python解二元二次方程_BUUCTF平台Crytpo部分Writeup

weixin_39763953 于 2020-12-03 18:21:44 发布 1018 收藏
文章标签： python解二元二次方程

[BJDCTF 2nd]rsa1

题目链接(buu平台)

题解

首先拿到题目靶机。nc交互获取得到题目数据

得到的条件有

e

p^2+q^2

p-q

c

明显是一个rsa加密。要去求解这个题目。因为求解rsa嘛。我们本质上肯定是想通过最基础的rsa解密去求解的。也就是我们要获取到私钥d以及公钥n。

这边我们通过去求解p和q的值。来求解rsa加密。

根据已知信息

假设

p^2+q^2=a

p-q=b

其中a、b已知

即求解二元二次方程

求解方程可以使用z3-solver进行求解

安装方法如下

找到官网

下载对应系统的第三方库文件

例如我是windows系统，我下载的就是

z3_solver-4.8.8.0-py2.py3-none-win_amd64.whl

然后pip安装这个z3-solver库

pip install z3_solver-4.8.8.0-py2.py3-none-win_amd64.whl

然后求解出p和q的值进行一个基础的rsa解密就好了。

题解exp如下

```
import z3

import gmpy2

from Crypto.Util.number import *

#公钥e

e=11630623

#p^2+q^2

p2q2=15909182389345558287406192115461241526732647651294905495639698072728275609950042360

#p-q

p_q=-
10280575127345266887562552539261933088876569047406331328665081651581615559232136816943346

#密文c

c=c=16828607056351898528593346827961041518302251064718691085202163294074947863618526294 6

#使用z3对二元二次方程进行求解

s=z3.Solver()

#定义两个变量

p,q=z3.Ints("p q")

#添加方程

s.add(p*p+q*q==p2q2)

s.add(p-q==p_q)

s.add(p>0)

arr=[]

#校验是否有解

if s.check()==z3.sat:

arr=s.model()

print(arr)

#得到p，q

q =
941805517054390346866216927829557738940927623477259047618501401680307530988625412344565 2
```

p =
838999765780937677990591402436938408052161933003195734331850585164491375396304044175130 6

#简单验证一下

assert(p*p+q*q==p2q2)

assert(p-q==p_q)

#rsa基础解密

n=p*q

phi=(p-1)*(q-1)

d=gmpy2.invert(e,phi)

m=pow(c,d,n)

flag=long_to_bytes(m)

print(flag)

EasyProgram

题目链接(buu平台)

拿到题目下载附件，得到两个文件

file.txt

用python读入字节流，判断长度为38.长度为38我们知道flag{}加上中间32位md5值就是一个标准的flag.也就先判断file.txt存放的是flag的密文

附件.txt

一个加密算法，大概是类似伪代码之类的。也比较容易看懂

get buf unsign s[256]

get buf t[256]

we have key:whoami

we have flag:????????????????????????????????

for i:0 to 256

set s[i]:i

for i:0 to 256

set t[i]:key[(i)mod(key.lenth)]

for i:0 to 256

set j:(j+s[i]+t[i])mod(256)

swap:s[i],s[j]

```
for m:0 to 38
set i:(i + 1)mod(256)
set j:(j + S[i])mod(256)
swap:s[i],s[j]
set x:(s[i] + (s[j]mod(256))mod(256))
set flag[m]:flag[m]^s[x]
fprint flagx to file
```

简单的把代码理一下，改成python的实现

```
key="whoami"
flag="flag{test_test_test_test_test_test_te}"
s=[]
t=[]
flag_enc=""
for i in range(256):
s.append(i)
for i in range(256):
t.append(ord(key[i%len(key)]))
j=0
for i in range(256):
j=(j+s[i]+t[i])%256
tmp=s[i]
s[i]=s[j]
s[j]=tmp
i=0
j=0
for m in range(38):
i=(i+1)%256
j=(j+s[i])%256
tmp=s[i]
s[i]=s[j]
s[j]=tmp
```

x=(s[i]+s[j]%256)%256

flag_enc+=chr(ord(flag[m])^s[x])

print(flag_enc)

