HGAME2023-writeup-Week3 by crumbling

Crypto

RSA大冒险2

challenge1，e很大，考虑维纳攻击；

challenge2，e，r不互素（参考了https://blog.csdn.net/weixin_44617902/article/details/109681910#:~:text=RSA%E2%80%93e%E4%B8%8E%CF%86%20%28n%29%E4%B8%8D%E4%BA%92%E7%B4%A0%E6%97%B6,%E6%AD%A3%E5%B8%B8%E7%9A%84rsa%E5%BA%94%E8%AF%A5%E6%9C%89e%E4%B8%8E%CF%86%20%28n%29%E4%BA%92%E7%B4%A0%EF%BC%8C%E4%BD%86%E6%98%AF%E8%BF%99%E9%A2%98%E5%B9%B6%E6%B2%A1%E6%9C%89%E8%80%8C%E4%B8%94%E4%B8%A4%E7%BB%84%E7%AD%89%E5%BC%8Fe%E4%B8%8E%CF%86%20%28n%29%E5%85%AC%E7%BA%A6%E6%95%B0%E9%83%BD%E4%B8%BA14%E3%80%82的处理）且p,q离得不是很远（平方差遍历）；

challenge3，p高位泄露（原理参考文章密码学学习笔记 之 Coppersmith's Method | Van1sh的小屋 (jayxv.github.io)；代码由Coppersmith.sage改编而来）

查资料看到sage实现Coppersmith.方法有small_roots和利用Coppersmith.sage，前者测试后（因为pq大小不确定，用的beta=0.4而不是0.5）仍需要爆破29位左右，所以从Coppersmith.sage改了点代码用，随后通过改动参数epsion，实现格的维数增加（测试中似乎模数M也在增大）以扩大上界到246位，并爆破剩余7位【总之在最后附上了sage代码和运行结果图】；

