CTF中RSA常用攻击方法真题及脚本汇总

原创

前言

具体的思路在上一篇博文中，这一篇主要整理题型和相应脚本。思路上一篇博文地址
代码不精，部分参考其他道友的代码。慢慢更新。

可能会遇见的问题

- 如果缺少某些模块，例如`gmpy2`，请自行百度或者goolge下载办法。linux可以参考这篇博文
  。
- 代码中的`gmpy`模块的`gmpy.root`与`gmpy2.iroot`基本可以互换。
- 部分代码只能在`python2.x`环境中运行，因为某些模块没有相应的python3版本。
- 整理的还没有完，喜欢的可以收藏

常规破解模数n

这一种一般都是属于签到题目，一般都是通过`yafu`或者在线网站`http://factordb.com/index.php`分解n，就可以按照正常的解密思路进行解密了。
这种类型的题目主要分为两种：

- 以文本形式给出 RSA的各个参数

```
#!/usr/bin/env python3
# coding:utf-8
#power by jedi

import gmpy2
import binascii

n = 966808932627497190635859236054960349099463975227350564265384373280336699853387254070662881265937565163000758
606154308757944030571837175048514574473061401566330836334647176655282619268592560172726526643074499534129878217 4
090460455336568970501174384963572315759991855276750710028039518006352200290159320074651178187399489037502008308 5
611566869100770683695224484271941945294625927525177329833816238993051883827270490888701647400705139719458839603 9
111216708866214614779627566959335170676055025850932631053641576566165694121420546081043285806783239296799795655 1
911219663775901757806189449105328169881430567570540526799685389014608935712049043949757140810554552405238956533 0
531551774572933411454975669533417114287608047710507040954477798160215276215461073854016379616429522281024330905 1
503090866674634440359226192530724635477051576515179864461174911975667162597286769079380660782647952944808596310 4
769739391561874720769529357282490611374818875891039735910828729886419582702851696508037923955563633040562900778 0
145398082209758357430968293569726020486275692386555639768669685423956454140718570994010780653677316026376446344 3
859425726953142964148216209968437587044617613518058779287167853349364533716458676066734216877566181514607693882 3
75533
# p 和 q通过在线网站http://factordb.com/index.php分解
p = gmpy2.mpz(310935513029228809998830208036655366162721470228774287453148308675193510132489142448801010943658159
980501154153084396100667001391643762749806500051502679498536716532334917842894939889468693960937309663256592497
965458780801192062835123429808544757340971089756707788360038227894054989413747980167536893779923551227744017809 3
018559845824089436224619424862391138228416967759586450147530819464414060227296169923028299302050766893998020507 9
239221924230430230318076991507619960330144745307022538024878444458717587446601559546292026245318907293584609320 1
153746322352707956339337553509285375982422142166744964096259287974504 73)
q = gmpy2.mpz(310935513029228809998830208036655366162721470228774287453148308675193510132489142448801010943658159
980501154153084396100667001391643762749806500051502679498536716532334917842894939889468693960937309663256592497
965458780801192062835123429808544757340971089756707788360038227894054989413747980167536893779923551227744017809 3
018559845824089436224619424862391138228416967759586450147530819464414060227296169923028299302050766893998020507 9
239221924230430230318076991507619960330144745307022538024878444458717587446601559546292026245318907293584609320 1
153746322352707956339337553509285375982422142166744964096259289977 7221)
e = gmpy2.mpz(65537)
phi_n = (p-1)*(q-1)
d = gmpy2.invert(e, phi_n)
c = gmpy2.mpz(168502910088858295634315070244377409556567637139736308082186369003227771936407321783557795624279 16
216230520043644690397638594867789766546629085276987756216748714238530802734163981640105508182049700201890889620 2
860342391029082581621987305533097386652183849657065952062433988387640990383623264405525144003500286531262674315 9
005370018450432253631483597667710338996801110761816727970774105847475095819320455408017777385488727475978999653 6
695082750552943248377982115815292889994783719639155566616548644187818328800875356110899571596192047292784487756 9
855940505148843530998878113722830427807926679324241141182238903567682042410145345551889442158895157875798990903 7
151057826820838864616613070635834476961688286871269561479558864933838055135576041790290509816787550549456078663 5
319579365410840393924272386165191915236992390400296687399481182639108031814626041697849937718254068440979035725 7
490816203138499369634490897553227763563553981246891677613446390134477832143175248992161641698011195968792105201 8
479760823227866233902424702267406858222181402631820242262286921593805576615916330720919450773341919878602624483 8
512359945964722856213736917806907280449804946313623385633781738597799014557104223179533299552398817489543281987 2
832170029690848)

m = pow(c, d, n)
print("十进制:\n%s"%m)
m_hex = hex(m)[2:]
print("十六进制:\n%s"%(m_hex,))
#print("ascII:\n%s"%((binascii.b2a_hex(hex(m)[2:])).decode('hex'),))
print("ascii:\n%s"%(binascii.a2b_hex(m_hex).decode("utf8"),))
```