简单分析一下，就是一个最简单的流密码，可以把前面的操作看作生成了一个密钥流。这个密钥流明显是固定的(也就是说和加密的明文无关)。那就很简单，异或解密就可以了。

题解exp如下：

```
f=open("file.txt","rb")

st=f.read()

print(len(st))

flag_enc=st

key="whoami"

s=[]

t=[]

flag=""

for i in range(256):

s.append(i)

for i in range(256):

t.append(ord(key[i%len(key)]))

j=0

for i in range(256):

j=(j+s[i]+t[i])%256

tmp=s[i]

s[i]=s[j]

s[j]=tmp

i=0

j=0

for m in range(38):

i=(i+1)%256

j=(j+s[i])%256

tmp=s[i]

s[i]=s[j]
```

s[j]=tmp

x=(s[i]+s[j]%256)%256

flag+=chr(flag_enc[m]^s[x])

print(flag)

[AFCTF2018]可怜的RSA

题目链接(buu平台)

拿到附件：

flag.enc

public.key

就明显是拿到两个附件，一个是RSA加密的密钥文件，还有一个是flag加密后的密文的文件。

这边的话一种就是常规的用openssl，个人不太喜欢用。python能解决的问题我一般就python解决了。这边用到python的Crypto库，是做CTF密码学非常常用的一个库。

pip install pycrypto

安装好这个Crypto库

然后导入Crypto.PublicKey.RSA

用RSA模块的import_key函数将我们的publickey读入，拿到RSA加密的公钥n和e的具体参数

from Crypto.PublicKey import RSA

f=open("public.key","r")

key=RSA.import_key(f.read())

f.close()

e=key.e

n=key.n

然后发现n可以分解，直接在线网站http://factordb.com/把n给分解了，得到公钥n的两个因子。

然后就是常规的RSA求解私钥

这边因为flag.enc是RSA的PKCS1_OAEP加密得来的。所以我们这边也是给生成一个私钥文件。

在做到这边的时候，如何导出一个私钥文件。找了一下百度上的方法。都是先generate后给参数分别赋值的。但是我发现我并不行，试了一下python3和python2下的Crypto库都得到一个报错

Exception has occurred: AttributeError

can't set attribute

也就是现在无法通过这么直接赋值了。

这种情况的话，可以去看下python调用的Crypto库里面的RSA模块的一个底层的实现。

发现有一个construct函数，传入一个rsa_components参数，是一个元组型的数据，也就是tuple类型的，分别是(n,e,d,p,q)

phi=(p-1)*(q-1)

d=gmpy2.invert(e,phi)

rsa_components=(n,e,int(d),p,q)

arsa=RSA.construct(rsa_components)

arsa.exportKey()

然后导出的私钥，对加密后的密文，使用PKCS1_OAEP模块进行解密即可。

题解exp如下

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
f=open("public.key","r")
key=RSA.import_key(f.read())
f.close()
e=key.e
n=key.n
import base64
from Crypto.Util.number import *
import gmpy2
p= 3133337
q=254783260649374192922001721363994977190818429145282283164559062116931183219713999360041
print(p*q==n)
f=open("flag.enc","r")
c_base64=f.read().strip("\n")
c_bytes=base64.b64decode(c_base64)
c=bytes_to_long(c_bytes)
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
rsa_components=(n,e,int(d),p,q)
arsa=RSA.construct(rsa_components)
rsakey = RSA.importKey(arsa.exportKey())
```

