密，所以可以使用同模攻击

原理如下：(来自[ctf-wiki][https://ctf-wiki.github.io/ctf-wiki/crypto/asymmetric/rsa/rsa_module_attack/#_6])

设两个用户的公钥分别为 e_1 和 e_2，且两者互质。明文消息为 m，密文分别为：

$$
\begin{aligned}
c_1 &= m^{e_1} \bmod N \\
c_2 &= m^{e_2} \bmod N
\end{aligned}
$$

当攻击者截获 c_1 和 c_2 后，就可以恢复出明文。用扩展欧几里得算法求出 $re_1 + se_2 = 1 \bmod n$的两个整数r 和 s，由此可得：

$$
\begin{aligned}
c_1^r c_2^s &\equiv m^{re_1} m^{se_2} \bmod n \\
&\equiv m^{(re_1+se_2)} \bmod n \\
&\equiv m \bmod n
\end{aligned}
$$

代码如下

```
e1 = 0x33240
e2 = 0x3e4f
n = 
0x9439682bf1b4ab48c43c524778c579cc844b60872275725c1dc893b5b
cb358b9f136e4dab2a06318bb0c80e202a14bc54ea334519bec023934e0
1e9378abf329893f3870979e9f2f2be8fff4df931216a77007a2509f49f
697bf286285e97fac5dc6e4a164b5c2cc430887b18136437ba67777bda0
5aafdeaf918221c812b4c7d1665238f84ab0fab7a77fcae92a0596e5834
3be7a8e6e75a5017c63a67eb11964970659cd6110e9ec6502288e9e443d
86229ef2364dfecb63e2d90993a75356854eb874797340eece1b19974e8
6bee07019610467d44ec595e04af02b574a97fa98bdb2e779871c804219
cab715f4a80fef7f8fb52251d86077560b39c1c2a1
```

```
message1 =
0x7c7f315a3ebbe305c1ad8bd2f73b1bb8e300912b6b8ba1b331ac2419d
3da5a9a605fd62915c11f8921c450525d2efda7d48f1e503041498f4f06
76760b43c770ff2968bd942c7ef95e401dd7facbd4e5404a0ed3ad96ae5
05f87c4e12439a2da636f047d84b1256c0e363f63373732cbaf24bda22d
931d001dcca124f5a19f9e28608ebd90161e728b782eb67deeba4cc81b6
df4e7ee29a156f51a0e5148618c6e81c31a91036c982debd1897e6f3c1e
5e248789c933a4bf30d0721a18ab8708d827858b77c1a020764550a7fe2
ebd48b6848d9c4d211fd853b7a02a859fa0c72160675d832c94e0e43355
363a2166b3d41b8137100c18841e34ff52786867d
message2 =
0xf3a8b9b739196ba270c8896bd3806e9907fca2592d28385ef24afadc2
a408b7942214dad5b9e14808ab988fb15fbd93e725edcc0509ab0dd1656
557019ae93c38031d2a7c84895ee3da1150eda04cd2815ee3debaa7c265
1b62639f785f6cabf83f93bf3cce7778ab369631ea6145438c3cd4d93d6
f2759be3cc187651a33b3cc4c3b477604477143c32dfff62461fdfd9f8a
a879257489bbf977417ce0fbe89e3f2464475624aafef57dd9ea6033979
3c69b53ca71d745d626f45e6a7beb9fcbd9d1a259433d36139345b7bb4f
392e78f1b5be0d2c56ad50767ee851fac670946356b3c05d0605bf243b8
9c7e683cc75030b71633632fb95c84075201352d6


def gcdext(a, b):
    if b == 0:
        return 1, 0, a
    else:
        x, y, q = gcdext(b, a % b)  # q = gcd(a, b) =
gcd(b, a%b)
        x, y = y, (x - (a // b) * y)
        return x, y, q


def gcd(a, b):
    while a != 0:
```

```
        a, b = b % a, a
    return b


def findModReverse(a, m):
    if gcd(a, m) != 1:
        return None
    u1, u2, u3 = 1, 0, a
    v1, v2, v3 = 0, 1, m
    while v3 != 0:
        q = u3 // v3
        v1, v2, v3, u1, u2, u3 = (u1 - q * v1), (u2 - q *
v2), (u3 - q * v3), v1, v2, v3
    return u1 % m


s, t, pp = gcdext(e1, e2)
times = s * e1 + t * e2
if s < 0:
    s = -s
    message1 = findModReverse(message1, n)
if t < 0:
    t = -t
    message2 = findModReverse(message2, n)
plain = (pow(message1, s, n) * pow(message2, t, n)) % n
print(plain)
print(pow(plain, 1 / times))

plain = 21165526257396688106282379522017964460741216237106 9
i = 59594981651654789
```

