2019强网杯crypto writeup

原创

a16511232 于 2019-05-27 20:21:00 发布 3829 收藏 1

本次write包含以下题目

- copperstudy
- randomstudy
- 强网先锋-辅助

---

copperstudy

题目描述

```
nc 119.3.245.36 12345
```

连上去返回

```
[+]proof: skr=os.urandom(8)
[+]hashlib.sha256(skr).hexdigest()=1651d7f27f35e8f177633a014dbc4eb04a99e74c09a7545d1c1f08e0f8b7f7c3
[+]skr[0:5].encode('hex')=58d0f5f7fc
[-]skr.encode('hex')=
```

破解一个原文8个字符的sha256，已知前5个字符
可以使用hashcat爆破比较快(10秒内)，也可以自己写python破解

```
#python3
#path修改为本地hashcat文件夹路径
import os
path="D:\\hashcat-5.1.0\\"
str1=input("[+]hashlib.sha256(skr).hexdigest()=")
str2=input("[+]skr[0:5].encode('hex')=")
print(path+f"hashcat64.exe -a 3 --hex-salt -m 1420 {str1}:{str2} --potfile-disable ?b?b?b  -o res.txt --out
os.chdir(path)
os.system(path+f"hashcat64.exe -a 3 --hex-salt -m 1420 {str1}:{str2} --potfile-disable ?b?b?b  -o res.txt -
with open(path+"res.txt", 'r') as f:
    lines = f.readlines()
    last_line = lines[-1]
    #print(last_line)
    if(last_line[0:4]=="$HEX"):
        print(str2+last_line[5:11])
    else:
        print(str2+hex(last_line))
```

第一问

```
[+]Generating challenge 1
[+]n=0x20918380c97c52ae96b53b371e822a972927d4cc96e1659a52ff4ba6849fd16856959ae83a91d2c0775fb06de75c5117c82f
[+]e=3
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0x1e494321effffbaf32beebe37f042db1baeb12f42740ffe43c8b7d8e18b79170208231610e8f2f65ff85e8f5c
[+]((m>>72)<<72)=0x258aa871ef627602f03f2100efe0c045d4a307153ca71f26e22fc2bcb72a16ae695d5bad767d258aa42e02a5
[-]long_to_bytes(m).encode('hex')=
```

一开始没有思路，看到题目叫做copperstudy，搜索关键字ctf rsa copper，果然搜出东西，有一位叫Coppersmith对RSA提出了很多种攻击算法。
题外话，搜索是有技巧的，通常题目的解法不是第一次出现，利用题目的标题/hint/部分关键代码，加上ctf或writeup关键字，通常能搜到点有用的东西。
参考链接
强网杯2018 - nextrsa - Writeup
CTF中常见的RSA相关问题总结
RSA大礼包（二）Coppersmith 相关

第一问的解法
coppersmith攻击，已知m的高位

```
#sage脚本
load("coppersmith.sage")
N = #N的值
e = 3 #e的值
m = #m的大概值
c = #c的值
ZmodN = Zmod(N)
P. = PolynomialRing(ZmodN)
f = (m + x)^e - c
dd = f.degree()
beta = 1
epsilon = beta / 7
mm = ceil(beta**2 / (dd * epsilon))
tt = floor(dd * mm * ((1/beta) - 1))
XX = ceil(N**((beta**2/dd) - epsilon))
roots = coppersmith_howgrave_univariate(f, N, beta, mm, tt, XX)
```

第二问

```
[++++++++++++++++]challenge 1 completed[++++++++++++++++]
[+]Generating challenge 2
[+]n=0x4ac5cbf84a2f9a1042c552c77075459d2273994453caea11fbf696b9a8d41937b48be43c71ec6c37470ba9d280a23301b817
[+]e=65537
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0x19304208b44ce0bf457d757e368edde74922e89a51290937a386cb320e9c59bb80f77c4a4f1d1b0699593dc1c
[+]((p>>128)<<128)=0x80f7a73798f638d10180223d7b482035b69b51ffe09ad9e42602cc9d489837be7d1ac92e90b09837144c12
```