```
rsakey = PKCS1_OAEP.new(rsakey)

decrypted = rsakey.decrypt(c_bytes)

print(decrypted)
```

[AFCTF2018]BASE

题目链接(buu平台)

附件

flag_encode.txt

拿到附件是一个flag加密后的文本，其实是flag经过很多次base系列编码后的一个内容。文件有20+MB，文本编辑器发现是一行编码后的内容。

这题主要是三种编码，base64/base32/base16。base16也就是16进制。

就疯狂解码就行了。主要是考察一个脚本的编写吧，也只能这么理解。。

题解exp如下：

```
import base64

f=open("flag_encode.txt","r")

c=f.readline()

while True:

ok=0

try:

c=base64.b64decode(c).decode("ascii")

ok=1

except:

pass

try:

c=base64.b16decode(c).decode("ascii")

ok=1

except:

pass

try:

c=base64.b32decode(c).decode("ascii")

ok=1

except:
```

pass

if not ok:

print(c)

break

[WUSTCTF2020]情书

题目链接(buu平台)

附件

attachment.txt

拿到附件发现说明里面是RSA加密

并且给了密钥，密文。直接写脚本解就完事了。

接出来的内容为字母表的索引，找到对应的字母即可

题解exp如下：

en=[156,821,1616,41,140,2130,1616,793]

import string

table=string.ascii_lowercase

print(table)

for i in en:

print(table[pow(i,937,2537)],end="")

[NPUCTF2020]Classical Cipher

题目链接(buu平台)

附件

key.txt

flag.zip

flag.zip为加密的zip文件，zip密码为key.txt中

因为这边有一组明密文的对应关系，即key→pvb，古典密码学思想直接词频分析。

得到结果：the_key_is_atrash

尝试用这个去解密发现密码错误。

然后仔细分析一下这个词频分析结果。发现有atbash，这个词的话就比较熟悉，古典密码里面经典的单表替换密码。

其实压缩包密码为the_key_is_atbash。主要是因为词频分析的话atrash更符合表达。

直接atbash解密即可

解开压缩包拿到图片

猪圈密码的话是比较常见的一种编码了。可以百度一下，这边要注意一点，这个猪圈密码和CTFwiki上面给出的那个表不同。对应关系参照的是这个表：

然后还有一些动物啥的，其实是象形文字。。。怎么说呢，勉勉强强算是理解成一种编码吧。这边对应的编码表也是给大家找来了。

这边对应下两张表去解出明文即可。

[网鼎杯 2020 青龙组]you_raise_me_up

题目链接(buu平台)

附件

you_raise_me_up.py

附件就是一个加密脚本，然后输出的话也是直接注释在了脚本里面。

其实乍一看你会发现他和RSA加密很像，最后的加密过程为。

c = pow(m, bytes_to_long(flag), n)

我们知道RSA加密就是令明文为m，取公钥e和n，密文c=pow(m,e,n)

这边的一个明显区别为，可以理解为flag明文作为RSA加密里面的公钥e进行的求解。

这个在密码学里面是基于离散对数的一种加密，我们在求解明文的时候，也就相当于是求解基于同余运算和原根的一种对数运算。

求解这种问题的话我们用python的sympy模块的discrete_log函数进行求解就可以了。discrete_log(n,c,m)

题解exp如下：

```
m = 391190709124527428959489662565274039318305952172936859403855079581402770986890308469084
```

```
c = 666585139420321424585678945072365863252081679162179677590976689523300023402364287878602
```