以下为3题解密代码（有亿点长）：

```
from __future__ import print_function
import libnum
from gmpy2 import *

#challenge1
def continued_fractions_expansion(numerator, denominator):  # (e,N)
    result = []

    divident = numerator % denominator
    quotient = numerator // denominator
    result.append(quotient)

    while divident != 0:
        numerator = numerator - quotient * denominator
```

```
        tmp = denominator
        denominator = numerator
        numerator = tmp

        divident = numerator % denominator
        quotient = numerator // denominator
        result.append(quotient)

    return result


def convergents(expansion):
    convergents = [(expansion[0], 1)]
    for i in range(1, len(expansion)):
        numerator = 1
        denominator = expansion[i]
        for j in range(i - 1, -1, -1):
            numerator += expansion[j] * denominator
            if j == 0:
                break
            tmp = denominator
            denominator = numerator
            numerator = tmp
        convergents.append((numerator, denominator))  # (k,d)
    return convergents


def newtonSqrt(n):
    approx = n // 2
    better = (approx + n // approx) // 2
    while better != approx:
        approx = better
        better = (approx + n // approx) // 2
    return approx


def wiener_attack(cons, e, N):
    for cs in cons:
        k, d = cs
        if k == 0:
            continue
        phi_N = (e * d - 1) // k
        # x**2 - ((N - phi_N) + 1) * x + N = 0
        a = 1
        b = -((N - phi_N) + 1)
        c = N
        delta = b * b - 4 * a * c
        if delta <= 0:
            continue
        x1 = (newtonSqrt(delta) - b) // (2 * a)
        x2 = -(newtonSqrt(delta) + b) // (2 * a)
        if x1 * x2 == N:
            return [x1, x2, k, d]
```

```
n =
553782095265004581296048924371045755664507381985058827436223686023724106108470248339037132692637072537085837626208238206694945279145962286705293740731358659277286342275660937990580020162349237936379177407897977717357917575867189349407799312092530195579155422283397123231551332287125206244950846263338513801370
e =
366220312665124070816946730302610781266785327012919989614244451877290710364824237909435496048265690376142730703424569465497261527343766823407170812408556054737331801370337229775276608434252496009781432110315661860365937075670157366615425625824015568282809129902858318906921803984238831137069786514082333334 1
c =
0x1501f566f4f56d490566df19c4a7eaf98e80696417e23e945f56ba6c846e77442ad8861590a3901b7f807477e8b5581ffa2ebd564275a45f327fa1ea8d1b4aa4378758546b94cd3d8d531832576bdf79a9c58e8aa83d67cea8e32da4994e580c767af6969944394e1274811369f24ffe1c6f7ff0abb588919324f4080fdec40c
expansion = continued_fractions_expansion(e, n)
cons = convergents(expansion)
p, q, k, d = wiener_attack(cons, e, n)
m = pow(c, d, n)
print(libnum.n2s(m))
#wiener_attack_easily!!! e很大 维纳攻击

#challenge2
from Crypto.Util.number import getPrime, long_to_bytes,inverse
n =
gmpy2.mpz(4961017322230839746998447525271366824239282266576117094434342831155192413928967158213949628552419429298887621746214156899163812821479408146845536594788490195083994722524046581470424778836680025308470808164188669620282475387191709577771773721958729664487331644091953796717165279223661153673788833931697040536 39)
c =
gmpy2.mpz(0x2265bc7f211657999b76579dae6d187d4e79556e343ca6bdba5719e023fa731df52d791e7b0e955d9857294e3d9c65a31c1f8452a3a1d6fadf0a2c48130a4a2397303e2eb1751495b9e2b0ceef9bfabfe9e745df5c6f219c7f49f3f5a198258a0ede1628624b94d002bc27eb21a0850b95b3036e4aa8391514051adb998828ba)
e =gmpy2.mpz(42718)

