Week2 Crypto+noisy_pic_encode WP

[Week2] Weiner?

挺好一个题，推式子

$$e(\phi(n)-\alpha)\ \equiv 1\ \ mod\ \phi(n)$$

$$\alpha\ \Rightarrow 540\ bit$$

$$e\alpha\ \equiv\ \ -1\ \ mod\ \phi(n)$$

连分数的界不满足，换`Boneh Durfee Attack`的式子打**二元coppersmith**

$$e\alpha = k\phi(n)-1$$

$$k\phi(n)-1\ \equiv 0\ \ mod\ e$$

$$k(N+1-p-q)-1\ \equiv 0\ \ mod\ e$$

$$2k(\frac{N+1}{2}+\frac{-p-q}{2})-1\ \equiv 0\ \ mod\ e$$

$$A=\frac{N+1}{2}$$

$$y=\frac{-p-q}{2}$$

$$x=2k$$

$$\Rightarrow\ f(x,y)=-1+x*(A+y)\ \ mod\ e$$

记得加上`d`参数😐

```
from Crypto.Util.number import *
import itertools
import gmpy2

#coppersmith
def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()
    R = f.base_ring()
    N = R.cardinality()
    f /= f.coefficients().pop(0)
    f = f.change_ring(ZZ)
    G = Sequence([], f.parent())
    for i in range(m + 1):
        base = N ^ (m - i) * f ^ i
```

```
        for shifts in itertools.product(range(d), repeat=f.nvariables()):
            g = base * prod(map(power, f.variables(), shifts))
            G.append(g)
    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)
    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)
    B = B.dense_matrix().LLL()
    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, 1 / factor)
    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B * monomials):
        H.append(h)
        I = H.ideal()
        if I.dimension() == -1:
            H.pop()
        elif I.dimension() == 0:
            roots = []
            for root in I.variety(ring=ZZ):
                root = tuple(R(root[var]) for var in f.variables())
                roots.append(root)
            return roots
    return []

N =
1314596361022263601788434137011403172147643922847975466136258513357215772850621
2347596527067586650222459274406620426552790297738353152849720423437462560555681
6517580494897658435650751133340274048678178623575098042428929574567517762442336
2628850033135579266133310939066794989576020744982707207831070003406571869291023
2295467839036884860293554895052679418605488806127749209769365402536184743143796
3673920788908534300700742533165771733390761822749508651625602220145681531660759
0101323411864355697243931573114759377441529089731272636561720060861764765971073
9072455612784775579390431491294765545111404573041044115550645441
e =
8960889569116202817034041199600680457143327503651899780353738081347111381092127
9244852492024595905722103481031999798568207838828059958846690492837655050222036
0742257718372037761622864282080761983722863756008088415882634026001060060311173
5199435770236093318278326212741151164635443844291353503075658946949144377722037
3129465680277165665513263247100111142420305926137758839791038091692215211730254
7901705495476187536674910249683601732034854930638521303031734659120486284972299
5807229768275322657097111993317733988852667371020548600116650016607706180857563
5084108036481345785746544136911337919609398789241299061758183831
c =
4106930513742442178324611549616242987951988433582714914604753207421555091402166
1867828069420188146927017222097255410240285578345494623679531286292618699454866
```

```
8289226849528868841259345841327272428474631816162949074164301237629452793623895
7273363589386765557845509163716495965025308600250452959168677833132244061677131
7982980843863506600010845273263173533650662414588269425278932849918308542129012
9959542829695786508487349445912131793234336012192447192855865782833460017721973
9853077634851134938719870083927340099859510189847262177665373373127245818697220
878423310672114513284418556110527390296708350510317213859422388
A = int((N+1)/2)

R.<x,y> = Zmod(e)[]
f = x*(A+y)-1
roots = small_roots(f,[2^540,2^1024],m=4,d=5)

y = roots[0][1]
sum = (2*y*(-1))%e
p = (gmpy2.iroot(sum*sum-4*n,2)[0]+sum)//2
q = n//p
phi = (p-1)*(q-1)
d = gmpy2.invert(e,phi)
m = pow(c,d,n)
print(long_to_bytes(m))

#b'SBCTF{still_not_enough_entropy_:(}'
```