```
n = 2 ** 512

import sympy

from Crypto.Util.number import *

flag=sympy.discrete_log(n,c,m)
```

print(long_to_bytes(flag))

[ACTF新生赛2020]crypto-aes

题目链接(buu平台)

附件

aes.py

output

分析附件，发现aes.py是一个简单的aes cbc模式的加密脚本。

def main():

key=os.urandom(2)*16

iv=os.urandom(16)

print(bytes_to_long(key)^bytes_to_long(iv))

aes=AES.new(key,AES.MODE_CBC,iv)

enc_flag = aes.encrypt(FLAG)

print(enc_flag)

if __name__=="__main__":

main()

很显然的是，key和iv为两个随机比特流。但是特点很明显。

其中key是长度为两个字节的比特流重复了16次。iv就是一个长度为16字节的比特流

故这边，key的长度为32字节，iv的长度为16字节

然后给了我们key和iv异或的输出结果

这边易得到一个结论，key的低16字节与iv的16字节异或，而key的高16字节就保留了，可以看作高16字节的每一位都是与0异或，而与0异或结果就是它本身。

这边我们将output文件中得到的key与iv异或结果的10进制值，转换成16进制值。我们知道16进制的每两位表示的是一个字节。这边可以看到输出结果的高位都是c981的一个重复。即可得key中重复的两字节的16进制分别是c9和81，如此我们也就得到了key，再异或得到iv进行aes基础解密即可。

题解exp如下：

import os

from Crypto.Util.number import *

from Crypto.Cipher import AES

xor_re=91144196586662942563895769614300232343026691029427747065707381728622849079757

key=b'\xc9\x81'*16

```
print(key)

key_long=bytes_to_long(key)

iv=(long_to_bytes(xor_re^key_long))

print(iv)

c=b'\x8c-\xcd\xde\xa7\xe9\x7f.b\x8aKs\xf1\xba\xc75\xc4d\x13\x07\xac\xa4&\xd6\x91\xfe\xf3\x14\x10|\xf8p'

aes=AES.new(key,AES.MODE_CBC,iv)

flag=aes.decrypt(c)

print(flag)
```

[INSHack2017]rsa16m

题目链接(buu平台)

附件

rsa_16m

附件rsa_16m是一个7+MB的文件，里面主要的一个数据为RSA加密体系中的n,e,c。

可以发现的是n极大，e虽然也挺大的，为0x10001。但是这边明文flag的e次方还是小于n的。故这一题直接用rsa里面的小明文攻击即可。

直接对密文c开e次方根

题解exp如下:

```
f=open("rsa_16m","r")

f.readline()

c=int(f.readline().strip("\n").split(" = ")[1],16)

e=0x10001

import gmpy2

from Crypto.Util.number import *

flag=long_to_bytes(gmpy2.iroot(c,e)[0])

print(flag)
```

[XNUCA2018]Warmup

题目链接(buu平台)

附件

Buggy_Server.py

sniffed.pcapng

简单分析一下Buggy_Server.py脚本，是一个模拟发送消息的脚本，使用的是rsa加密，通过传入user将flag消息进行rsa加密发送给user，并抓取所有消息的一个流量。

直接对流量包进行一个分析，追踪一下TCP流，可以发现

题目给出的流量包的内容，即发送给user的rsa加密数据。

这边发现，发送给Alice和Carol这两个用户，使用的公钥n是一样的。那就是一个rsa里面经典的共模攻击。

题解exp如下：