还是coppersmith攻击，已知因子p的高位

```
load("coppersmith.sage")
N = 0x4ac5cbf84a2f9a1042c552c77075459d2273994453caea11fbf696b9a8d41937b48be43c71ec6c37470ba9d280a23301b8173
ZmodN = Zmod(N)
P.<x> = PolynomialRing(ZmodN)
qbar = 0x80f7a73798f638d10180223d7b482035b69b51ffe09ad9e42602cc9d489837be7d1ac92e90b09837144c1220ed4ff0ea00
f = x - qbar
beta = 0.5
dd = f.degree()
epsilon = beta / 7
mm = ceil(beta**2 / (dd * epsilon))
tt = floor(dd * mm * ((1/beta) - 1))
XX = ceil(N**((beta**2/dd) - epsilon)) + 10000000000000000000000000000000
roots = coppersmith_howgrave_univariate(f, N, beta, mm, tt, XX)
```

第三问

```
[+++++++++++++++]challenge 2 completed[+++++++++++++++]
[+]Generating challenge 3
[+]n=0x6076ea10cc4cef8ceb867f3958946426d25fb06a9d3192d55390bd5611664432bf57d8e2c50cbb897e6086d185145b0af11e
[+]e=3
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0x27fcc420e465972031f4ef78bed383aa40af28f940a15b8366d640653241e26cc0cfe9f7df9e884b68c259456
[+]d=invmod(e,(p-1)*(q-1))
[+]d&((1<<512)-1)=0xfd7a028dfde00006c3c94b076e29b9786800722872f5ffabe50df2eac3766d801903baafa26eab26e5aa7c9
[-]long_to_bytes(m).encode('hex')=
```

coppersmith攻击，已知d的低位

```
#sage脚本
def partial_p(p0, kbits, n):
    PR.<x> = PolynomialRing(Zmod(n))
    nbits = n.nbits()

    f = 2^kbits*x + p0
    f = f.monic()
    roots = f.small_roots(X=2^(nbits//2-kbits), beta=0.3)  # find root = n^0.3
    if roots:
        x0 = roots[0]
        p = gcd(2^kbits*x0 + p0, n)
        return ZZ(p)

def find_p(d0, kbits, e, n):
    X = var('X')

    for k in xrange(1, e+1):
        results = solve_mod([e*d0*X - k*X*(n-X+1) + k*n == X], 2^kbits)
        for x in results:
            p0 = ZZ(x[0])
            p = partial_p(p0, kbits, n)
            if p:
                return p


n = 0x6076ea10cc4cef8ceb867f3958946426d25fb06a9d3192d55390bd5611664432bf57d8e2c50cbb897e6086d185145b0af11ea
e = 3
d = 0xfd7a028dfde00006c3c94b076e29b9786800722872f5ffabe50df2eac3766d801903baafa26eab26e5aa7c90a7d0540d43cdb

beta = 0.5
epsilon = beta^2/7

nbits = n.nbits()
kbits = 511
d0 = d & (2^kbits-1)
print "lower %d bits (of %d bits) is given" % (kbits, nbits)

p = find_p(d0, kbits, e, n)
print "found d: %d" % p
q = n//p
print d
print inverse_mod(e, (p-1)*(q-1))
```

第四问

```
[+++++++++++++++]challenge 3 completed[+++++++++++++++]
[+]Generating challenge 4
[+]e=3
[+]m=random.getrandbits(512)
[+]n1=0x8782029df97597a52ae2657b3e2bd6e13abb98c66c232b58ebaf4a0840e21d92a3d2c96d2168bcf6e23a2e7ac21438ee331
[+]c1=pow(m,e,n1)=0x44ae824c4704561e3d726ec074ed549b00873d2332e40619ed7b986f467d032e536e3be84a801026aa9d830
[+]n2=0x28abcf38a463248ed10564e1709a8c4345d5a0b627d5fa608e9c7280d1aa60f209ff179fd4e7298e4c39ed9539a06103317
[+]c2=pow(m,e,n2)=0x1dfcfb67c84c6a0a701aaacbc429b3d67ba804bd9d8401a1e9b46f7ff4f20cc94f6156d71128bc081ad9e45
[+]n3=0x3c2a8de96e6e1b8f15c9b01abfdf3027d78eef0c77396cd1e676afc0a4ba0e3ba6b194387e0fbed1296bf523222e4aae43c
[+]c3=pow(m,e,n3)=0x16c7c51ebd1e420ef58c4c675c0528043f021af620e0e57cc781714c737684df21782295fc343d44a52b938
[-]long_to_bytes(m).encode('hex')=
```

