# 第四届江西省高校网络安全技能大赛 复现 2021-09-30

原创

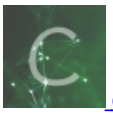
路由(_)生 于 2021-10-01 16:02:47 发布 5003 收藏 23

分类专栏： crypto 文章标签： 网络安全 深度学习



本文链接：https://blog.csdn.net/qq_52193383/article/details/120568283







### 文章目录



## crypto

### Yusa的密码学课堂—CBC第二课

**题目：**

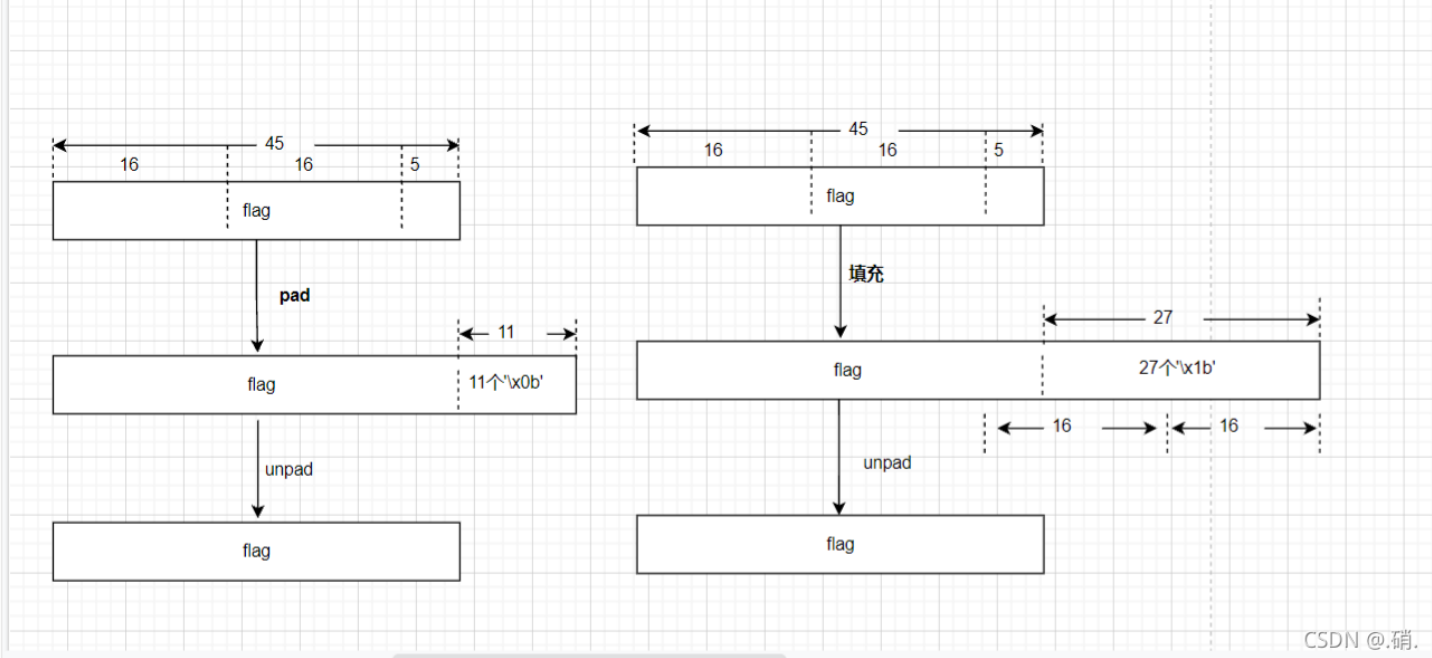
```
from Crypto.Cipher import AES
import os
flag='DASCTF{******************************}'
BLOCKSIZE = 16



def pad(data):
        pad_len = BLOCKSIZE - (len(data) % BLOCKSIZE) if  len(data) % BLOCKSIZE != 0 else 0
        return data + chr(pad_len) * pad_len

def unpad(data):
        num = ord(data[-1])
        return data[:-num]


def _enc(data,key,iv):
 cipher = AES.new(key,AES.MODE_CBC,iv)
 encrypt = cipher.encrypt(pad(data))
 return encrypt
```

```
def enc(data,key):
        try:
                iv = raw_input("Your iv: ").decode('hex')
                cipher = AES.new(key,AES.MODE_CBC,iv)
                encrypt = cipher.encrypt(pad(data))
                return encrypt
        except:
                exit()

def dec(data,key,iv):
 try:
  cipher = AES.new(key,AES.MODE_CBC,iv)
  encrypt = cipher.decrypt(data)
  return unpad(encrypt)
 except:
  exit()


def task():
        try:
                key = os.urandom(16)
                iv = os.urandom(16)
                cipher = _enc(flag,key,iv).encode('hex')
                print cipher
                paintext = raw_input("Amazing function: ").decode('hex')
                print enc(paintext,key).encode('hex')

                backdoor = raw_input("Another amazing function: ")
                assert backdoor != cipher

                if dec(backdoor.decode('hex'),key,iv) == flag:
                        print flag
                else:
                        print "Wow, amazing results."
        except Exception as e:
                print str(e)
                exit()
if __name__ == "__main__":
        task()
```