```
import sys
import binascii
sys.setrecursionlimit(1000000)
def egcd(a, b):
if a == 0:
return (b, 0, 1)
else:
g, y, x = egcd(b % a, a)
return (g, x - (b // a) * y, y)
def modinv(a, m):
g, x, y = egcd(a, m)
if g != 1:
raise Exception('modular inverse does not exist')
else:
return x % m
c1=22917655888781915689291442748409371798632133107968171254672911561608350738343707972888
n=25118186052801903419891574512806521370646053661385577314262283167479853375867074736882
e1=7669
c2=20494665879116666159961016125949070097530413770391893858215547229071116025581822729797
e2=6947
s = egcd(e1, e2)
s1 = s[1]
s2 = s[2]
if s1<0:
s1 = - s1
```

c1 = modinv(c1, n)

elif s2<0:

s2 = - s2

c2 = modinv(c2, n)

m=(pow(c1,s1,n)*pow(c2,s2,n)) % n

print(m)

print (binascii.unhexlify(hex(m)[2:].strip("L")))

[b01lers2020]safety_in_numbers

题目链接(buu平台)

附件

enc.py

flag.enc

pubkey.pem

这边的话enc.py里面要注意一个点，他从int转成bytes和bytes转int都是用的小端模式也就是byteorder = 'little',这个和我们正常的一个转换刚好相反的。

然后可以发现是pubkey.pem极大，有1.6MB，这边如果是使用RSA.importKey是无法取出他的公钥n，e的。因为根本无法导入。但是我们其实知道公钥文件的格式，

e其实就在base64编码的公钥文件的尾部

import base64

key_end_base64="l8UvtaFCpNsgheCRz1j+HD9cHH05ozrbHMe/rtEUQa6fmQcAJbDNBXZV+yabO1aSKwVm5Z

key=base64.b64decode(key_end_base64)

print(hex(bytes_to_long(key)))

得到结果如下

明显最后的16进制值为0x10001，这也是rsa中最常用的一个公钥e。

因为n极大，这题也是直接小明文攻击求解。

题解exp如下：

from Crypto.Util.number import *

import base64

import gmpy2

key_end_base64="l8UvtaFCpNsgheCRz1j+HD9cHH05ozrbHMe/rtEUQa6fmQcAJbDNBXZV+yabO1aSKwVm5Z

```
key=base64.b64decode(key_end_base64)

print(hex(bytes_to_long(key)))

f=open("flag.enc","rb")

c_bytes=f.read()

c=int.from_bytes(c_bytes,byteorder="little")

m=gmpy2.iroot(c,0x10001)[0]

flag=long_to_bytes(m)[::-1]

print(flag)
```

[AFCTF2018]MyOwnCBC

题目链接(buu平台)

附件

MyOwnCBC.py

flag_cipher

```
def MyOwnCBC(key, plain):

if len(key)!=32:

return "error!"

cipher_txt = b""

cipher_arr = []

cipher = AES.new(key, AES.MODE_ECB, "")

plain = [plain[i:i+32] for i in range(0, len(plain), 32)]

print (plain)

cipher_arr.append(cipher.encrypt(plain[0]))

cipher_txt += cipher_arr[0]

for i in range(1, len(plain)):

cipher = AES.new(cipher_arr[i-1], AES.MODE_ECB, "")

cipher_arr.append(cipher.encrypt(plain[i]))

cipher_txt += cipher_arr[i]

return cipher_txt
```

简单的对MyOwnCBC.py脚本进行分析，可以发现，其实就是ECB模式下的AES，然后将每个分组单独进行了AES ECB加密，可以看到的是每一组AES加密的key为上一组加密结果的密文。

出题人设计这个题目的意图应该是想带大家了解一下AES CBC模式，用ECB模式进行一个大概的模拟(当然不是这个原理)。主要是想表达CBC模式下会把上一轮的加密影响扩散到下一轮的意思吧。

那解密也很简单，因为key其实就是每组加密后的密文嘛，直接AES ECB模式下解密即可。

题解exp如下：

```
from Crypto.Cipher import AES
from Crypto.Random import random
from Crypto.Util.number import long_to_bytes
f=open("flag_cipher","rb")
st=f.read()
print(len(st))
def MyOwnCBC(key, plain):
cipher_txt = b""
cipher = AES.new(key, AES.MODE_ECB)
cipher_txt=cipher.decrypt(plain)
return cipher_txt
#for i in range(len(st)//32):
flag=""
for i in range(1,10):
plain=MyOwnCBC(st[(52-i-1)*32:(52-i)*32],st[(52-i)*32:(53-i)*32])
flag=plain.decode()+flag
print(flag)
```

[RoarCTF2019]RSA

题目链接(buu平台)

附件

attachment

题面如下：