- 以文件形式给出参数:

```python
#/usr/bin/env python
#coding:utf-8
#power by jedi

from Crypto.PublicKey import RSA
import gmpy2
import rsa

#1.从公钥文件中分解n和e
public_key = RSA.importKey(open("./tmp/pubkey.pem").read())
n = public_key.n
e = public_key.e
#print("n=\n%s\ne=\n%s"%(n,e))

#2.在线分解n得到p和q
p = 275127860351348928173285174381581152299
q = 319576316814478949870590164193048041239

#3.计算出d
d = int(gmpy2.invert(e, (p-1)*(q-1)))
#print(d)

#通过已知条件，生成私钥，并解密密文
private_key = rsa.PrivateKey(n, e, d, p, q)#生成私钥
with open("./tmp/flag.enc") as f:
    flag = rsa.decrypt(f.read(), private_key)
    print(flag)
    #print(rsa.decrypt(f.read(), private_key).decode())
```

题目有：2017第二届广东省强网杯线上赛-50-RSA，ISC2016训练赛——phrackCTF-200-mediumRSA等等。

加密指数e=1

一般思路和形式是：

e=1

加密过程变为

> c = pow(m, e, N) = pow(m, 1, N) = m mod N
> 下面公式可以得到m
> m = c+N * k (k=0,1,2,3,4…)

```python
#/usr/bin/env python3
#coding:utf-8
import binascii
import gmpy2
N_hex=0x180be86dc898a3c3a710e52b31de460f8f350610bf63e6b2203c08fddad44601d96eb454a34dab7684589bc32b19eb27cffff8c0
7179e349ddb62898ae896f8c681796052ae1598bd41f35491175c9b60ae2260d0d4ebac05b4b6f2677a7609c2fe6194fe7b63841cec632e3
a2f55d0cb09df08eacea34394ad473577dea5131552b0b30efac31c59087bfe603d2b13bed7d14967bfd489157aa01b14b4e1bd08d9b92ec
0c319aeb8fedd535c56770aac95247d116d59cae2f99c3b51f43093fd39c10f93830c1ece75ee37e5fcdc5b174052eccadcadeda2f1b3a4a
87184041d5c1a6a0b2eeaa3c3a1227bc27e130e67ac397b375ffe7c873e9b1c649812edcd
e_hex=0x1
c_hex=0x4963654354467b66616c6c735f61706172745f736f5f656173696c795f616e645f7265617373656d626c65645f736f5f63727564
656c797d

c_hex = gmpy2.mpz(c_hex)
N_hex = gmpy2.mpz(N_hex)

i = 0
while i<10:
    m_hex = hex(c_hex + gmpy2.mpz(hex(i))*N_hex)
    print(m_hex[2:])
    try:
        print(binascii.a2b_hex(m_hex[2:]).decode("utf8"))
    except binascii.Error as e:
        print("位数非偶数，跳过...")
    i += 1
```

ICECTF-RSA？

同种类型的还有这题

> n��0x1c2a3d4b5f6e7a718d9ff8b9a828fd9s9a98sa9adafae4b07549593d9f1d06adafaea9aa9f8e0a8f9d9b8
> e:0x01
> c:0x666c61677b7273615f31732d737469316c5f653473795f6e6f777d