a=gmpy2.iroot(n,2)[0]
while 1:
    B2=pow(a,2)-n
    if gmpy2.is_square(B2):
        b=gmpy2.iroot(B2,2)[0]
        p=a+b
        q=a-b
        break
    a+=1

r= (p - 1) * (q - 1)
#gcd(e,r)=2
d = inverse(e//2, r)
flag=gmpy2.iroot(pow(c,d,n),2)[0]
flag=long_to_bytes(flag)
print(flag)
#how_to_solve_e_and_phi_uncoprime_condision
```

```
#challenge3
import random
from random import randint
from Crypto import Random
import math
n=119451025791476741962752740369458352457778273547549792449546184325595064995005
204448983678969062022812998345140613692490246682754881582165777543802195028097640
111033104811459943431343968907240976654541912284985153864478801828269700160309603
247907403952033044473851896326244953416865721568990607558861170642499
e=65537
leak=738704328696677176628312770872913074144597593268607196054054874119242625252
358
leak=(leak<<7)+80
p=
(leak<<246)+776759947513239762978714786402650396782227446634014180876853377486663
06567#左移右移的优先级也还蛮低的
q=n//p
c=0x29d3dc57b8e1b263898af2b8507c4a6c89fd6c4c74548d71ed9b963d546f92a649262c326015
4ce84646cae0cefc2caa48aad154d2f18034aabfe61741c4d87835d9fea129b8b36686fd9118ee52
4f9e964b34eb79c5031678380eb2b1478972a94c5692021f5f1af7841a601da12be502fbcad37c9c
68e33361e524f172718
#512为p高位攻击需要约288位，但这里只有253位，差距过大，爆破也不容易。还有什么别的路吗
r= (p - 1) * (q - 1)
d = inverse(e, r)
flag=pow(c,d,n)
flag=long_to_bytes(flag)
print(flag)
#'now_you_know_how_to_use_coppersmith
```

```
from __future__ import print_function
import time
from Crypto.Util.number import getPrime

def coppersmith_howgrave_univariate(pol, modulus, beta, mm, tt, XX):
    dd = pol.degree()
    nn = dd * mm + tt
    polZ = pol.change_ring(ZZ)
    x = polZ.parent().gen()

    gg = []
    for ii in range(mm):
        for jj in range(dd):
            gg.append((x * XX)**jj * modulus**(mm - ii) * polZ(x * XX)**ii)
    for ii in range(tt):
        gg.append((x * XX)**ii * polZ(x * XX)**mm)

    BB = Matrix(ZZ, nn)

    for ii in range(nn):
        for jj in range(ii+1):
            BB[ii, jj] = gg[ii][jj]

    BB = BB.LLL()
```

```
    new_pol = 0
    for ii in range(nn):
        new_pol += x**ii * BB[0, ii] / XX**ii

    potential_roots = new_pol.roots()
    print("\n# Solutions")
    print("potential roots:", potential_roots)


N =
119451025791476741962752740369458352457778273547549792449546184325595064995005204448983678969062022812998345140613692490246682754881582165777543802195028097640111033104811459943431343968907240976654541912284985153864478801828269700160309603247907403952033044473851896326244953416865721568990607558861170642499
qbar =
738704328696677176628312770872913074144597593268607196054054874119242625252358;
q0=qbar<<7
beta = 0.5
epsilon = beta / 40
for i in range(2**7):
    qbar=q0+i
    hidden = 246
    qbar = qbar<<hidden
    F.<x> = PolynomialRing(Zmod(N), implementation='NTL')
    pol = x + qbar
    dd = pol.degree()
    mm = ceil(beta**2 / (dd * epsilon))
    tt = floor(dd * mm * ((1/beta) - 1))
    XX = ceil(N**((beta**2/dd) - epsilon))
    start_time = time.time()
    roots = coppersmith_howgrave_univariate(pol, N, beta, mm, tt, XX)
    print(i)
    print(("in: %s seconds " % (time.time() - start_time)))
```

```
78
in: 16.215312480926514 seconds

# Solutions
potential roots: []
79
in: 16.373196840286255 seconds

# Solutions
potential roots: [(776759947513239762978714786402650396782227446634014180876853377486663065 67, 1)]
80
in: 15.584603786468506 seconds

# Solutions
potential roots: [(-354022173944926207954595614072817453347362247366381952320974591340613590 97, 1)]
81
in: 15.125889301300049 seconds
```