```
A=(((y%x)**5)%(x%y))**2019+y**316+(y+1)/x
p=next_prime(z*x*y)
q=next_prime(z)
A =
n =
c =
```

这一题我们发现n直接可以分解。

这一点我也很费解。。因为RoarCTF当时也有参与出题和测题。明明不应该出现这个问题。这个factordb的网站有收录就很离谱。

这边介绍一下预期解。

我们首先看到

A=(((y%x)**5)%(x%y))**2019+y**316+(y+1)/x

这边可以发现，其实这个方程里面最大的一项是(((y%x)5)%(x%y))2019

这边之前底数大于等于2，那么这一项直接就大于A了。所以这一项必然是0或者1

其次就是y316了。发现当y为84的时候y316就已经大于A了。所以得到y肯定是小于84的

然后最后一项的结果肯定也很小。所以这边基本就确定了y为83了。然后爆破一下x，得到x为2

到这里我们就得到了x*y的一个值。然后我们知道p和q共有一个大因子z，虽然最后是取了一个next_prime，但是我们可以先得到一个大概的值

得到一个z的大概值

xy=x*y

zz_near=n//xy

z_near=gmpy2.iroot(zz_near,2)[0]

然后写个二分法去求解一下pq的值

low=0

hei=10000000

while low

mid=(low+hei)//2

p=gmpy2.next_prime((z_near-mid)*x*y)

q=gmpy2.next_prime((z_near-mid))

if p*q-n>0:

low=mid+1

elif p*q-n<0:

hei=mid-1

else:

break

然后基础的RSA解密即可

题解exp如下：

import gmpy2

```
from Crypto.Util.number import *

y=0

A=26833491826787145242474695127934760098610147810049249054841274803081613777681928680615

n =
11793080604350737432598229182302728514880723911798736960958351535388981485608809967145 43

c =
41971850275428383625653350824107291609587853887037624239544762751558838294718672159979 92

'''

for i in range(1000):

if i**316>A:

print(i)

break

'''

y=83

x=0

'''

for x in range(1,1000):

if (((y%x)**5)%(x%y))**2019+y**316+(y+1)//x==A:

print(x)

break

'''

x=2

xy=x*y

zz_near=n//xy

z_near=gmpy2.iroot(zz_near,2)[0]

low=0

hei=10000000

while low

mid=(low+hei)//2

p=gmpy2.next_prime((z_near-mid)*x*y)

q=gmpy2.next_prime((z_near-mid))
```

```
if p*q-n>0:
low=mid+1
elif p*q-n<0:
hei=mid-1
else:
break
print(p*q==n)
phi=(p-1)*(q-1)
e=0x10001
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
flag=long_to_bytes(m)
print(flag)
```

[AFCTF2018]One Secret, Two encryption

题目链接(buu平台)

附件

public1.pub

public2.pub

flag_encry1

flag_encry2

给了一份题面

一份秘密发送给两个人不太好吧，那我各自加密一次好啦~~~

素数生成好慢呀

偷个懒也……不会有问题的吧？

这边首先把两份公钥文件的n,e取出来看了一下

```
from Crypto.PublicKey import RSA
import gmpy2
from Crypto.Util.number import *
f=open("public1.pub","r")
rsa1=RSA.import_key(f.read())
n1=rsa1.n
```

```
e1=rsa1.e

f.close()

f=open("public2.pub","r")

rsa2=RSA.import_key(f.read())

n2=rsa2.n

e2=rsa2.e

f.close()

print(n1,e1)

print(n2,e2)
```

测试发现n1 n2有公约数。这边也是印证了题面给的懒得生成素数，所以共用了其中一个素数。这边直接gcd求解公约数即可。

然后分解出p q进行一个基础的RSA解密

题解exp如下：

```
from Crypto.PublicKey import RSA

import gmpy2

from Crypto.Util.number import *

f=open("public1.pub","r")

rsa1=RSA.import_key(f.read())

n1=rsa1.n

e1=rsa1.e

f.close()

f=open("public2.pub","r")

rsa2=RSA.import_key(f.read())

n2=rsa2.n

e2=rsa2.e

f.close()

#print(n1,e1)