大致的意思是，服务器会先给我们flag的密文，然后我们输入一个明文，服务器给出对应的密文，之后再输入一个与flag密文不同的密文backdoor，但是backdoor解出的明文必须与flag一样才会给出flag。

我们可以控制的是一个明文paintext，一个密文backdoor 以及一个加密向量IV。

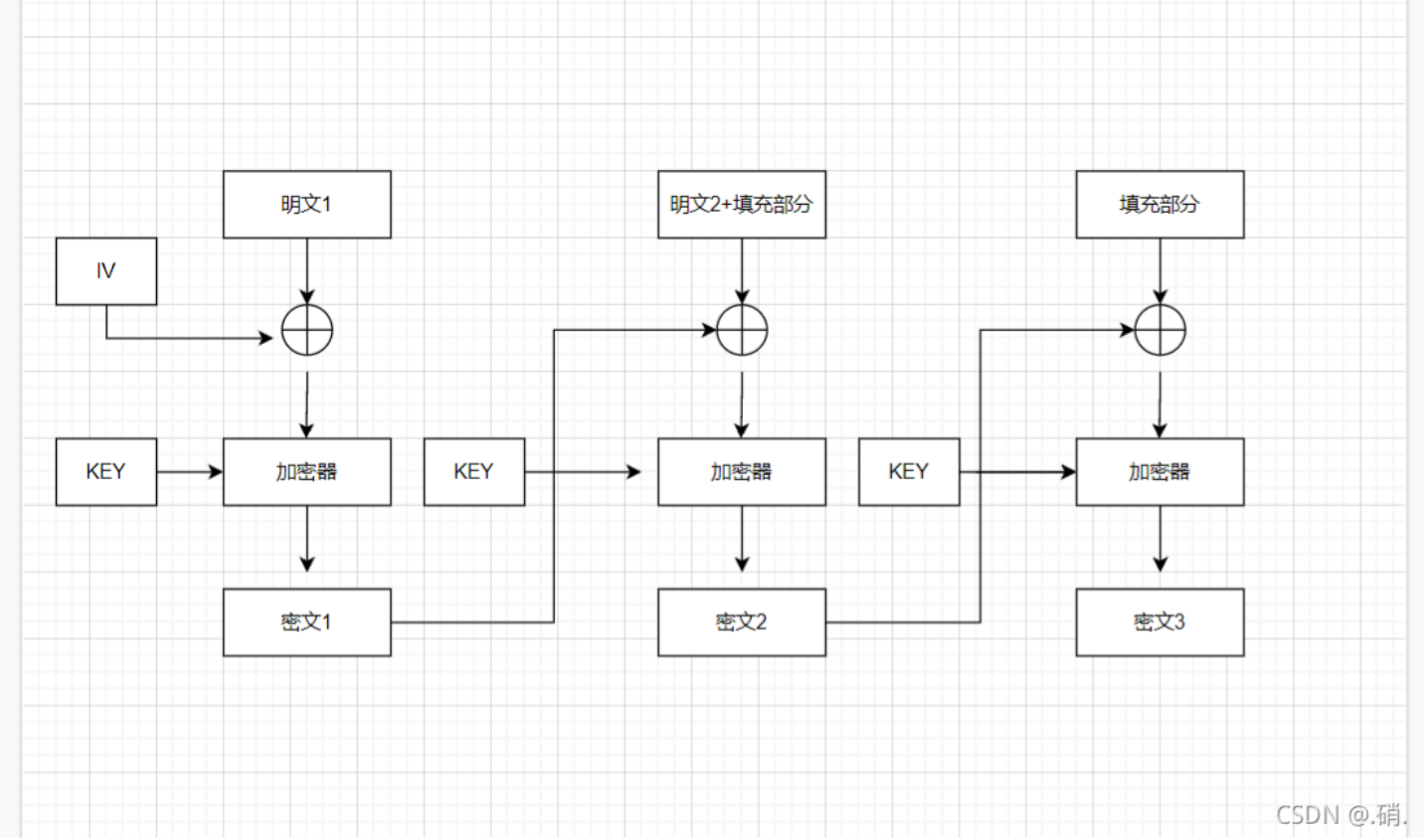
想要让不同的密文解出相同的明文，可能吗？不过这道题特殊的地方在 **pad()** 和 **unpad()** 上，这就是突破口！