ezBlock

太菜了，妹成功，关于差分分析还是稍微有些迷糊，或许还需要1、2天才弄得清楚吧

```
from Crypto.Util.number import *
import random
```

```
def s_substitute(m):
    c = 0
    s_box = {0: 0x6, 1: 0x4, 2: 0xc, 3: 0x5, 4: 0x0, 5: 0x7, 6: 0x2, 7: 0xe, 8:
0x1, 9: 0xf, 10: 0x3, 11: 0xd, 12: 0x8,
             13: 0xa, 14: 0x9, 15: 0xb}
    #单表替换表
    for i in range(0, 16, 4):
        t = (m >> i) & 0xf#4位取
        t = s_box[t]#查表
        c += t << i#对应放回
    return c
#制作inv_s_box?
inv_s_box=
{6:0x0,4:0x1,12:0x2,5:0x3,0:0x4,7:0x5,2:0x6,14:0x7,1:0x8,15:0x9,3:0xa,13:0xb,8:0
xc,10:0xd,9:0xe,11:0xf}
def enc(m, key):
    n = len(key)
    t = m
    for i in range(n - 1):
        t = t ^ key[i]
        t = s_substitute(t)
    c = t ^ key[n - 1]
    return c


"""
f = flag[6:-1]
assert flag == 'hgame{' + f + '}'
key = [int(i, 16) for i in f.split('_')]#遇_分割 有4个即5块
print(len(key))
m_list = [i * 0x1111 for i in range(16)]#选择了特殊的明文，也就是说 以下为进行了差分分析
的第一步？
c_list = [enc(m, key) for m in m_list]#主要加密段
print(c_list)
"""

key="xxxx_xxxx_xxxx_xxxx_xxxx"
s_box = {0: 0x6, 1: 0x4, 2: 0xc, 3: 0x5, 4: 0x0, 5: 0x7, 6: 0x2, 7: 0xe, 8: 0x1,
9: 0xf, 10: 0x3, 11: 0xd, 12: 0x8,
             13: 0xa, 14: 0x9, 15: 0xb}
m_list = [i * 0x1111 for i in range(16)]
encs=['6fae', '8497', '763b', '1524', '2d09', '0c11', 'b74c', 'e88d', '3273',
'936f', 'fa5a', 'd1b0', '49f5', 'c0e2', '5bd8', 'aec6']
encs=[28590, 33943, 30267, 5412, 11529, 3089, 46924, 59533, 12915, 37743, 64090,
53680, 18933, 49378, 23512, 44742]
#进一步拆分成单个字节来分析  进行多次差分攻击（每次将未知的最开始的输入和最后的输出中间作为一个
整体的sbox）
#0^a=a  key[0] 查表 ^key[1] ... ^key[4]
#1^a=a翻转 key_0 查表 ...
#key_0+key[0]=全1

#https://www.bilibili.com/video/BV1jt4y1z7Tb/?spm_id_from=333.337.search-
card.all.click&vd_source=230d5993c02c8821538d172f6b14bc0d
#https://blog.csdn.net/weixin_43734797/article/details/124055427
s = []
```

```
def creat_s_table():#制作分布表
    for i in range(16):
        s.append([])
        for j in range(16):
            s[i].append(0)
    for i in range(16):
        for j in range(16):
            s[s_box[i]^s_box[j]][i^j]+=1
    for i in range(16):
        print(s[i])
creat_s_table()
def get_possible(inputdiff,outputdiff):
    for i in range(16):
        a=i^inputdiff#求出输入对的另一个值
        if s_box[a]^s_box[i]==outputdiff:
            print(a,"+",i,"-->",s_box[a],"+",s_box[i])

def get_p(chain):
    p=1#/(16**4)
    for j in range(4):
        p*=s[chain[j+1]][chain[j]]
    return p

#通过输出/输入 差分获得差分变化链
def get_chain(indiff,outdiff):
    max=1
    for i1 in range(16):
        t = [indiff]
        if s[i1][indiff] != 0:
            k1 = i1
            t.append(k1)
            for i2 in range(16):
                t=[indiff,k1]
                if s[i2][k1]!=0:
                    k2=i2
                    t.append(i2)
                    for i3 in range(16):
                        t=[indiff,k1,k2]
                        if s[i3][k2] != 0:
                            k3 = i3
                            t.append(i3)
                            for i4 in range(16):
                                t = [indiff, k1, k2,k3]
                                if s[i4][k3] != 0 and i4==outdiff:
                                    t.append(i4)
                                    if max<get_p(t):#找概率最大的那条路线(有相等的可能
这咋整)
                                        max=get_p(t)
                                        possible_line=t
    return possible_line

def get_key_p(p,j,t,pkey,input1):
    # 生成密钥概率表
    count = s[t[j+1]][t[j]]
    for i in range(16):
```

```
        a = i ^ t[j]  # 求出输入对的另一个值
        if s_box[a] ^ s_box[i] == t[j+1]:
            pkey[i^input1]+=1/count

m=[i for i in range(16)]
input=0
for keyround in range(4):
    c = 0
    for bits in range(0, 16, 4):
        pkey0 = [0 for i in range(16)]
        for i in range(16):
            t = get_chain(m[input] ^ m[i], ((encs[0] >> bits) & 0xf) ^ ((encs[i]
>> bits) & 0xf))  # 获得对应输入输出差分的最高概率差分链
            get_key_p(get_p(t),keyround, t, pkey0, input)
            for j in range(16):
                if max(pkey0) == pkey0[i]:
                    c += i << bits
    print(c)
```

ezDH

是叫ECDH？总之就是题目先进行DH得到shared_secret，然后进行椭圆加密。根据最后椭圆加密的代码，可以代换得到m=c-P1*shared_secret；而与求解shared_secret有关的一个是椭圆曲线的离散对数问题（Pa=G×shared_secret），一个是DH（同样是离散对数问题:求解g^x=A mod p）。

这里从后者入手（主要是前者求解失败了）

sage求解（(12条消息) sage之离散对数求解*ckm1607011的博客-CSDN博客*discrete_log）获得alice的私钥（也许不是，因为在我没有找到正确方法前的尝试中发现g似乎并不是模数n的原根，此处求解的x可能是另一个能得到相同结果的值？）