[Week2] ez_block

搬运选手(*Libr)的wp

是个aes_cbc的块密码。可以不管中间AES的细节原理，直接看外层性质。

```
C0 = iv
C(i) = AES_ECB_encrypt(C(i-1)^P(i))
```

给我们的内容中包括了一个完整的C(n)，P，以及前几个块加密内容的某几位。还有缺失两位的key。

```
AES_ECB_decrypt(C(i)) ^ P(i) = C(i-1)
```

所以爆破密码的最后两位，解密的数据和前文比较。可以爆破出key

```
from Crypto.Cipher import AES
import binascii
```

```
from base64 import b64decode


def xor(a, b):
    return bytes([x ^ y for x, y in zip(a, b)])


hexify = binascii.hexlify
cipher = b64decode(b64decode(open("cipher.txt", "r").read()))
key = "3N7g309d6Y7enT**"
message = "Security is not a joke, mind it. But complete security is a
myth".encode()
print(len(cipher), len(message), len(key))
bcipher = [cipher[i : i + 32] for i in range(0, len(cipher), 32)]
p = [message[i : i + 16] for i in range(0, len(message), 16)]
known = bytearray.fromhex(bcipher[-1].decode())
# print(known)
print(p, bcipher)
det = [bcipher[-1].decode()]
get_key = ""
for i in range(0, 128):
    for j in range(0, 128):
        nkey = key[:-2] + chr(i) + chr(j)
        dec = AES.new(nkey.encode(), mode=AES.MODE_ECB).decrypt(known)
        dec = xor(dec, p[-1])
        dec = hexify(dec).decode()
        if dec[-4:] == bcipher[-2][-4:].decode():
            assert dec[:2] == bcipher[-2][:2].decode()
            print(dec, bcipher[-2].decode(), nkey)
            det.append(dec)
            print("Found key", nkey)
            get_key = key[:-2] + chr(i) + chr(j)
            break
```

然后依次和原文异或后解密得到全部密文。

```
aes = AES.new(get_key.encode(), mode=AES.MODE_ECB)
realcipher = [bcipher[-1]]
for i in range(3):
    r = p.pop()
    c = bytearray.fromhex(bcipher.pop().decode())
    print(r, c.hex())
    dec = aes.decrypt(c)
    dec = xor(dec, r)
    dec = hexify(dec).decode()
```

```
        print(dec)
        bcipher[-1] = dec.encode()
        realcipher.append(dec.encode())
    realcipher.reverse()
    print(realcipher)
```

然后有

```
    AES_ECB_decrypt(C(1)) ^ P(1) = C(0) = IV = flag
```

可以得到IV。

```
    p = [message[i : i + 16] for i in range(0, len(message), 16)]
    t = xor(aes.decrypt(bytearray.fromhex(realcipher[0].decode())), p[0])
    print(t)
```

[Week2] ez_rsa

common_prime RSA，经验多的师傅应该一眼可以认出来，这里采用***CRYPTANALYSIS OF RSA AND ITS VARIANTS***一书中P203的攻击方法

Attack 11.1. *Let N be a common prime RSA modulus with balanced primes having common factor $g = N^{\gamma}$. It is expected that the modulus can be factored with $O(N^{1/4-\gamma/2})$ operations, each requiring time polynomial in $\log(N)$.*

Their method is a modification of Pollard's rho method. In particular, the usual map $x \mapsto x^2 + 1 \bmod N$ is replaced with $x \mapsto x^{N-1} + 3 \bmod N$. Letting $f(x) = x^{N-1} + 3 \bmod N$, and starting with some initial value $x_1 = x_2$, the attack consists of repeatedly computing

$$x_1 = f(x_1)$$
$$x_2 = f(f(x_2)),$$

until $\gcd(x_1 - x_2, N) > 1$ and the factorization is revealed. Since $N - 1 = 2gh$ and $p - 1 = ga$ there can be at most a values of $x^{N-1} \bmod p$. Thus the expected number of steps before a collision (*i.e.*, $d > 1$) is found is $O(\sqrt{a}) = O(N^{1/4-\gamma/2})$. For any integer $\ell > 0$, when the common factor has size

$$\gamma \approx \frac{1}{2} - \frac{2\ell}{\log_2(N)},$$

exp:

```
from Crypto.Util.number import *
from gmpy2 import invert

f = lambda x,n: (pow(x, n - 1, n) + 3) % n
def phllard_rho(n):
    i = 1
    while True:
        a = getRandomRange(2, n)
        b = f(a, n)
        j = 1
        while True:
            p = GCD(abs(a - b), n)
            if p == n:
                break
            elif p > 1:
                return (p, n // p)
            else:
                a = f(a, n)
                b = f(f(b, n), n)
            j += 1
        i += 1


n =
```

```
c =
e =

p,q = phllard_rho(n)
d = invert(e,(p-1)*(q-1))
print(long_to_bytes(pow(c,d,n)))
```