所以，我们可以通过改变flag的填充部分（增大填充部分）来使得不同密文能解出相同明文。

那如何使得后来的填充部分跟flag填充完的部分结合在一起呢？

这里我们控制的IV就起到了作用。




从图中可以看出，利用flag密文的后16位作为IV，能使得填充部分跟flag融为一体，而填充的字符的ord() = len(flag)，这样一来，明文填充后就是flag+服务器填充部分+我们自己的填充部分，密文的话就是服务器发送的flag密文+我们自己填充部分的密文,解密后再unpad的结果就是flag。

(大概的意思就是这样，可能思路有点不清晰)

代码：

```python
from pwn import *

context.log_level = 'debug'
#context(os='linux', arch='amd64', log_level='debug')
#os设置系统为linux系统，arch设置架构为amd64，log_level设置日志输出的等级为debug
p = remote('49.233.13.133', '52001')#连接指定地址和端口
cipher = bytes.fromhex(p.recvline()[:-1].decode())#recvline(keepends = True)接收一行，keepends为是否保留行尾的\n
c = cipher[-16:]

p.recvuntil(b'Amazing function:')#recvuntil(delims, drop=False)一直读到delims的pattern出现为止。
pad = chr(8+16).encode() * 16
p.sendline(pad.hex())#sendline(data)发送一行数据，相当于在数据末尾加\n。

p.recvuntil(b'Your iv: ')
p.sendline(c.hex())
cx = bytes.fromhex(p.recvline()[:-1].decode())

p.recvuntil(b'Another amazing function: ')
backdoor = cipher + cx
p.sendline(backdoor.hex())

flag = p.recvline().decode()
print(flag)
```

## Yusa的密码学课堂—CBC第三课

**题目：**

```python
from Crypto.Cipher import AES
import os
flag='DASCTF{******************************}'
BLOCKSIZE = 16



def pad(data):
 pad_len = BLOCKSIZE - (len(data) % BLOCKSIZE) if  len(data) % BLOCKSIZE != 0 else 0
 return data + "=" * pad_len

def unpad(data):
 return data.replace("=","")


def enc(data,key):
 cipher = AES.new(key,AES.MODE_CBC,key)
 encrypt = cipher.encrypt(pad(data))
 return encrypt


def dec(data,key):
 try:
  cipher = AES.new(key,AES.MODE_CBC,key)
  encrypt = cipher.decrypt(data)
  return unpad(encrypt)
 except:
  exit()
def s_2_l(data):#分组
 s=[]
 for i in range(len(data)//BLOCKSIZE):
  s.append(data[BLOCKSIZE*i:BLOCKSIZE*(i+1)])
 return s

def task():
 try:
  key = os.urandom(16)
  asuy = enc(flag,key)
  print asuy.encode('hex')

  paintext = raw_input("Amazing function(in hex): ")
  paintext = paintext.decode('hex')
  print enc(paintext,key).encode('hex')
  asuy = raw_input("Another amazing function(in hex): ").decode('hex')
  yusa = dec(asuy,key)

  flag_l = s_2_l(flag)
  yusa_l = s_2_l(yusa)
  for each in yusa_l:
   if each in flag_l:
    print(r"You're not yusa!")
    exit()
  print yusa.encode('hex')
 except Exception as e:
  print str(e)
  exit()
if __name__ == "__main__":
 task()
```