以下为代码（好像排得有点乱，因为做题的时候python和sage混着用，所以现在有点搞不清楚了，但总之数据应该是没有问题的）：

```
from Crypto.Util.number import *
#sage
N=0x2be227c3c0e997310bc6dad4ccfeec793dca4359aef966217a88a27da31ffbcd6bb271780d8b
a89e3cf202904efde03c59fef3e362b12e5af5afe8431cde31888211d72cc1a00f7c92cb6adb17ca
909c3b84fcad66ac3be724fbcbe13d83bbd3ad50c41a79fcdf04c251be61c0749ea497e65e408dac
4bbcb3148db4ad9ca0aa4ee032f2a4d6e6482093aa7133e5b1800001
A=0x22888b5ac1e2f490c55d0891f39aab63f74ea689aa3da3e8fd32c1cd774f7ca79538833e9348
aebfc8eba16e850bbb94c35641c2e7e7e8cb76032ad068a83742dbc0a1ad3f3bef19f8ae6553f39d
8771d43e5f2fcb986bd72459456d073e70d5be4d79ce5f10f76edea01492f11b807ebff0faf6819d
62a8e972084e1ed5dd6e0152df2b0477a42246bbaa04389abf639833
B=0x1889c9c65147470fdb3ad3cf305dc3461d1553ee2ce645586cf018624fc7d8e566e04d416e68
4c0c379d5819734fd4a09d80add1b3310d76f42fcb1e2f5aac6bcdd285589b3c2620342deffb7346
4209130adbd3a444b253fc648b40f0acec7493adcb3be3ee3d71a00a2b121c65b06769aada82cd14
32a6270e84f7350cd61dddc17fe14de54ab436f41b9c9a0430510dde
```

```
R =
GF(687219428995849307159753802176826720614443448112639362736756075536712284271368245160895405670269902708573338310356219613086190099255844887741784618855016920048234684293006570562990541691001806147536244341734294184552071957617135670324033740343310715129099061775681349948606808564402940626123330630160286881579340386334965540482118792747008000000000001)
h =
R(540796674413309590547032066175613285958637901666634897565628660345827128531895632332769254527238568105185551546884800907144544023580239099170312695514311814299629198269815034460915665907705550408976290234380655787714985060565068167472978255265037935146155973659529920142339151638254231864440665367013821709077111288852709366498638723330482641100314675)
g = R(2)
x = discrete_log(h, g)
assert g**x == h
p=686479766013060971498190079908139321726943530014330540939446345918554318339765605212255964066145455497729631139148085803712198799971664381257402829111505715 1
a=-3
b=109384903807373427451111239076680556993620759895168374899458639449595311615073501601370873757375962324859213229670631330943845253159101291214232748847898598 4
E = EllipticCurve(GF(p), [a, b])
#shared_secret=pow(B,x,N)
shared_secret=210079986310539059026469300720655426951919393335485123618687987029746658763798439576376646226161293006261472023436870446704160814487044427272498126726277523426054334489402918696632471053566567964288295442264373353184045686758631952518445700562724692050307270259376021103356466045824174058551054535796277627154836626119222585862131679625673279898703654
G=E(62058779183337702873234036705436617341291700859541987678208619622611742026469763791817352577598677606558357118451443264706138823954459754822198698282109759153475351956909044812130266914587199895248867449669290021764126870271692995160201860564302206748373950979891071705183465400186006709376501382325624851012261206)
P1=E(20326389595757377985537342389531770656710211124500024718242257344917356046000030284917291314457344324425102019559774724087284152270187464672501070804830736473510147080793750133751646930018687527128938175786714269902604502700248948154299853980250781583789623838631244520649113071664767897964611902120411142027848868)
c=E(667037343734418040412798382148217814937411681754468809498641263157585402138545967685447533506836969887598813500969818725552350184101343089213337157798748052247533506836969887598813500969818725552350184101343089213337157798748052247700716101983207165347315916182076928764076602008846695049181874187707051395)
m=c-P1*shared_secret
print(m)#m=132921474085671049652309683999312068131632119453884168476253021086106037584159648226385219814
```

Reverse

其实费了点时间，但太菜了什么都没弄明白，时间还是不太够用。

脑中的想法：ak密码后玩逆向——>什么？cpp有重磅hint，写完2题密码就去逆向——>卒