由于times=3，所以plain可以开三次方，得到59594981651654789

所以flag就是`hgame{59594981651654789}`

（我也不知道为什么。。。）

2. MixedRSA_Easy

Question

Description

47.95.212.185 38610 由CNSS友情赞助 比心

URL

http://plqbnxx54.bkt.clouddn.com/MixedRSA_Easy.py

Base Score

400

Answer

（玩了这么久，总算拿了个二血，多了3%的分数加成，开心）

题目给的代码如下

```
#!/usr/bin/python3
from Crypto.Util.number import getPrime
from signal import alarm


def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a


def ex_gcd(m, n):
```

```
    x, y, x1, y1 = 0, 1, 1, 0
    while m % n:
        x, x1 = x1 - m // n * x, x
        y, y1 = y1 - m // n * y, y
        m, n = n, m % n
    return n, x, y


def inv(x, p):
    g, y, k = ex_gcd(x, p)
    if y < p:
        y += p
    return y


def xor(a, b):
    return bytes(x ^ y for x, y in zip(a, b))


class MixedRSA:
    n = e = d = iv = BLOCK = 0

    def __init__(self, iv, block=256):
        self.BLOCK = block
        p = getPrime(block * 4)
        q = getPrime(block * 4)
        self.n = p * q
        phi = (p - 1) * (q - 1)
        self.e = getPrime(64)
        self.d = inv(self.e, phi)
        iv *= block // len(iv)
        self.iv = iv.rjust(block, b'\x00')

    def padding(self, s):
```

```
        return b'\x00' * (-len(s) % self.BLOCK) + s

    def rsa_encrypt(self, m):
        c = pow(int(m.hex(), 16), self.e, self.n)
        c = hex(c)[2:].rjust(self.BLOCK * 2, '0')
        return bytes.fromhex(c)

    def rsa_decrypt(self, c):
        m = pow(int(c.hex(), 16), self.d, self.n)
        m = hex(m)[2:].rjust(self.BLOCK * 2, '0')
        return bytes.fromhex(m)

    def encrypt(self, plaintext):
        plaintext = self.padding(plaintext)
        imd = self.iv
        for i in range(0, len(plaintext), self.BLOCK):
            m = xor(imd[-self.BLOCK:], plaintext[i: i +
self.BLOCK])
            imd += self.rsa_encrypt(m)
        imd = imd[self.BLOCK:]
        cipher = self.iv
        for i in range(0, len(imd), self.BLOCK):
            c = self.rsa_encrypt(cipher[-self.BLOCK:])
            cipher += xor(c, imd[i: i + self.BLOCK])
        return cipher[self.BLOCK:]

    def decrypt(self, cipher):
        cipher = self.iv + cipher
        imd = b''
        for i in range(self.BLOCK, len(cipher),
self.BLOCK):
            c = self.rsa_encrypt(cipher[i - self.BLOCK: i])
            imd += xor(c, cipher[i: i + self.BLOCK])
        imd = self.iv + imd
```

```
        plaintext = b''
        for i in range(self.BLOCK, len(imd), self.BLOCK):
            m = self.rsa_decrypt(imd[i: i + self.BLOCK])
            plaintext += xor(m, imd[i - self.BLOCK: i])
        return plaintext


if __name__ == '__main__':
    alarm(60)
    mix = MixedRSA(FLAG)
    while True:
        print("Choose:\n[1] Encrypt (hex)\n[2] Decrypt
(hex)")
        op = input()
        if op == '1':
            msg = bytes.fromhex(input())
            print(mix.encrypt(msg).hex())
        elif op == '2':
            msg = bytes.fromhex(input())
            print(mix.decrypt(msg).hex())
        else:
            print('Bye')
            break
```