[Week2] strange_rsa

Condition on composite numbers easily factored with elliptic curve method

Masaaki Shirase

Future University Hakodate,
shirase@fun.ac.jp

Abstract. For a composite integer N that we would like to factor, we consider a condition for the elliptic curve method (ECM) using N as a scalar value to succeed and show that if N has a prime factor p such that $p = (DV^2+1)/4$, $V \in \mathbb{Z}$, $D \in \{3, 11, 19, 35, 43, 51, 67, 91, 115, 123, 163, 187, 235, 267, 403, 427\}$, we can find a non-trivial divisor of N (multiple of p) in a short time. Although, Cheng already provided the same result for $D \in \{3, 11, 19, 43, 67, 163\}$ [2], this paper uses another approach. In other words, to factor N, Cheng's work uses the ECM using the N-th division polynomial and this paper uses the ECM using arithmetic on a residue ring of $\mathbb{Z}_N[X]$. In the authors' implementation on PARI/GP, a 1024-bit N was factored in a few seconds when p was 512 bits.

Keywords: Prime factorization, Elliptic curve method, Class polynomial, Residue ring

是这篇，但github有轮子了，应该也在某些比赛被拿来魔改过了，我只在cryptohack见过一次，开场被打烂也正常

```
#D的取值:3, 11, 19, 35, 43, 51, 67, 91, 115, 123, 163, 187, 235, 267, 403, 427

#modified from https://github.com/crocs-muni/cm_factorization
class FactorRes(object):
    def __init__(self, r=None, c=None, u=None, a=None):
        self.r = r
        self.c = c
        self.u = u
        self.a = a
```

```
def xgcd(f, g, N):
    toswap = False
    if f.degree() < g.degree():
        toswap = True
        f, g = g, f
    r_i = f
    r_i_plus = g
    r_i_plus_plus = f
    s_i, s_i_plus = 1, 0
    t_i, t_i_plus = 0, 1
    while (True):
        lc = r_i.lc().lift()
        lc *= r_i_plus.lc().lift()
        lc *= r_i_plus_plus.lc().lift()
        divisor = gcd(lc, N)
        if divisor > 1:
            return divisor, None, None
        q = r_i // r_i_plus
        s_i_plus_plus = s_i - q * s_i_plus
        t_i_plus_plus = t_i - q * t_i_plus
        r_i_plus_plus = r_i - q * r_i_plus
        if r_i_plus.degree() <= r_i_plus_plus.degree() or
r_i_plus_plus.degree() == -1:
            if toswap == True:
                return r_i_plus, t_i_plus, s_i_plus
            else:
                return r_i_plus, s_i_plus, t_i_plus,
        r_i, r_i_plus = r_i_plus, r_i_plus_plus
        s_i, s_i_plus = s_i_plus, s_i_plus_plus
        t_i, t_i_plus = t_i_plus, t_i_plus_plus

def Qinverse (Hx, a, N):
    r,s,t = xgcd(a.lift(), Hx, N)
    if (s,t) == (None, None):
        res = r, 0
    else:
        rinv = r[0]^(-1)
        res = 1, s * rinv
    return res

def CMfactor(D, N):
    Hx = hilbert_class_polynomial(-D)
    res = FactorRes()
    ZN = Integers(N)
    R.<x> = PolynomialRing(ZN)
    Hx = R(Hx)
    Q.<j> = QuotientRing(R, R.ideal(Hx))
```

```
    gcd, inverse = Qinverse(Hx, 1728 - j, N)
    if gcd == 1:
        a = Q(j * inverse)
    return CMfactor_core(N, a, Q, ZN, Hx, res)

def CMfactor_core(N, a, Q, ZN, Hx, res):
    for c in [1..10]:
        E = EllipticCurve(Q, [0, 0, 0, 3 * a * c ^ 2, 2 * a * c ^ 3])
        for u in [1..10]:
            rand_elem = ZN.random_element()
            res.rand_elem = int(rand_elem)
            w = E.division_polynomial(N, Q(rand_elem),
two_torsion_multiplicity=0)
            poly_gcd = xgcd(w.lift(), Hx, N)[0]
            r = gcd(ZZ(poly_gcd), N)
            res.c = c
            res.u = u
            if r > 1 and r != N:
                return r, N//r

def main():
    #sys.setrecursionlimit(50000)

#####################################INPUTS####################################
    d =
    n =
    c =
    e =

#####################################INPUTS####################################

    p, q = CMfactor(d, n)

    phi = (p - 1) * (q - 1)
    d = pow(e, -1, phi)
    flag = int(pow(c, d, n))
    print(flag.to_bytes((flag.bit_length() + 7) // 8, 'big').decode())

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

[Week2] hard_DSA

DSA里的签到题，所以叫 `hard_DSA` ，本质还是个共享k的DSA attack，不过这里用的二次曲线，但方法都类似，最后化到 `mod q` 意义下解单变量多项式就好了，DSA基础还不会去wiki自己学，简单带着推三行

$$s_1k_1 = h_1 + xr_1$$

$$s_2(a^2k_1 + bk_1 + c) = h_2 + xr_2$$

$$as_2r_1^2x^2 + (2as_2h_1r_1 + br_1s_1s_2 - r_2s_1^2)x + (as_2h_1^2 + bs_1s_2h_1 + cs_1^2s_2 - h_2s_1^2) \equiv 0 \ \ mod \ q$$

exp:

```
from Crypto.Util.number import *
import gmpy2

p =
1872031591043158059193940353592116839365521280346275200001068397138632267936019
0523709798850878500492542988467798727805228514893709963045091700762204270704740
5540953261004493187227349318887899177184472339714989927443158004631485243809510
3125094017361118710622974249110055586588662243291975014783052367828885723
q = 1285718328334833204968780401116235523937945665141
g =
1390669380253508027717671015782610502765161746128567191411026064872592119873088
6918061797725067720964235230209822970050765804957525927628346065653557382723266
3689549468009206694512592890959156892314115797147147483431102168317849991735362
617832602604262224188572249960945313463130156739956202185587397119030604
y =
1313152181726722609739389492301050134299256968070154304258731622474208252181606
0500430041525716356566259638630640361811093641180422936139664689766596819989866
6941383438934510069368383758469939195380228999154368021040812035796407395947612
694797409862485108118415201125855895074058546503575350869633028961617513
h1 = 920626813732182387519470617510060777584519262688
r1 = 1227129929482183160818861021370743307541786073496
s1 = 83817863030083524148643364160282703231165130419
h2 = 1289090588504546329066834974120515061281237038274
r2 = 366168901249053092743356502931093123937744582601
s2 = 561267834645473985672611391537861808432196241940
a = 792199365162861065507319
b = 670145214693898828671636
c = 832546118137422553660585

A = a*s2*r1*r1
B = 2*a*s2*h1*r1+b*r1*s1*s2-r2*s1*s1
C = a*s2*h1*h1+b*s1*s2*h1+c*s1*s1*s2-h2*s1*s1

P.<x> = PolynomialRing(Zmod(q))
f = A*x^2+B*x+C
x = f.monic().roots()
print(x)
```

```
#[(1081314590957984661565230749837987983260015390204, 1),
(153403194845674357474881951660365459489, 1)]

print(long_to_bytes(153403194845674357474881951660365459489))
#b'shared_k_nonOn0!'
```

[Week2] noisy_pic_encode

预期解是PCA或者SVD搞一下，一血做法是手动提取了一些奇奇怪怪的数据特征，确实也是可以的

```
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
A = np.load("problem.npy")
pca = PCA()
pca.fit(A)
data_pca = pca.transform(A)
plt.scatter(data_pca[:, 2], data_pca[:, 3], s=0.01)
plt.gca().invert_yaxis()
plt.gca().invert_xaxis()
plt.axis("equal")
plt.show()
```

十行exp，不能再多了