#print(n2,e2)

p=gmpy2.gcd(n1,n2)

q=n2//p

assert(p*q==n2)
```

```
phi=(p-1)*(q-1)

d=gmpy2.invert(e2,phi)

c_bytes=open("flag_encry2","rb").read()

c=bytes_to_long(c_bytes)

m=pow(c,d,n2)

flag=long_to_bytes(m)

print(flag)
```

[INSHack2019]Yet Another RSA Challenge – Part 1

题目链接(buu平台)

附件

yarsac.py

output.txt

题面很清晰，就是一个RSA加密，然后给了公钥n的其中有一个因子p，但是p是16进制表示的，最后输出的时候把9F替换成了FC。那么我们在给的output.txt文件中得到p的时候，其中的FC的位置就有可能为9F，亦有可能为FC。这边写个脚本爆破一下求解即可。

题解exp如下：

```
import gmpy2

from Crypto.Util.number import *

n=719579745653303119025873098043848913976880838286635817351790189702008424828505522253333

cipher=596380963583874022971492302071822444225514552231574984926542429117396590795270181O

e=65537

guess=["9F","FC"]

p_=0

for a in guess:

for b in guess:

for c in guess:

for d in guess:

p="DCC5A0BD3A1"+a+"0BEB0DA1C2E8CF6B474481B7C12849B76E03C4C946724DB577D2825D6AA193D

p=int(p,16)

q=n//p

if (p*q==n):
```

```
p_=p
p=p_
q=n//p
print(p*q==n)
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(cipher,d,n)
flag=long_to_bytes(m)
print(flag)
```

[GUET-CTF2019]NO SOS

题目链接(buu平台)

附件：

attachment.txt

首先发现一个很类似摩斯电码的东西。但是题目也写了no sos，所以应该不是莫斯电码。并且也发现没有间隔。与之类似的有可能是2进制的表示或者培根密码。这边将.转为A将—转为B后发现位数刚好为5的倍数。

培根密码解密得到flag

[UTCTF2020]basic-crypto

题目链接(buu平台)

附件：

attachment.txt

这题附件里面表示的就很明显，每一组都是8位二进制。直接转ascii字符即可。

题解exp如下：

```
f=open("attachment.txt","r")
arr=f.read().split(" ")
re=""
for i in arr:
re+=chr(int(i,2))
print(re)
```

[ACTF新生赛2020]crypto-rsa3

题目链接(buu平台)

附件：

rsa3.py

output.txt

可以发现就是最基础的RSA加密，其中p q为相邻的素数。直接yafu分解即可。

然后基础RSA解密即可

题解exp如下：

n=17760650483649924697095903022687160888596932177821105108052463408451697333144164499389

c=14573903785113823547710005409453611689847750526930736416823750714074908512897030709057

e=65537

p =
13326909050357447643526585836833969378078147057723054701432842192988717649385731430095 05

q =
13326909050357447643526585836833969378078147057723054701432842192988717649385731430095 05

```
from Crypto.Util.number import *
import gmpy2
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
flag=long_to_bytes(m)
print(flag)
```

[MRCTF2020]babyRSA

题目链接(buu平台)

附件：

baby_RSA.py

题目脚本比较长，一步一步分析。

```
if __name__ == "__main__":
_E = base
_P = gen_p()
_Q = gen_q()
```

```
assert (gcd(_E, (_P - 1) * (_Q - 1)) == 1)

_M = bytes_to_long(flag)

_C = pow(_M, _E, _P * _Q)

print("Ciphertext = ", _C)
```

首先，在主函数里面可以看到。其实这就是一个RSA加密。无非是_P和_Q都是带入了两个函数里面求解的。

```
def gen_q():

sub_Q = getPrime(1024)

Q_1 = getPrime(1024)

Q_2 = getPrime(1024)