首先要拿到`FLAG`，其实也就是要拿到mix里面的那个`iv`，因为根据

```
mix = MixedRSA(FLAG)
```

以及

```
def __init__(self, iv, block=256):
    ...
    iv *= block // len(iv)
    self.iv = iv.rjust(block, b'\x00')
```

可以看出`iv`是由FLAG不断重复，然后左边补0产生的一个大小是256B的Bytes。

举个例子

```
FLAG = b'hgame{...}'
```

那么就会得到

```
iv =
'00000000000006867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d6867616d657b2e2e2e7d'
```

很明显后面是以`6867616d657b2e2e2e7d`为重复的，其实也就是`hgame{...}`的ASCII的十六进制。

要拿到`iv`的话，主要要关心的其实就是`encrypt`和`decrypt`两个函数

```
def encrypt(self, plaintext):
    plaintext = self.padding(plaintext)
    imd = self.iv
    for i in range(0, len(plaintext), self.BLOCK):
        m = xor(imd[-self.BLOCK:], plaintext[i: i +
self.BLOCK])
        imd += self.rsa_encrypt(m)
    imd = imd[self.BLOCK:]
    cipher = self.iv
    for i in range(0, len(imd), self.BLOCK):
        c = self.rsa_encrypt(cipher[-self.BLOCK:])
        cipher += xor(c, imd[i: i + self.BLOCK])
    return cipher[self.BLOCK:]
```

这个看上去像是CBC，并且是256B为一个Block，如果假设这里的`plaintext`刚好就是256B的长度，那么这个函数就可以化简为

```
def encrypt(self, plaintext):
        return xor(self.rsa_encrypt(xor(self.iv,
plaintext)),
                                    self.rsa_encrypt(self.iv))
```

那么对于plaintext为256B（也就是一个Block）的时候，其实用数学符号写出来就是

$$encrypt(plaintext) = RSA(plaintext \oplus iv) \oplus RSA(iv)$$

> 上面$RSA(x)$表示对x进行加密，$RSA^{-1}(x)$表示解密，下同

那么同理，对于decrypt，当cipher刚好为256B的时候，可以得到

$$decrypt(cipher) = RSA^{-1}(cipher \oplus RSA(iv)) \oplus iv$$

但是光凭这样两个化简版的函数，并不能得到`iv`

另外又发现（设$ZERO$为长度为256B，并且全为0的Bytes）

$$encrypt(ZERO) = ZERO = decrypt(ZERO) = ZERO$$

看上去没什么用，但是实际上根据原来的代码，我们可以控制下一个Block的“iv”。

所以可以把原来的$plaintext$拼在$ZERO$后面，变成两个Block。

那么根据原来的代码可以得到

$$\begin{aligned}&encrypt(ZERO + plaintext)\\&= ZERO + RSA(plaintext \oplus RSA(iv)) \oplus RSA(ZERO)\\&= ZERO + RSA(plaintext \oplus RSA(iv))\end{aligned}$$

（上面一步是因为$RSA(ZERO) = ZERO$，并且
$Bytes \oplus ZERO = Bytes$）

所以取后半部分，设
$$encryptZero(plaintext) = RSA(plaintext \oplus RSA(iv)$$

那么同理可得$decryptZero(cipher) = RSA^{-1}(cipher) \oplus RSA(iv)$

那么我们现在就有了四个函数

$$encrypt(plaintext) = RSA(plaintext \oplus iv) \oplus RSA(iv)$$

$$decrypt(cipher) = RSA^{-1}(cipher \oplus RSA(iv)) \oplus iv$$

$$encryptZero(plaintext) = RSA(plaintext \oplus RSA(iv)$$

$$decryptZero(cipher) = RSA^{-1}(cipher) \oplus RSA(iv)$$

又因为

$$
\begin{aligned}
&decrypt(decryptZero(ZERO)) \\
&= RSA^{-1}(RSA^{-1}(ZERO) \oplus RSA(iv) \oplus RSA(iv)) \oplus iv \\
&= RSA^{-1}(RSA^{-1}(ZERO))) \oplus iv \\
&= ZERO \oplus iv \\
&= iv
\end{aligned}
$$