加密指数e=2_rabin

适用于e=2

```python
#!/usr/bin/python
# coding=utf-8
# 适合e=2
import gmpy
import string
from Crypto.PublicKey import RSA

# 读取公钥参数
with open('./tmp/pubkey.pem', 'r') as f:
    key = RSA.importKey(f)
    N = key.n
    e = key.e

p = 275127860351348928173285174381581152299
q = 319576316814478949870590164193048041239
with open('./tmp/flag.enc', 'r') as f:
    cipher = f.read().encode('hex')
    cipher = string.atoi(cipher, base=16)
    # print cipher

# 计算yp和yq
yp = gmpy.invert(p,q)
yq = gmpy.invert(q,p)

# 计算mp和mq
mp = pow(cipher, (p + 1) / 4, p)
mq = pow(cipher, (q + 1) / 4, q)

# 计算a,b,c,d
a = (yp * p * mq + yq * q * mp) % N
b = N - int(a)
c = (yp * p * mq - yq * q * mp) % N
d = N - int(c)

for i in (a,b,c,d):
    s = '%x' % i
    if len(s) % 2 != 0:
        s = '0' + s
    print s.decode('hex')
```

ISC2016训练赛——phrackCTF–400-varyhardRSA

同模攻击

题目的特点是N是一样的
当n不变的情况下，知道n,e1,e2,c1,c2 可以在不知道d1,d2的情况下，解出m
e1，e2互质

```python
#! /usr/bin/env python2
# -*- coding: utf-8 -*-

from libnum import n2s, s2n
from gmpy2 import invert

# 扩展欧几里得算法
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 main():
  n = 116547141139745534253172934123407786743246513874292261984447028928003798881819567221547298751255790928878194794155722543477883428672342894945552668904410126460402501558930911637857436926624838677630868157884406020858164140754510239986466552869866296144106255873879659676368694043769795604582888907403261286211
  c1 = 78552378607874335972488545767374401332953345586323262531477516680347117293352843468592985447836452620945707838830990843415342047337735534418287912723395148814463617627398248738969202758950481027762126608368555442533803610260859075919831387641824493902538796161102236794716963153162784732179636344267189394853
  c2 = 98790462909782651815146615208104450165337326951856608832305081731255876886710141821823912122797166057063387122774480296375186739026132806230834774921466445172852604926204802577270611302881214045975455878277660638731607530487289267225666045742782663867519468766276566912954519691795540730313772338991769270201
  e1 = 1804229351
  e2 = 17249876309
  s = egcd(e1, e2)
  s1 = s[1]
  s2 = s[2]
  # 求模反元素
  if s1 < 0:
    s1 = - s1
    c1 = invert(c1, n)
  elif s2 < 0:
    s2 = - s2
    c2 = invert(c2, n)

  m = pow(c1, s1, n) * pow(c2, s2, n) % n
  print(n2s(m))  # 二进制转string


if __name__ == '__main__':
  main()
```

```
#!/usr/bin/python
# coding=utf-8
# 文件类型
import string
import gmpy


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 main():
    with open('flag.enc1', 'r') as f1:
        c1 = f1.read().encode('hex')
        c1 = string.atoi(c1, base=16)

    with open('flag.enc2', 'r') as f2:
        c2 = f2.read().encode('hex')
        c2 = string.atoi(c2, base=16)

    n = 0x00b0bee5e3e9e5a7e8d00b493355c618fc8c7d7d03b82e409951c182f398dee3104580e7ba70d383ae5311475656e8a964d380
cb157f48c951adfa65db0b122ca40e42fa709189b719a4f0d746e2f6069baf11cebd650f14b93c977352fd13b1eea6d6e1da775502abff89
d3a8b3615fd0db49b88a976bc20568489284e181f6f11e270891c8ef80017bad238e363039a458470f1749101bc29949d3a4f4038d463938
851579c7525a69984f15b5667f34209b70eb261136947fa123e549dfff00601883afd936fe411e006e4e93d1a00b0fea541bbfc8c5186cb6
220503a94b2413110d640c77ea54ba3220fc8f4cc6ce77151e29b3e06578c478bd1bebe04589ef9a197f6f806db8b3ecd826cad24f5324cc
dec6e8fead2c2150068602c8dcdc59402ccac9424b790048ccdd9327068095efa010b7f196c74ba8c37b128f9e1411751633f78b7b9e56f7
1f77a1b4daad3fc54b5e7ef935d9a72fb176759765522b4bbc02e314d5c06b64d5054b7b096c601236e6ccf45b5e611c805d335dbab0c35d
226cc208d8ce4736ba39a0354426fae006c7fe52d5267dcfb9c3884f51fddfdf4a9794bcfe0e1557113749e6c8ef421dba263aff68739ce0
0ed80fd0022ef92d3488f76deb62bdef7bea6026f22a1d25aa2a92d124414a8021fe0c174b9803e6bb5fad75e186a946a17280770f1243f4
387446ccceb2222a965cc30b3929L

    e1 = 17
    e2 = 65537
    s = egcd(e1, e2)
    s1 = s[1]
    s2 = s[2]

    # 求模反元素
    if s1 < 0:
        s1 = -s1
        c1 = gmpy.invert(c1, n)
    elif s2 < 0:
        s2 = -s2
        c2 = gmpy.invert(c2, n)

    m = pow(c1, s1, n) * pow(c2, s2, n) % n
    print '{:x}'.format(int(m)).decode('hex')
if __name__ == '__main__':
    main()
```