同一个密文m，使用相同的较小的e=3，不同的n加密
低加密指数广播攻击

```
#python
from gmpy2 import invert, iroot
e=3
n1=0x8782029df97597a52ae2657b3e2bd6e13abb98c66c232b58ebaf4a0840e21d92a3d2c96d2168bcf6e23a2e7ac21438ee331961
c1=0x44ae824c4704561e3d726ec074ed549b00873d2332e40619ed7b986f467d032e536e3be84a801026aa9d830a999c3bb56a694d
n2=0x28abcf38a463248ed10564e1709a8c4345d5a0b627d5fa608e9c7280d1aa60f209ff179fd4e7298e4c39ed9539a06103317405
c2=0x1dfcfb67c84c6a0a701aaacbc429b3d67ba804bd9d8401a1e9b46f7ff4f20cc94f6156d71128bc081ad9e457fc9fba53cda2c2
n3=0x3c2a8de96e6e1b8f15c9b01abfdf3027d78eef0c77396cd1e676afc0a4ba0e3ba6b194387e0fbed1296bf523222e4aae43c229
c3=0x16c7c51ebd1e420ef58c4c675c0528043f021af620e0e57cc781714c737684df21782295fc343d44a52b938788438b8880c93b

def broadcast(n1, n2 ,n3, c1, c2, c3):
    n = [n1, n2, n3]
    C = [c1, c2, c3]
    N = 1
    for i in n:
        N *= i
    Ni = []
    for i in n:
        Ni.append(N / i)
    T = []
    for i in xrange(3):
        T.append(long(invert(Ni[i], n[i])))
    X = 0
    for i in xrange(3):
        X += C[i] * Ni[i] * T[i]
    m3 = X % N
    m = iroot(m3, 3)
    return m[0]

print(hex(broadcast(n1, n2 ,n3, c1, c2, c3)))
```

第五问

```
[+++++++++++++++]challenge 4 completed[+++++++++++++++]
[+]Generating challenge 5
[+]n=0x2030ca024a23fb978752ccc2897947fd9c82b682915771e447fc1eefa6be8cbcc00df7cc2dfc401516b88b06a044b6fa595c
[+]e=3
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0x45204e3e2d780d6fded3ed4c53ca2a0300a78bd7f9b30afb5e3267bcb7074756ab386a165cf0678e3af272151
[+]x=pow(m+1,e,n)=0x1c17423e4aa0ee916c513f9f6f7f7f6efda060974ad06282bd846a50571c2b26465aba50a48eda745b0bb5b
[-]long_to_bytes(m).encode('hex')=
```

高位相同的m
短填充攻击Coppersmith Shortpad Attack
参考脚本https://github.com/ValarDragon/CTF-Crypto/blob/master/RSA/FranklinReiter.sage

```
#sage
def franklinReiter(n,e,r,c1,c2):
    R.<X> = Zmod(n)[]
    f1 = X^e - c1
    f2 = (X + r)^e - c2
    # coefficient 0 = -m, which is what we wanted!
    return Integer(n-(compositeModulusGCD(f1,f2)).coefficients()[0])

  # GCD is not implemented for rings over composite modulus in Sage
  # so we do our own implementation. Its the exact same as standard GCD, but with
  # the polynomials monic representation
def compositeModulusGCD(a, b):
    if(b == 0):
        return a.monic()
    else:
        return compositeModulusGCD(b, a % b)

def CoppersmithShortPadAttack(e,n,C1,C2,eps=1/30):
    """
    Coppersmith's Shortpad attack!
    Figured out from: https://en.wikipedia.org/wiki/Coppersmith's_attack#Coppersmith.E2.80.99s_short-pad_at
    """
    P.<x,y> = PolynomialRing(ZZ)
    ZmodN = Zmod(n)
    g1 = x^e - C1
    g2 = (x+y)^e - C2
    res = g1.resultant(g2)
    P.<y> = PolynomialRing(ZmodN)
    # Convert Multivariate Polynomial Ring to Univariate Polynomial Ring
    rres = 0
    for i in range(len(res.coefficients())):
        rres += res.coefficients()[i]*(y^(res.exponents()[i][1]))

    diff = rres.small_roots(epsilon=eps)
    recoveredM1 = franklinReiter(n,e,diff[0],C1,C2)
    print(recoveredM1)
    print("Message is the following hex, but potentially missing some zeroes in the binary from the right e
    print(hex(recoveredM1))

eps=1/25
e=3
n=0x2030ca024a23fb978752ccc2897947fd9c82b682915771e447fc1eefa6be8cbcc00df7cc2dfc401516b88b06a044b6fa595ce67
c1=0x45204e3e2d780d6fded3ed4c53ca2a0300a78bd7f9b30afb5e3267bcb7074756ab386a165cf0678e3af272151b0635c784df30
c2=0x1c17423e4aa0ee916c513f9f6f7f7f6efda060974ad06282bd846a50571c2b26465aba50a48eda745b0bb5b410d0b89a199256
CoppersmithShortPadAttack(e,n,c1,c2,eps)
```