所以netcat以后，实际操作如下

1. 选择decrypt，发送1024个0，也就是512B。
2. 截取接受到后半部分，再次选择decrypt，发送。

```
ubuntu@VM-0-10-ubuntu:~$ nc 47.95.212.185 38610
Choose:
[1] Encrypt (hex)
[2] Decrypt (hex)
2
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
135aceb9c725c34921767bad994ed1b2673a033f989f93679a45585b40a3df65f8205410810921f86d15eedc2ea5b04fcfb7e6c825d66c4f889670cbf83b1ad1
31f07a2a198876fe916e997ee8e3c520f4b84cc5c5757a69d50a8cb4787aba9d385d9fa72a418f4b648958da3d732644eb40fd1834c6f6e33fe208b5f99ea824
1c94c480f9d3dc5697f03740f674832504ae1091530ab63615d6b10473bbbe8cd249f951042e0f1e94f78e9e134c168cb07285ba2d386ff650a24cf93531903f
a92bc17eea66b0a33379c0248daec7a2137b2c2c7830c27773b5dfb362584d6e0bc3c86f03a0d5ddf9bd652043389a630cef7b2e08f80a34a45d0bb356cdc421
Choose:
[1] Encrypt (hex)
[2] Decrypt (hex)
2
135aceb9c725c34921767bad994ed1b2673a033f989f93679a45585b40a3df65f8205410810921f86d15eedc2ea5b04fcfb7e6c825d66c4f889670cbf83b1ad1
31f07a2a198876fe916e997ee8e3c520f4b84cc5c5757a69d50a8cb4787aba9d385d9fa72a418f4b648958da3d732644eb40fd1834c6f6e33fe208b5f99ea824
1c94c480f9d3dc5697f03740f674832504ae1091530ab63615d6b10473bbbe8cd249f951042e0f1e94f78e9e134c168cb07285ba2d386ff650a24cf93531903f
a92bc17eea66b0a33379c0248daec7a2137b2c2c7830c27773b5dfb362584d6e0bc3c86f03a0d5ddf9bd652043389a630cef7b2e08f80a34a45d0bb356cdc421
0000000000006867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f
6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273
343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974
746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d
```

所以截取iv如下

> 0000000000006867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f6861

> 72645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d

正如前面说过的这里的循环体

`6867616d657b415f6c6974746c655f686172645f5273343f7d`

就是flag的ASCII的十六进制，解码得到

`hgame{A_little_hard_Rs4?}`

3. Sign_in_SemiHard

Question

Description

47.95.212.185 38611 仍然是由CNSS友情赞助 大量的心！ **hint:**不要想着一步到位 一段一段来 或者 碰碰运气(如果你比较欧的话

URL

http://plqbnxx54.bkt.clouddn.com/Sign_in_SemiHard.py

Base Score

500

Answer

照样看源码

```
#!/usr/bin/python3
from Crypto.Cipher import AES
from hashlib import md5
```

```
from os import urandom
import string
from signal import alarm


class Sign:
    block = T = 0
    key = salt = ""

    def __init__(self, key, salt):
        self.key = key
        self.salt = salt
        self.block = len(key)

    def register(self, username):
        if b'admin' in username:
            return None
        sig = md5(self.salt + username).digest()
        padlen = self.block - len(username) % self.block
        username += bytes([padlen] * padlen)
        iv = urandom(self.block)
        aes = AES.new(self.key, AES.MODE_CBC, iv)
        c = aes.encrypt(username)
        return iv + c + sig

    def login(self, cipher):
        if len(cipher) % self.block != 0:
            return None
        self.T -= 1
        iv = cipher[:self.block]
        sig = cipher[-self.block:]
        cipher = cipher[self.block:-self.block]
        aes = AES.new(self.key, AES.MODE_CBC, iv)
        p = aes.decrypt(cipher)
```

```
        p = p[:-p[-1]]
        return [p, md5(self.salt + p).digest() == sig]


if __name__ == '__main__':
    unprintable = b""
    for i in range(256):
        if chr(i) not in string.printable:
            unprintable += bytes([i])
    alarm(60)
    s = Sign(urandom(16), urandom(16))
    while True:
        print("Choose:\n[1] Register\n[2] Login")
        op = input()
        if op == '1':
            user = input("Input your username(hex): ")
            token = s.register(bytes.fromhex(user))
            if not token:
                print("Sorry, invalid username.")
            else:
                print("Your token is: %s" % token.hex())
        elif op == '2':
            token = input("Input your token: ")
            res = s.login(bytes.fromhex(token))
            if not res:
                print("Sorry, invalid token.")
            elif not res[1]:
                user = res[0].hex()
                print("Sorry, your username(hex) %s is
inconsistent with given signature." % user)
            else:
                user =
res[0].strip(unprintable).decode("Latin1")
                print("Login success. Welcome, %s!" % user)
```

```
                if user == "admin":
                    print("I have a gift for you: %s" %
FLAG)
        else:
            print("See you")
            break
```