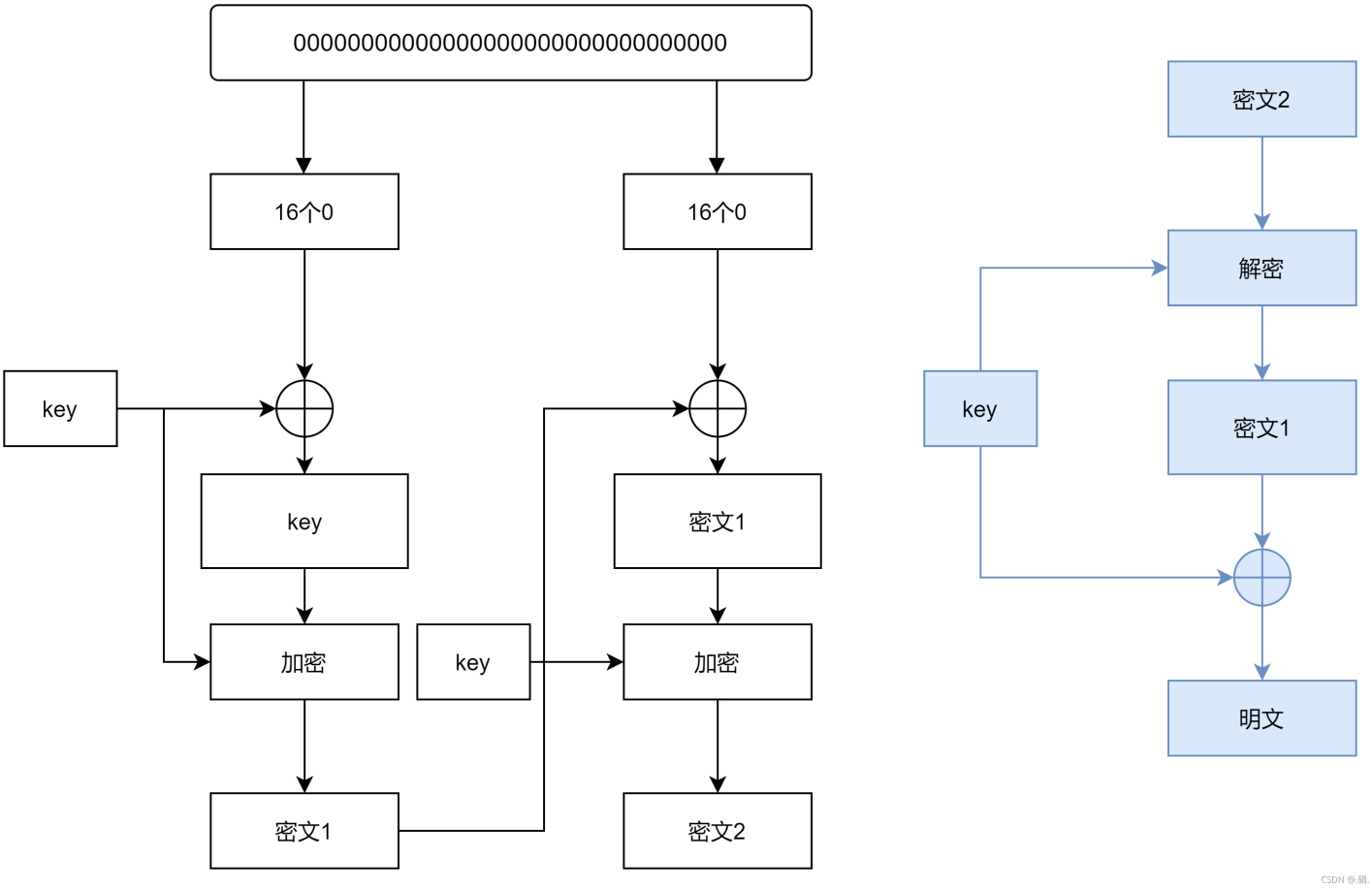
大致的意思是，服务器给flag的密文，我们发送一个明文paintext，收到对应的密文，再发送一个密文asuy，解密得到对应的明文yusa。再将flag和yusa分组，并要求yusa_1中不能含有flag_1的元素，满足就会给出yusa，不满足就退出。