实验吧题目

低加密指数e攻击

尝用于加密指数e=3的情况

```python
#!/usr/bin/env python
#coding:utf-8

import gmpy2
from Crypto.PublicKey import RSA

public_key = "./tmp/pubkey.pem"
cipher_file = "./tmp/flag.enc"

#读入公钥
with open(public_key, "r") as f:
    key = RSA.importKey(f)
    n = key.n
    e = key.e

#读入密文
with open(cipher_file, "r") as f:
    cipher = f.read().encode("hex")
    cipher = int(cipher, 16)
    #print(cipher)

#破解密文
def get_flag():
    i = 0
    while True:
        if(gmpy2.iroot(cipher+i*n, 3)[1] == True):
            flag_bin = int(gmpy2.iroot(cipher+x*n, 3)[0])
            flag = hex(flag_bin)[2:-1].decode("hex")
            print(flag)
            break
        i += 1

def get_flag_for():
    for x in xrange(118600000, 118720000):
        if(gmpy2.iroot(cipher+x*n, 3)[1] == 1):
            flag_bin = int(gmpy2.iroot(cipher+x*n, 3)[0])
            flag = hex(flag_bin)[2:-1].decode("hex")
            print(flag)
            break

if __name__ == "__main__":
    get_flag_for()
    #get_flag()
```