Q = sub_Q ** Q_2 % Q_1

print("Q_1: ", Q_1)

print("Q_2: ", Q_2)

print("sub_Q: ", sub_Q)

return sympy.nextprime(Q)
```

_Q这个参数的话可以看到我们已知Q_1、Q_2、sub_Q就可以直接求解。

```
import sympy

import gmpy2

from Crypto.Util.number import *

Q_1=10376643984946558808462504949579385763455651706456348843314822452463810597116105176

Q_2=15101073427691693979059146127898148644254803503235079730649610513635872358695312348

sub_Q=16899252979359331575789599510143024199495363833091931480013053680980182497111203

Q=pow(sub_Q,Q_2, Q_1)

q=sympy.nextprime(Q)

print(q)
```

然后看到gen_p()这个函数

```
def gen_p():

P = [0 for i in range(17)]

P[0] = getPrime(128)

for i in range(1, 17):

P[i] = sympy.nextprime(P[i-1])
```

```
print("P_p :", P[9])

n = 1

for i in range(17):

n *= P[i]

p = getPrime(1024)

factor = pow(p, base, n)

print("P_factor :", factor)

return sympy.nextprime(p)
```

这边我们要求解p，其实就相当于求解一个RSA。n是由17个连续的质数作为因子的。然后我们得到了其中的第十个也就是下标为9的因子。那么我们通过sympy.prevprime和sympy.nextprime去求解得到所有的质数进行RSA基础解密得到p，然后取p的下一个质数作为_P即可。

题解exp如下：

```
import sympy

import gmpy2

from Crypto.Util.number import *

Q_1=10376643984946558808462504949579385763455651706456348843314822452463810597116105176

Q_2=15101073427691693979059146127898148644254803503235079730649610513635872358695312348

sub_Q=16899252979359331575789599510143024199495363833091931480013053680980182497111203

Q=pow(sub_Q,Q_2, Q_1)

q=sympy.nextprime(Q)

print(q)

P_p=20602792684730861271967757255499114342 1

P=[]

for i in range(9):

P_p=sympy.prevprime(P_p)

P.append(P_p)

P=P[::-1]

P_p=20602792684730861271967757255499114342 1

P.append(P_p)

for i in range(9):

P_p=sympy.nextprime(P_p)
```

```
P.append(P_p)

n_p=1

for i in range(17):

n_p *= P[i]

phi_p=1

for i in range(17):

phi_p *= (P[i]-1)

P_factor=213671742765908980787116579976289600595864704574134469173111790965233629909513884

base = 65537

d_p=gmpy2.invert(base,phi_p)

p=sympy.nextprime(pow(P_factor,d_p,n_p))

Ciphertext =
170918724051636714146086218774945104764409488579176167357467433084084279218979504996839 4

phi=(p-1)*(q-1)

d=gmpy2.invert(base,phi)

m=pow(Ciphertext,d,p*q)

flag=long_to_bytes(m)

print(flag)
```

[WUSTCTF2020]dp_leaking_1s_very_d@angerous

题目链接(buu平台)

https://buuoj.cn/challenges#[WUSTCTF2020]dp_leaking_1s_very_d@angerous

附件：

attachment

题意很明显，是dp泄露。

dp=d%(p-1)

这种参数是为了让解密的时候更快速产生的

题解exp如下：

```
import gmpy2

import rsa

import binascii

p=0
```

```
e=65537

n =
156808343598578774957375696815188980682166740609302831099696492068246337198792510898818 4

c =
108542078809057774666748066235473292495343753790443966020636060807418393737258696352569 3

dp =
734763139918837027274765680404546851353356952885439663987181004382601658386317353877499 1

temp=dp*e

for i in range(1,e) :

if (temp-1)%i==0:

x=(temp-1)//i+1

y=n%x

if y==0:

p=x

break

q=n//p

d=gmpy2.invert(e,(p-1)*(q-1))

key=rsa.PrivateKey(n,e,d,p,q)

m=pow(c,d,n)

print(binascii.unhexlify(hex(m)[2:]))
```