这样看来满不满足都无法求出flag。

注意到这里用到的密钥key和向量IV是相同的。而且我们能控制输入的有一个明文和一个密文。

同样借助图片来理解：

假如我们输入的明文全是由0组成的，借助异或性质就可以消除一些不必要的麻烦，简化问题。




这样就可以很直观的看出 ，利用32个0作为第一次输入的明文，得到的密文的后16位作为第二次输入的密文，解出来的明文 = key ^ 密文1，然后我们就可以求出key，从而求出flag。

代码：

```python
from pwn import *
from Crypto.Cipher import AES

def unpad(s):
    return s.replace(b"=", b"")

context.log_level = 'debug'
sh = remote('49.233.13.133', '51903')

cipher = bytes.fromhex(sh.recvline()[:-1].decode())
c1, c2, c3 = cipher[:16], cipher[16:32], cipher[32:48]

sh.recvuntil(b'Amazing function(in hex): ')
plaintext = b'\x00' * 32
sh.sendline(plaintext.hex())
cx = bytes.fromhex(sh.recvline()[:-1].decode())

sh.recvuntil(b'Another amazing function(in hex): ')
yusa = cx[16:32]
sh.sendline(yusa.hex())

asuy = bytes.fromhex(sh.recvline()[:-1].decode())

keyx = xor(asuy,cx[:16])

aes = AES.new(keyx, AES.MODE_CBC, keyx)
print(aes.decrypt(cipher))
```

# Misc

## 奇奇怪怪的编码

编码1：
```
++++++++[>>++>++++>++++++>++++++++>++++++++++>++++++++++++>++++++++++++++>++++++++++++++++>++++++++++++++++++>++++++++++++++++++++>++++++++++++++++++++++>++++++++++++++++++++++++>++++++++++++++++++++++++++<<<<<<<<<<<<<<<<-]>>>>>>>++++++.>----.<-----.>-----.>-----.<<.+.<<<+++++++.------.>>>+.+.---.<<<.
```

编码2：
```
([][(!![]+[])[!+[]+!+[]+!+[]]+([][[]]+[])[+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(!![]+[])[!+[]+!+[]+!+[]]+(![]+[])[!+[]+!+[]+[]+!+[]]()+[])[!+[]+!+[]]+[!+[]+!+[]+[]+!+[]+!+[]]+[!+[]+!+[]+[]+!+[]+!+[]+!+[]+!+[]+!+[]+!+[]+!+[]+!+[]]+[!+[]+!+[]+[]+!+[]+!+[]]+(!![]+[])[!+[]+!+[]+[]+!+[]]+([][[]]+[])[+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(!![]+[])[!+[]+!+[]+[]+!+[]]+(![]+[])[!+[]+!+[]+[]+!+[]]()+[])[!+[]+!+[]]+[!+[]+!+[]+[]+!+[]]+(![]+[])[+[]]+[!+[]+!+[]]+[+!+[]]
```

编码3：
```
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook! Ook! Ook. Ook? Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook? Ook. Ook? Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook! Ook. Ook? Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook!
Ook! Ook. Ook? Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook? Ook. Ook? Ook! Ook. Ook? Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook! Ook? Ook! Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook? Ook. Ook? Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook! Ook. Ook! Ook! Ook! Ook! Ook! Ook. Ook? Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook! Ook!
Ook. Ook? Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook?
Ook. Ook? Ook! Ook. Ook? Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook. Ook!
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook. Ook. Ook. Ook. Ook.
Ook! Ook. Ook! Ook. Ook! Ook! Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook. Ook! Ook! Ook! Ook!
Ook! Ook. Ook. Ook. Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook! Ook! Ook. Ook?
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook? Ook. Ook? Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook. Ook? Ook.
```

