CTF-CYRPTO-RSA-Recovery

原创

大熊何在 于 2020-11-20 18:48:09 发布

分类专栏： CRYPTO

本文链接：https://blog.csdn.net/zippo1234/article/details/109857939

CTF-CYRPTO-RSA-Recovery

RSA-Recovery

题目分析

1. 私钥恢复

开始

1.题目

给出三个文件：

flag.enc、private.corrupted、pubkey.pem

很明显是加密结果、损坏的私钥、公钥。

2.分析

核心在于恢复损坏的私钥。Plaid CTF 2014、Jarvis OJ God Like RSA两题，都与此题类似。所以就直接借（zhao）鉴（chao）了。

PlaidCTF 2014 RSA writeup

RSA 私钥恢复和最优非对称加密填充

在此向大神表示崇高的敬意，我依然只能当个脚本小子。。。

3.私钥恢复

说实在话我根本没有看懂到底是在干嘛。这里有一个全过程的，可惜我依然看不懂，搬运过来给大家吧。

[Jarvis OJ] Crypto - God Like RSA

这位大神全程做下来。。。佩服。

（1）私钥结构

```
struct
       {
       BIGNUM *n;              // public modulus
       BIGNUM *e;              // public exponent
       BIGNUM *d;              // private exponent
       BIGNUM *p;              // secret prime factor
       BIGNUM *q;              // secret prime factor
       BIGNUM *dmp1;           // d mod (p-1)
       BIGNUM *dmq1;           // d mod (q-1)
       BIGNUM *iqmp;           // q^-1 mod p
       // ...
       };
```

我们来看下损坏的私钥

```
modulus:  //n
privateExponent:          //d
prime1:        //p
prime2:        //q
exponent1:         //dmp1,dpmask
exponent2:       // *dmq1,dqmask
```

（2）恢复原理

原WP的内容是这样

- given a candidate for (p mod 16**(t - 1)), generate all possible candidates for (p mod 16**t) (check against mask for prime1)
- calculate q = n * invmod(p, 16**t) (and check against mask for prime2)
- calculate d = invmod(e, 16**t) * (1 + k * (N - p - q + 1)) (and check against mask for private exponent)
- calculate d_p = invmod(e, 16**t) * (1 + k_p * (p - 1)) (and check against mask for exponent1)
- calculate d_q = invmod(e, 16**t) * (1 + k_q * (q - 1)) (and check against mask for exponent2)
- if any of checks failed - check next candidate

简单翻译一下

- 给定一个(p mod 16**(t - 1))的候选对象，生成(p mod 16**t)的所有候选对象，用p的掩码来校验
- 计算q = n * invmod(p, 16**t)，invmod是模逆，用q的掩码来校验
- 计算d = invmod(e, 16**t) * (1 + k * (N - p - q + 1))，用d的掩码来校验
- 计算d_p = invmod(e, 16**t) * (1 + k_p * (p - 1))，用dmp1的掩码来校验
- 计算d_q = invmod(e, 16**t) * (1 + k_q * (q - 1))，用dmq1的掩码来校验
- 如果其中任何一个校验失败，就尝试下一个候选对象。

这里所谓的掩码，其实就是把缺失的部分去掉拼起来（个人是这么理解的），便于比对。

（3）恢复脚本

先上脚本吧：

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/11/20 17:51
# @Author : A.James
# @FileName: test.py

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)
#TODO：计算掩码，便于比对
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)
#TODO：定义已知值，e和n从公钥中获取，私钥中得到的值计算掩码，便于比对
E = 65537
N = 785705503915674478216252691888832589081562285888751164070887872256010731816804311271574453527229389975220560
542433483558835639345202224585842869163839137870869178546286155294311870309616422398547316168960024116423211976O
507408310969536051302019378648702900755497029983527670025725020192295241573459604439276808554379010736923250430 4
278694004788487386575546637678471577415710132141629359359066210179559793470792727732925727989074755273352924530 7
536520604887446279661718983128257681383016307133765702513775538144912103623179426116122261336691710735434418728 1
851395136350201208510401957952279354329185249384823938597134169225616305425934997975644778018950076621136931774 1
768471316720291211010555838270998890938604887384230024107156831826677186738927686593444727141669106424701685253 7
547158727525636475803013786061663905631645926728185129791955158739978200217046060318508782227603578632671694194 0
134018545681335641114157777426391349568396299316237446836370796279146465694028078223569030224614312687616682856 8
```

```
259937733899648298228876414231362872766431287168324805754222317843284880350637710112697166926454540577853304865035259658632895567251545460503996817839313718876562297298045865318829110678014052830933857911513952161682522472975055 7

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
```

运行后得到

```
checking 25800
checking 25900
checking 26000
checking 26100
checking 26200
checking 26300
```

```
Kq = 26920
pwned
p = 300614320036585100877988716148693180113899403527981470301298063599759113920912353440422884096291432293110602
315494782118716437253944707605282118013106017677278348869422107184120875412343984530468950308585799898740358494 3
986733490687364235211242891485596799399873268522110837978483302777129327555887695260846205014634059144904682513 5
890871650866799299533696175818103240024841274114925018619060818213433528894936128306780366785977567327073724428 2
114452599836144676407851632977344479757236646598226734566832843943867237163440902328829904611743016099718050757 6
832875732595678460436440182715243126089692763316307469412167 9
q = 261366625455518298207469420516382283250251305191755366940082422086167744698707656848582880428190638371802435
011173102786325094132176765594845134816776890426233481888765989016424591702323609667546924343167960143144982638 0
023439053911881705007497842197381776464428774530288586127744722718028860520089413816858620738448417048151182868 0
117688324729381172912436910052489279406590356734739774635376711681212908417321705094537960645308009611045658947 3
592973731543955004676895324550176474506164474454442549103719229446201142345476552099706570637150283504185184171 0
577270788564858723310386934098567043026986294363013706705288 3
```

4.常规套路

剩下就是常规套路破解了。

```
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/11/20 16:48
# @Author : A.James
# @FileName: tt.py
import gmpy2
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import libnum
from Crypto.Util.number import *

p = 3006143200365851008779887161486931801138994035279814703012980635997591139209123534404228840962914322931106023154947821187164372539447076052821180131060176772783488694221071841208754123439845304689503085857998987403584943986733490687364235211242891485596799399873268522110837978483302777129327555887695260846205014634059144904682513589087165086679929953369617581810324002484127411492501861906081821343352889493612830678036678597756732707372442821144525998361446764078516329773444797572366465982267345668328439438672371634409023288299046117430160997180507576832875732595678460436440182715243126089692763316307469412167 9
with open('pubkey.pem', 'r') as f:
    pk = RSA.importKey(f.read())
n = pk.n
e = pk.e
with open('flag.enc','rb') as f:
    flag = f.read()
c = bytes_to_long(flag)
q = n//p
phi = (p-1) * (q - 1)
d = gmpy2.invert(e,phi)
print(d)
m = pow(c,d,n)
print(libnum.n2s(m))
```

5.get flag

```
flag{RSA_from_JarvisOJ}
```

结语

反正我是没有搞懂。。。只求下次遇到类似的题目能够灵活套用吧。