ISC2016训练赛——phrackCTF–500-extremelyhardRSA

私钥恢复和最优非对称加密填充

```python
#!/usr/bin/env python
#coding:utf-8
# 恢复公钥
from Crypto.PublicKey import RSA

public_key_file = "./tmp/pubkey.pem"
with open(public_key_file, "r") as f:
    public_key = RSA.importKey(f)
    n = public_key.n
    e = public_key.e
print("n:\n%s\ne:\n%s"%(n,e))
```

```
print("n=\n%s\ne=\n%s"%(n,e))

# wqm @ wqm-top in ~/下载/i春秋/密码/RSA脚本整理/5私钥恢复和最优非对称加密填充/ISC2016训练赛—phrackCTF--600--godlikeR
SA [22:24:30]
$ cat get_private_key.py
#!/usr/bin/python
#-*- coding:utf-8 -*-
#私钥修复
import re
import pickle
from itertools import product
from libnum import invmod, gcd


def solve_linear(a, b, mod):
    if a & 1 == 0 or b & 1 == 0:
        return None
    return (b * invmod(a, mod)) & (mod - 1)  # hack for mod = power of 2


def to_n(s):
    s = re.sub(r"[^0-9a-f]", "", s)
    return int(s, 16)


def msk(s):
    cleaned = "".join(map(lambda x: x[-2:], s.split(":")))
    return msk_ranges(cleaned), msk_mask(cleaned), msk_val(cleaned)


def msk_ranges(s):
    return [range(16) if c == " " else [int(c, 16)] for c in s]


def msk_mask(s):
    return int("".join("0" if c == " " else "f" for c in s), 16)


def msk_val(s):
    return int("".join("0" if c == " " else c for c in s), 16)


E = 65537

N = to_n("""00:c0:97:78:53:45:64:84:7d:8c:c4:b4:20:e9:33:
    58:67:ec:78:3e:6c:f5:f0:5c:a0:3e:ee:dc:25:63:
    d0:eb:2a:9e:ba:8f:19:52:a2:67:0b:e7:6e:b2:34:
    b8:6d:50:76:e0:6a:d1:03:cf:77:33:d8:b1:e9:d7:
    3b:e5:eb:1c:65:0c:25:96:fd:96:20:b9:7a:de:1d:
    bf:fd:f2:b6:bf:81:3e:3e:47:44:43:98:bf:65:2f:
    67:7e:27:75:f9:56:47:ba:c4:f0:4e:67:2b:da:e0:
    1a:77:14:40:29:c1:a8:67:5a:8f:f5:2e:be:8e:82:
    31:3d:43:26:d4:97:86:29:15:14:a9:69:36:2c:76:
    ed:b5:90:eb:ec:6f:ce:d5:ca:24:1c:aa:f6:63:f8:
    06:a2:62:cb:26:74:d3:5b:82:4b:b6:d5:e0:49:32:
    7b:62:f8:05:c4:f7:0e:86:59:9b:f3:17:25:02:aa:
    3c:97:78:84:7b:16:fd:1a:f5:67:cf:03:17:97:d0:
    c6:69:85:f0:8d:fa:ce:ee:68:24:63:06:24:e1:e4:
    4c:f8:e9:ad:25:c7:e0:c0:15:bb:b4:67:48:90:03:
    9b:20:7f:0c:17:eb:9d:13:44:ab:ab:08:a5:c3:dc:
```

```
    c1:98:88:c5:ce:4f:5a:87:9b:0b:bf:bd:d7:0e:a9:
    09:59:81:fa:88:4f:59:60:6b:84:84:ad:d9:c7:25:
    8c:e8:c0:e8:f7:26:9e:37:95:7c:e1:48:29:0f:51:
    e7:bd:98:2f:f6:cc:80:e7:f0:32:0b:89:51:92:4e:
    c2:6d:50:53:2b:3b:77:72:d1:bd:1a:1f:92:d7:12:
    79:61:61:c5:a4:7e:b3:85:eb:f0:7c:6d:46:03:c5:
    e6:d5:81:2c:ba:7e:ea:8d:51:7d:63:55:34:2a:b6:
    d4:dc:31:5a:f1:99:e3:dc:8c:83:0b:a2:2a:d5:3c:
    41:48:41:54:1a:a9:e8:b6:70:bf:d3:fe:ed:19:17:
    14:94:13:b3:17:e3:8b:8e:6f:53:ed:e2:44:e8:4a:
    32:d6:5c:0d:a8:80:f5:fc:02:e9:46:55:d5:a4:d3:
    e7:c6:30:77:f9:73:e9:44:52:d8:13:9d:5d:bf:9e:
    fa:3a:b5:96:79:82:5b:cd:19:5c:06:a9:00:96:fd:
    4c:a4:73:88:1a:ec:3c:11:de:b9:3d:e0:50:00:1e:
    ac:21:97:a1:96:7d:6b:15:f9:6c:c9:34:7f:70:d7:
    9d:2d:d1:48:4a:81:71:f8:12:dd:32:ba:64:31:60:
    08:26:4b:09:22:03:83:90:17:7f:f3:a7:72:57:bf:
    89:6d:e4:d7:40:24:8b:7b:bd:df:33:c0:ff:30:2e:
    e8:6c:1d""")

p_ranges, pmask_msk, pmask_val = msk(""" 0: e:  :  :  :c :c :  :  :  :b :  :  :  :  :
      :ab: e: 2: 8:c :  :  : 1:6 :6 : 6: f:d9: 0:
    8 :5c:7 :06:  :  :  :0 : 3:5 :4b:  :6 :  :  :
    2 :  :6 :  :  :  :2 :bc: c:  :85:1 : 1:d : 3:
     1:b4:  : b: 1: 3: d:a :  :  :6e: 0:b :2 :  :
      :b :  :9 :e :  :82:8d:  :  :13:  :  : a: a:
      :  :4 :  :c : f:  :  :7 :e :0a:  :  : b: 5:
      : e:91:3 :  :3c: 9:  : 6:  :  :b5:7d: 1:  :
      :  :  :b :a1:99:6 :4 :3 :c :1a:02:4 :  : 9:
    9 :f : d:bd:  :0 :  :  :  :b3:  : 4:  :e9: 9:
      : d:  :  :7 :  :93:  : e:dc:  : 0:  :e7:  :
    e :  :2 : b: 2:5 :  :  :  :  : c:5f:  :  :e2:
      :  : 9:  :2a:  : e:  :  :2 :  :9f: 7:3 :  :
    b : f:b :  : 8: 7:  :  :f :6 :e :c :  :3 :  :
    f7: 5: 8: 5:  :  :  :  :  : 8: e:  :03: c:  :
    33:76:e : 1:7 : c:  : 0:  :0b:  : a:  : 2: 9:
      :c8:bf:  :  :06: 7:d5:  :02: c:b :e2: 7:2 :
      :  """)

q_ranges, qmask_msk, qmask_val = msk(""" 0: f:  :d0: 1:55: 4:31:  : b:c4:8 :  : e: d:
    34: 3:f :  :  :  :  : 8:99:1 :  : a:0 :  :4 :
    0 :  :f :  :a4:41:2 :  :a :  : 1:  : a: c:  :
      :  : 9:  :  : 2:f4: f:  :  :  :  :1 : 4:9 :
    a :  :  :79:0 :  :  :  :  : 2: 8:b :  :4 : 8:
      :9b: 1:  :d :  :f :e4:  :4 :c :e :  :3 :  :
     7:2 :  :d :8 :2 :7 :  :d :67:fc:e : 0:f9: 7:
    8 :  :  :  :1 :2f:  :51:  :  :2e:0a: e:3d: 6:
    b :  :dd:  : 0:fb:  :f4:  :  :  :b4: 9:c :  :
     a:  :  :  :d :  :  :6b: 2:  :9b: a:60:  :d6:
     0:4f:16:d1:  :  :5 :fc:  :f :  :8 :  :  :  :
     1: 6:e1:9 : e:4 : 6: c: d:d :73: 3:  :  :7 :
      :8 : 9:  :3b:f : 2:  :  :f1: e:  :  :1e:  :
    8 :  :  : 6:0 : 4:99:e :  : 5:  :  : 4:  :  :
      : a:81:64:  :7 :f : 9: d:  :9 :  : 7:93:f :
    ac:8c:  : 8:  : 0: d: 8:  :7 :  :1d:  :f :  :
    1 :a :6 :8 :  :60:  :b3:  :  :  :89:  :  :14:
      :5 """)

_, dmask_msk, dmask_val = msk("""  :  :  : f:8 :a5:d : 2: 0:b :7 :  : 1:  : 4:
```

```
    1:0d:  :3 :  :6 :  :  : b:  :  :  :e :  :  :
   0e: 0:db:  :1a:1c:c0:  : e:  :  :99:bc:8 :a5:
   7 :7 :7 : b:  :  : 8: 8:  :7 :55: 2:  :  :f :
   b2:  :  :b :f :4 :  : 8:  :b :  :  :  : 0:  :
   0 :  :6 :9 :  :  :  : b: 4:  : 0: a: 5:07:b :
    9: c:9a: 9:  : 7:9e:  : b:60:f :  :  :  :0 :
     : 3:0 :  :  :  : 1:b :  :  : b: 6:0 :f :  :
     : 2:18: 6: b:1 :  :  :  :  :d3:f3:  :a :  :
    3:  :  :  :  : 3: d: 1: 2:7 :  : d:  : 2: d:
     :  : d:4 :  :d :  :6d: c:a :b6:  :  :  : 1:
   69:  : 7:  :89:  :c :8 :61: d:25: 3:7 :1b: 4:
   b :  :8 :55:  :49: 1:2 :3 :  :1 :e9:a8: 3:  :
   9 :  : 1:f8:d3:  :e :  :d :  :9 :b6:  :  :71:
   1 :  :c1:  : b: 1:  : 6:e :  :64:  :  :1a:c :
     : b:  :bf:c :  : 0:  : 8:a :4 :  :26:a :5 :
   6 :  :  :  :eb:  :e5: a:  :3e:f9:10:0 :  :  :