第六问

```
[+++++++++++++++]challenge 5 completed[+++++++++++++++]
[+]Generating challenge 6
[+]n=0xbadd260d14ea665b62e7d2e634f20a6382ac369cd44017305b69cf3a2694667ee651acded7085e0757d169b090f29f3f86fe
[+]d=random.getrandbits(1024*0.270)
[+]e=invmod(d,phin)
[+]hex(e)=0x11722b54dd6f3ad9ce81da6f6ecb0acaf2cbc3885841d08b32abc0672d1a7293f9856db8f9407dc05f6f373a2d92467
[+]m=random.getrandbits(512)
[+]c=pow(m,e,n)=0xe3505f41ec936cf6bd8ae344bfec85746dc7d87a5943b3a7136482dd7b980f68f52c887585d1c7ca099310c4d
[-]long_to_bytes(m).encode('hex')=
```

e很大，d<N的0.292次方
使用Boneh and Durfee attack
参考https://cryptologie.net/article/241/implementation-of-boneh-and-durfee-attack-on-rsas-low-private-exponents/
https://github.com/mimoo/RSA-and-LLL-attacks/blob/master/boneh_durfee.sage

这题脚本太长了，不贴了，跟github的几乎一样，解得d=77676545508179537711737768020951023488723012931857506338263459335772499820757

```
总结
第一问答案
258aa871ef627602f03f2100efe0c045d4a307153ca71f26e22fc2bcb72a16ae695d5bad767d258aa42e02a5c6b270a8c355c8a3588
第二问答案
5f3cef264c352ae60757725eb56469e601e740f88e6e62ad26aecb8623fa4d66a5cb135b759bc0eccafca69f1d2f9899ca207f15ae8
第三问答案
e862dec14b008fe307046a787af4f72865680fb141c734681fc923dc7206bb67693ed08146057d708eb3666b544b4a3386fffb241f0
第四题答案
dc419a862dd40cdc94e251f21c95fac5c28e3e86d629d748e881de3a5ea022c238b87a28889b506c22d488d3bdd6a2c6810087140c3
第五问答案
2d09b4627af9e9c4e118386ce6062c62638d7208f894d2117b3fc910af5bec88ffd84d35e965411cb4f2095a5b39e9e4eb2d13df5a8
第六问答案
6b3bb0cdc72a7f2ce89902e19db0fb2c0514c76874b2ca4113b86e6dc128d44cc859283db4ca8b0b5d9ee35032aec8cc8bb96e8c115
```

几乎都是coppersmith相关的攻击，第一次接触，没想到RSA还有这么多讲究，稍微不注意就可能被解密

---

randomstudy

```
import random
import time
import subprocess

def bye():
    print "[+]bye~"
    sys.exit()

def challenge1():
    print "[+]Generating challenge 1"
    random.seed(int(time.time()))
    for i in range(200):
        recv=int(raw_input("[-]"))
        if recv==random.randint(0,2**64):
            print "[+++++++++++++++++]challenge 1 completed[+++++++++++++++++]"
            return
        else:
            print "[+]failed"
    bye()


def challenge2():
    print "[+]Generating challenge 2"
    for i in range(200):
        o = subprocess.check_output(["java", "Main"])
        tmp=[]
        for i in o.split("\n")[0:3]:
            tmp.append(int(i.strip()))
        v1=tmp[0] % 0xffffffff
        v2=tmp[1] % 0xffffffff
        v3=tmp[2] % 0xffffffff
        print "[-]"+str(v1)
        print "[-]"+str(v2)
        v3_get=int(raw_input("[-]"))
        if v3_get==v3:
            print "[+++++++++++++++++]challenge 2 completed[+++++++++++++++++]"
            return
        else:
            print "[+]failed"
    bye()

def challenge3():
    print "[+]Generating challenge 3"
    for i in range(1000):
        a=raw_input("[-]")
        target=random.getrandbits(32)
        if a!=str(target):
            print "[+]failed:"+str(target)
        else:
            print "[+++++++++++++++++]challenge 3 completed[+++++++++++++++++]"
            return
    bye()
```