第一个编码是brainfuck，解码得到：

```
flag{ab71cda1
```

第二个编码是jsfuck,解码得到：

```
b495e13b3f21
```

第三个编码是Ook，解码得到：

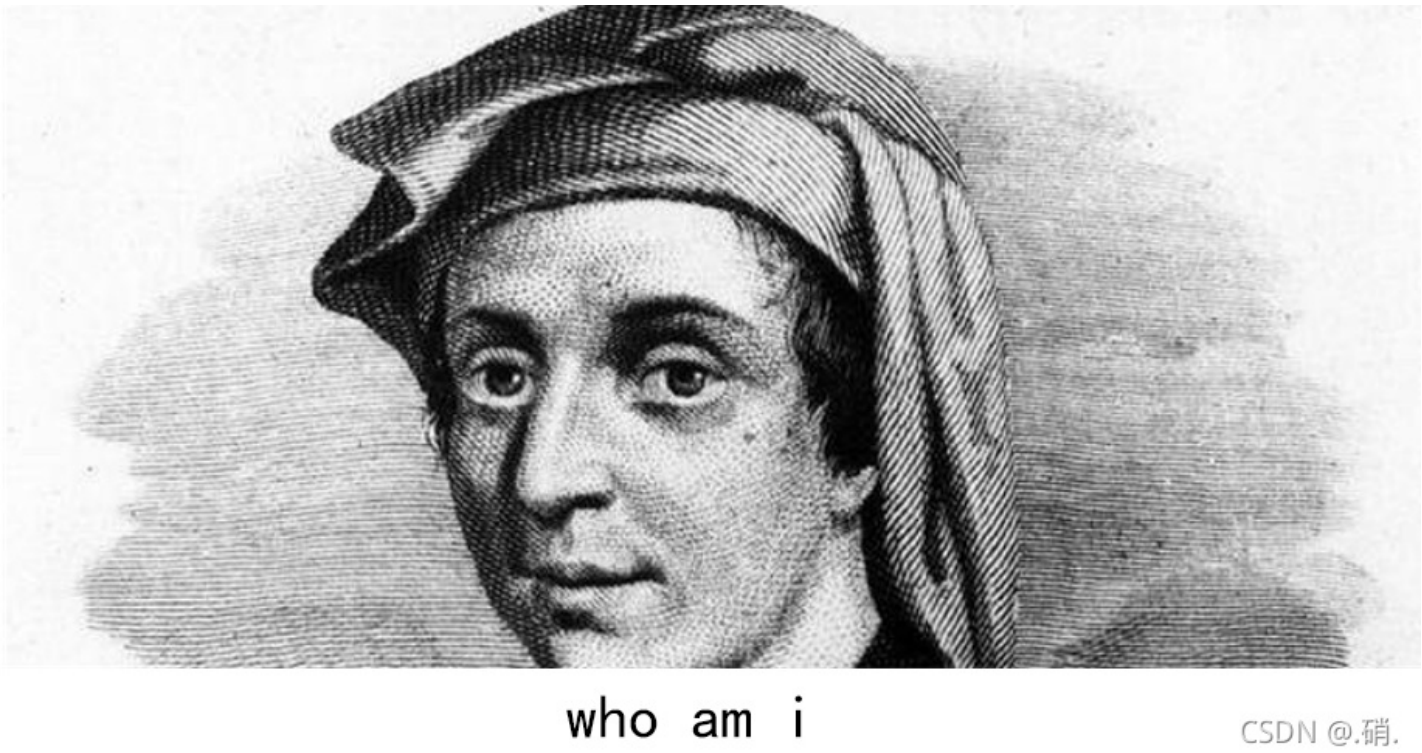
```
f6fd50221978}
```

合起来就是flag：

```
flag{ab71cda1b495e13b3f21f6fd50221978}
```

## extractall

不断解压缩包得到一张图和一个提示，图片高度更改一下：




（他是斐波那契Fibonacci）

```
恭喜你经过21次解压后找到了我，但flag不在这儿，哈哈哈~
```

本人菜，手动依次解压缩包的，到最后发现最后一个文件名为ufQ==]，估摸着密文应该是所有文件名的组合，在进行base64解密。

```
[REFTQ
1RGe0V
4dHJhY
eht8on
3RhbGx
8smjtq
zmckit
fSXNfU
9rskp5
a93su6
al0o68
p0l2vq
29fRnV
fco9e2
7ztjka
bvn8ta
a27s40
dxzk1l
yq6ik4
gec9bl
ufQ==]
```

不过将这些组合起来去解密并不能解出来，需要与斐波那契数列联系起来。

斐波那契数列前几项：

```
0 1 1 2 3 5 8 13 21 34 55
```

根据斐波那契数列提取文件名：

```
REFTQ1RGe0V4dHJhY3RhbGxfSXNfU29fRnVufQ==
```

解码后得到：

```
DASCTF{Extractall_Is_So_Fun}
```

参考：
https://blog.csdn.net/m0_49109277/article/details/120199397