    6:0 :  : 8:  : 1:72: c:0 : f:5 : f:9c: 0: e:
    7:b :  :  :  :  :d9: 4:  : e:c :68:  :  :  :
    c:  :3a:  :  :a0:ea: 3: 4:  :72:a :d : 8:  :
     :0d:5 :0 : a: 7:c :bb: 6: 4:a :ce:d :2 : 1:
     :  :17:6 :  : c: b:  : f:  :3 : 5:6 :3 :0e:
     : 7:c :3e: 2: 9: 7: 6: f: e: f: 9:  :f3: 9:
   a :c1:6 :  : 1:9 :  :43:  : f: 5:  :0 :27: 4:
   4 :a :  :e9:  : 8: 4:3 :8a: 6:16:d5:c : e: e:
     :d : c:b :a8:  : 7:  : 9:  :7 :7d:  :  :  :
     :  :  :4 :2 :  : 3: 3: 6:  :  :  :7b:0 :  :
    e:  :0 :  :a :  : 5:  :  :  : 5:1 :82:c :0d:
   4 :2 :fd:36: 5:50:0 :  :  :d : f: 6:  :  :e :
   0 :  :  :ce:  :9e:8 :  :0 :d :07:b3:  :  :  :
   0 :e4:  :  :68:b :c :  : c:5 :  :  :3 : 7: 2:
    c:e0:  :5 :  :  :b4:  :ef: 7:  :1 :e : 0:f :
     :6 :  :  :  :e0:c :3 :  :  : 3:  : d:  :  :
    3: 3: c: a:  :b : a:71: 3: 0:a :  :4 :5d:  :
   0 :4 """)

_, dpmask_msk, dpmask_val = msk("""  : 3:2a:  : d:  :  :  :  :0 :1 : f:  :  : 6:
   1 :2 :1b:07: a:e :b :c5:58:7 :  :e8: 7: 1: c:
     : 1:b :a0: 4:0f:5 :67:  :3 :7 :6 :f9:  : c:
     :79: 0:1 :65:  :8 :  :99: d:d :  :2 :9 :0 :
    e:  :0 :  :  :  : d:  :d :7 :6 :a9: a:8b: b:
     :  : 7: a:37:  :  :7 :1 :6 :  :c2: 7:6 :b :
    e:  :  :  :  :  :  :b :3a:5 :  :  :  :  :  :
     :  :  :cd:8 :  : d:  :7 : 3:  : f:e : c:  :
     : a:  :c : f:c : 7:b :5 :  :  :2 :8 :8 :6 :
   0a: a:  :  :3 :db:  : 4:00:  : d:  :b : 5:  :
   20: 2: 5:  :82:  : 0: 6:  :8a:  :7 :  : 8:  :
    4: 1:  :  :  : 8:46:  :  :  :  :  : 0:f :c8:
   2 :  : c:7 :  : 1:  :  :2 : 0: 5:  :  : 1:9b:
    6:9 : 0:74:  :c :  :e :  :  :cb:b :3 :3 :  :
    2:  :  :47:  :2 : 0:5 :  :  : d: 6:83:  :  :
     :c7:  :  :0b:  :  : c:  :3 :8 :  :9 :4 : 7:
   5 :c0:fe:  :f9: 1:  :0 : e: 8:02:  : f:  :c :
   55:61""")

_, dqmask_msk, dqmask_val = msk("""  :0b:7 :4 :0 : 0:6 : 7:7e:  : 5:  : 7:  : a:
   a :d : 0: 6: 4:86:  :  :8 :  :  :  :  :e :8f:
    9:  :  :  : 1:  :2 :  : 7: b:1 :5 : f:  :8 :
     :d :21:  :e : d:  :c9:e : b:  :  :1 :  :  :
     :d :a2:b7:  :  :  :f3:  :42:  :e : c:  :f :
     : 0:f :7 : 4: 5:34:  :4 : c:  :  :8 :d : 8:
```

```
    5 :af: 3:1d: 5:4 :  :2 :  :6 :c : 6:a :1 :5 :
     a:9 :  :d :  :  :0a:a1:  :f :7 :9 :b :  :  :
     f:2 :27: f:  :0 :f6:4d:  :  :  :  :5 :  :
     4:08:  : 5:  : 8: 5:  :  :  :18: 4: 8:57: 2:
     f: a:  :  :a8: f: c:f : e: 1:9 :c : 4:9 :  :
      :  :  :  :  : 1:  :2 :  :d1:  : 6:e : d:  :
      : f:04:2 :8d:  : 3:  :  :b : 8:  :d6:  : 2:
      :  :  :6 :  : f:  :  : 0:6 :  :51:  :48:19:
      :  :  :69:4 : c:  :c :  : f:  :f4:d :  : f:
     d:0 :0d:b :3 : 3:2 :  :  :6 : b:5 :2 :  : c:
     1:5a: f:f :  :  :7e:3e:  :d :f :0 : d: c: 6:
     1""")


def search(K, Kp, Kq, check_level, break_step):
    max_step = 0
    cands = [0]
    for step in range(1, break_step + 1):
        #print " ", step, "( max =", max_step, ")"