nc 119.3.245.36 23456

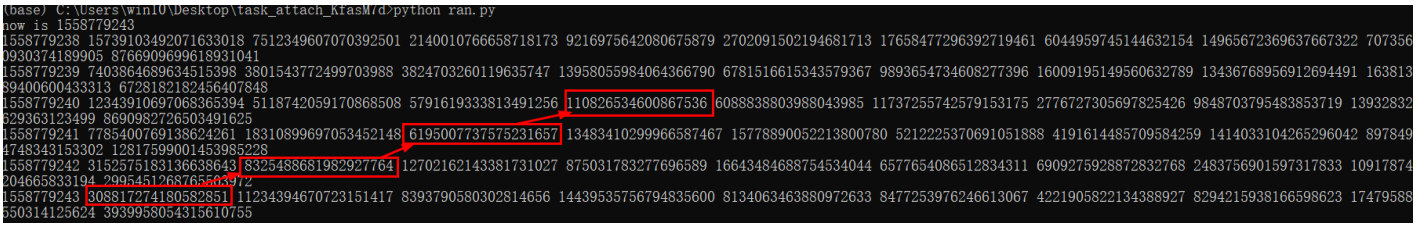
第一步也是要破解一个sha256，跟上题的脚本一样

第一问

python生成随机数
由于`random.seed(int(time.time()))`中
时间可以猜测，故随机数种子可以猜测，则随机数可以猜测
输入队伍token前先等等，同时本地一起执行脚本，服务器时间和我本机时间相差约1到2秒

```
#python3
import random
import time

t = int(time.time())
print(f'now is {t}')
for i in range(-5,1):
    print(t+i,end=' ')
    random.seed(t+i)
    for j in range(10):
        print(random.randint(0,2**64),end=' ')
    print('')
```


(base) C:\Users\win10\Desktop\task_attach_KfasM7d>python ran.py
now is 1558779243
1558779238 1573910349207163301 8 751234960707039250 1 214001076665871817 3 921697564208067587 9 270209150219468171 3 176584772963927194 61 604495974514463215 4 149656723696376673 22 707356
0930374189905 8766909699618931041
1558779239 7403864689634515398 3801543772499703988 3824703260119635747 1395805598406436679 0 6781516615343579367 9893654734608277396 1600919514956063278 9 134367689569126944 91 163813
8940600433313 6728182182456407848
1558779240 1234391069706836539 4 511874205917086850 8 579161933381349125 6 110826534600867536 6088838803988043985 1173725574257915317 5 2776727305697825426 9848703795483853719 1393283 2
529363123499 8690982726503491625
1558779241 7785400769138624261 1831089969705345214 8 619500773757523165 7 134834102999665874 67 157788900522138007 80 521222537069105188 8 419161448570958425 9 141403310426529604 2 897849
4748343153302 1281759900145398522 8
1558779242 3152575183136638643 8325488681982927764 1270216214338173102 7 875031783277696589 1664348468875453404 4 657765408651283431 1 690927592887283276 8 248375690159731783 3 109178 74
204665833194 2995451268765503972
1558779243 3088172741805828511 1123439467072315141 7 839379058030281465 6 144395357567948356 00 813406346388097263 3 847725397624661306 7 422190582213438892 7 829421593816659862 3 174795 88
550314125624 3939958054315610755
```

时间脚本

先猜测最后一行第二列，再猜测倒数第二行第三列，以此类推，同时记录猜测次数和猜对后的时间种子，第三问要用

第二问

java的随机数破解

```
[++++++++++++++++]challenge 1 completed[++++++++++++++++]
[+]Generating challenge 2
[-]44003964
[-]977222536
```

同时附件里有份Main.class，可知是python调用java运行了Main.class
用http://www.javadecompilers.com在线反编译，可以得到源码

```
import java.util.Random;

public class Main { public Main() {}
  public static void main(String[] paramArrayOfString) {
    Random localRandom = new Random();
    System.out.println(localRandom.nextInt());
    System.out.println(localRandom.nextInt());
    System.out.println(localRandom.nextInt());
  }
}
```
```

参考这篇文章
https://jazzy.id.au/2010/09/20/cracking_random_number_generators_part_1.html
代码参考
https://github.com/sacundim/cracking-prngs/blob/master/src/main/java/org/casillas/Main.java