这个看下来就是名副其实的CBC了，而且还是带盐的md5验证。

`register`的时候不许用`admin`，甚至不能包含`admin`。

然后又要`login`的时候，名字去掉头尾的unprintable字符刚刚好是`"admin"`。

那么就要用到CBC的`字节反转攻击`来让最后的名字是admin，

以及对带盐的MD5的进行`哈希长度扩展攻击`。

代码如下

```
import socket

pad = b'e4b8ad'
PAD =
b'e4b8ad8000000000000000000000000000000000000000000000000000
0000000000000000000000980000000000000060646d696e'

START =
b'\xe4\xb8\xad\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00'
ZERO =
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00'
END =
b'\x00\x00\x00\x00\x00\x00\x00\x00\x98\x00\x00\x00\x00\x00\x
00\x00'
```

```
ADMIN =
b'\x61\x64\x6d\x69\x6e\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x
0b\x0b'

targets = [START, ZERO, END, ADMIN]

admin =
b'\x60\x64\x6d\x69\x6e\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x
0b\x0b'.hex()


def xor(a, b): return bytes(x ^ y for x, y in zip(a, b))


def xor_attack(key, origin, target): return xor(target,
xor(bytes.fromhex(origin), bytes.fromhex(key)))


block = 32

hostname = '47.95.212.185'
port = 38611
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((hostname, port))

# 拿到PAD的token
repr(s.recv(1024))
s.sendall(b'1\n')
repr(s.recv(1024))
s.sendall(PAD + b'\n')
token = repr(s.recv(1024)).split(' ')[3].split('\\')[0]

# 字节反转攻击
token = token[:block * 3] + xor_attack(token[block * 3:block
```

```
* 4], admin, ADMIN).hex() + token[block * 4:block * 6]
s.sendall(b'2\n')
repr(s.recv(1024))
s.sendall(token.encode('utf-8') + b'\n')

for i in range(3):
    result = repr(s.recv(1024)).split(' ')[3]
    start = block * (2 - i)
    token = token[:start] \
            + xor_attack(token[start:start + block],
result[start:start + block], targets[2 - i]).hex() \
            + token[start + block:block * 6]
    s.sendall(b'2\n')
    repr(s.recv(1024))
    s.sendall(token.encode('utf-8') + b'\n')

repr(s.recv(1024))

# get md5 of start
s.sendall(b'1\n')
repr(s.recv(1024))
s.sendall(pad + b'\n')
# 这里输出的md5 应该放到hashpump中生成对应的md5
print(repr(s.recv(1024)).split(' ')[3].split('\\')[0][-
block:])

# input md5 from hashpump
s.sendall(b'2\n')
repr(s.recv(1024))
s.sendall(token[:block * 5].encode('utf-8') +
input().encode('utf-8') + b'\n')
print("Received:", repr(s.recv(1024)))
```

由于我没有直接在python用hashpump（安不起来。。。），所以我在倒数第二部输出了md5

然后 在linux里执行

`hashpump -d 中 -k 16 -a admin -s f5e0ad400b2a12944729d217df0860a2`

> PS：这里的`-d 中`是因为hashpump非要我输入一个data，但是按照代码又不能是`string.printable`，所以就弄了个中文，也就是前面那个奇怪的`\xe4\xb8\xad`

假设`f5e0ad400b2a12944729d217df0860a2`就是刚输出的md5

那么会得到

> d5c7a0383be95b611053e20298ba3f10
> \xe4\xb8\xad\x80\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\x00\x00\x00\x00\x00\x00\x00\x00\x98\x00\x00\x00\x00\x00\x00\x00admin

只要把`d5c7a0383be95b611053e20298ba3f10`输回到input里即可

返回得到

> Login success. Welcome, admin!\nI have a gift for you: hgame{hard_cryptooooo!}\nChoose:\n[1] Register\n[2] Login\n

所以flag就是`hgame{hard_cryptooooo!}`