        max_step = max(step, max_step)

        mod = 1 << (4 * step)
        mask = mod - 1

        cands_next = []
        for p, new_digit in product(cands, p_ranges[-step]):
            pval = (new_digit << ((step - 1) * 4)) | p

            if check_level >= 1:
                qval = solve_linear(pval, N & mask, mod)
                if qval is None or not check_val(qval, mask, qmask_msk, qmask_val):
                    continue

            if check_level >= 2:
                val = solve_linear(E, 1 + K * (N - pval - qval + 1), mod)
                if val is None or not check_val(val, mask, dmask_msk, dmask_val):
                    continue

            if check_level >= 3:
                val = solve_linear(E, 1 + Kp * (pval - 1), mod)
                if val is None or not check_val(val, mask, dpmask_msk, dpmask_val):
                    continue

            if check_level >= 4:
                val = solve_linear(E, 1 + Kq * (qval - 1), mod)
                if val is None or not check_val(val, mask, dqmask_msk, dqmask_val):
                    continue

                if pval * qval == N:
                    print "Kq =", Kq
                    print "pwned"
                    print "p =", pval
                    print "q =", qval
                    p = pval
                    q = qval
                    d = invmod(E, (p - 1) * (q - 1))
                    coef = invmod(p, q)

                    from Crypto.PublicKey import RSA
```

```
                    print RSA.construct(map(long, (N, E, d, p, q, coef))).exportKey()
                    quit()

            cands_next.append(pval)

        if not cands_next:
            return False
        cands = cands_next
    return True



def check_val(val, mask, mask_msk, mask_val):
    test_mask = mask_msk & mask
    test_val = mask_val & mask
    return val & test_mask == test_val


# K = 4695
# Kp = 15700
# Kq = 5155

for K in range(1, E):
    if K % 100 == 0:
        print "checking", K
    if search(K, 0, 0, check_level=2, break_step=20):
        print "K =", K
        break

for Kp in range(1, E):
    if Kp % 1000 == 0:
        print "checking", Kp
    if search(K, Kp, 0, check_level=3, break_step=30):
        print "Kp =", Kp
        break

for Kq in range(1, E):
    if Kq % 100 == 0:
        print "checking", Kq
    if search(K, Kp, Kq, check_level=4, break_step=9999):
        print "Kq =", Kq
        break
```

```python
#!/usr/bin/python
# coding=utf-8

#最优非对称加密填充
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

with open('./tmp/pubkey.pem', 'r') as f:
    key = RSA.importKey(f)
    N = key.n
    e = key.e
print N
print e

with open('./tmp/private.pem', 'r') as f:
    private = RSA.importKey(f)
    oaep = PKCS1_OAEP.new(private)

with open('./tmp/flag.enc', 'r') as f:
    print oaep.decrypt(f.read())
```

后记

暂时留空。