题目给出前两个随机数，要求猜测第三个
在java中，seed是48位，nextInt()返回32位，如果能拿到两个nextInt()，就能反推出seed
脚本如下，修改变量v1，v2
因为题目给的数字经过了转化，类似于int转unsigned int，要先转换一下再带入脚本，或者看运气，等给出了两个比较小的int(捂脸)

```
a=977222536
print(a if a<0x7fffffff else -1*(a&0x7fffffff))
```

```
package org.casillas;

import java.util.Random;

/**
 * A simple program demonstrating how easy it is to predict the output of {@link java.util.Random}.  Based
 * code here, but refactored a bit, and updated to Java 8:
 *
 * <ul>
 *     <li>https://jazzy.id.au/2010/09/20/cracking_random_number_generators_part_1.html</li>
 * </ul>
 */
public class Main {
    // These three values come from `java.util.Random`.  Note that these changed between Java 6 and 7.
    private static final long multiplier = 0x5DEECE66DL;
    private static final long addend = 0xBL;
    private static final long mask = (1L << 48) - 1;

    public static void main(String[] args) {
        Random target = new Random();
        int v1 = target.nextInt();
        int v2 = target.nextInt();
        long seed = guessSeed(v1, v2);
        System.out.printf("Guessed seed: 0x%016x.  Testing...\n", seed);
        testGuess(seed, target, v2);
        System.out.println("Success!!!");
    }


    /**
     * We can guess the state of the generator just from observing two consecutive {@code int}s.
     */
    private static long guessSeed(int v1, int v2) {
        System.out.printf("v1 = 0x%08x, v2 = 0x%08x\n", v1, v2);
        return guessSeed(maskToLong(v1), maskToLong(v2));
    }

    private static long guessSeed(long v1, long v2) {
        for (int i = 0; i < 65536; i++) {
            long guess = v1 * 65536 + i;
            if ((((guess * multiplier + addend) & mask) >>> 16) == v2) {
                return (guess ^ multiplier) & mask;
```

```
            }
        }
        throw new IllegalStateException("CAN'T HAPPEN: Could not guess the seed!");
    }

    /**
     * Convert an int to a long in a way that preserves all the bits (negative numbers/twos complement issu
     */
    private static long maskToLong(int n) {
        return n & 0x00000000ffffffffL;
    }


    private static void testGuess(long seed, Random target, int expected) {
        Random experiment = new Random(seed);
        int actual = experiment.nextInt();
        for (int i = 0; i < 100_000; i++) {
            if (actual != expected) {
                // This will never actually happen.
                String msg = "i = %d, expected = 0x%08x, actual = 0x%08x";
                throw new IllegalStateException(String.format(msg, i, expected, actual));
            }
            expected = target.nextInt();
            actual = experiment.nextInt();
        }
    }

}
```

同样，得到的数字可能是负数，转换一下

```
print(-2145376347 % 0xffffffff)
```

第三问

比第一问简单，就是要记录一下第一问的情况
输入第一问得到的种子，并输入第一问时的尝试次数

```
import random
import time

s = input("seed=")
random.seed(int(s))
ci = input("cishu=")
for i in range(int(ci)):
    print(random.randint(0,2**64),end=',')
print('')
print(random.getrandbits(32))
```

强网先锋-辅助

```
flag=open("flag","rb").read()

from Crypto.Util.number import getPrime,bytes_to_long
p=getPrime(1024)
q=getPrime(1024)
e=65537
n=p*q
m=bytes_to_long(flag)
c=pow(m,e,n)
print c,e,n

p=getPrime(1024)
e=65537
n=p*q
m=bytes_to_long("1"*32)
c=pow(m,e,n)
print c,e,n

'''
output:
24820838937466182485444267370237504001245434520824363343985049860235017106394020609491066932794628969688390
38290600395720427374966791868810679503289561331636299088723481081601295504376976771505994839239257982243281
'''
```

相比起copperstudy，这就是送分题了
两个n1，n2使用了同一个因子p，求一下n1，n2的最大公约数就得到p了
q=n1/p

```
import gmpy2
import libnum
c1=24820838937466182485444267370237504001245434520824363343985049860235017106394020609491066932794628969688
e1=65537
n1=14967030059975114950295399874185047053736587880127990542035765201425779342430662517765063258784685868107
c2=38290600395720427374966791868810679503289561331636299088723481081601295504376976771505994839239257982243
e2=65537
n2=14624662628725820618622370803948630854094687814338334827462870357582795291844925274690253604919535785934

p=gmpy2.gcd(n1,n2)
assert n1%p==0
q1=n1/p
q2=n2/p


phin1=(p-1)*(q1-1)
d1=gmpy2.invert(e1,phin1)
m1=pow(c1,d1,n1)
print(libnum.n2s(m1))
```

flag{i_am_very_sad_233333333333}

还有两道区块链和两道lfsr