AntCTF x D^3CTF_Crypto_部分复现

原创

M3ng@L 于 2022-03-12 17:03:34 发布 82 收藏

分类专栏： CTF比赛复现 文章标签： Crypto python

本文链接：https://blog.csdn.net/qq_51999772/article/details/123446566

AntCTF x D^3CTF_Crypto_部分复现

Reference： AntCTF & D^3CTF 2022 By W&M - W&M Team (wm-team.cn)

d3factor

Descripiton

```
from Crypto.Util.number import bytes_to_long, getPrime
from secret import msg
from sympy import nextprime
from gmpy2 import invert
from hashlib import md5

flag = 'd3ctf{'+md5(msg).hexdigest()+'}'
p = getPrime(256)
q = getPrime(256)
assert p > q
n = p * q
e = 0x10001
m = bytes_to_long(msg)
c = pow(m, e, n)

N = pow(p, 7) * q
phi = pow(p, 6) * (p - 1) * (q - 1)
d1 = getPrime(2000)
d2 = nextprime(d1 + getPrime(1000))
e1 = invert(d1, phi)
e2 = invert(d2, phi)

print(f'c = {c}')
print(f'N = {N}')
print(f'e1 = {e1}')
print(f'e2 = {e2}')
'''
c =
N =
e1 =
e2 =
'''
```

Analysis

由题，

$N=p\cdot\cdot\cdot phi=$

其中因为

```
d1 = getPrime(2000)
d2 = nextprime(d1 + getPrime(1000))
```

所以d 和d 之间的关系可以表示为$d = d +$

而我们需要知道 以及

经过简单的替代以及同余式两边同时乘以e

$e\cdot(d+\cdots)=$

399.pdf (iacr.org) 的部分结论

$\because phi=p\cdot(p-1)$

我们可以先计算a的大小（同余式参数均已知）

引入$coppersmith$攻击

> 在e阶的多项式f中，可以给定β，快速求出模某个b意义下较小的根，其中$b > n$ ，且b是 的因数
> 在sagemath中使用的函数即是 `small_roots`

可以发现现在我们所有的条件满足`coppersmith`定理的攻击条件

$g(x)$的p 是N的因数，而确实可以找到一个β使得$p \geq n$

所以我们可以解得x的实际大小，进而表示出p

$\therefore e$

Solving code

```
# sagemath
from Crypto.Util.number import *
import gmpy2,hashlib

c = 24206246313154736733887320743404102156573780967370209767226035295988643385324042248792190591059500056551007283611984995508624056600435919196815686117079675
N = 14767514276330719775995719833011510632583767311029559753641111470372046142203768837520322534078815682905200595153404346328587346894392684793994823155060434255411626465233884378421491251784478006161370442199165869422078386740010040072378614701764545437187521823123180684660517130879273706701775146668608223413804941540770204728147061232098657690487223808881754017918732738502813841473940750549501690021653574907965109508526312876897473604363841637582891597102644697220363208191233137733010727778444578953887977426315411011528190891502814898976835084000986938084735422129638688344852338581282200557278043264513100807915
e1 = 42573500601851832192011385837169104623329139427077913921653137926682945366570465686824588430957474130074612194672434453245633749049226369098972790483737427917560662340402559853340540067732991663330758581384963507109726898990642677186441085255638127911758849626278714658841487372398385504141547684044585017145753097722198112500610774110077952920916344640558569668218645201366964350727562043949202101954492291394147262487410260424937699061632388433129366011615678289193521757530889579162382630610069205913194549508465485452183401618145250832943010281366371333360845989891536174521587130554706932512968731135833808202 9
e2 = 10045126506586473838141905825133077895490946722550333732454328145195735376489979914521582319236923876049450391806874170260696555695944544086904458798494101185022794591894218061326541312872847190700371347525269238558212293976128684194168514565785053412372566093431876668490456782919358064418446864395913653385390295041780668238860517314667884744383738398034483804988003845978788149910086720544360935425135180129571068258422511559358553753530048988406634292745656220246732350810822223940151748310781902995241121125717188177122761188509812614895285400258103967866051974378426551806636116699187856351935526492629040644919
e = 65537

r = 7
a = (e2 - e1) * gmpy2.invert(e1*e2,N) % N
# assert a < N
P.<x> = PolynomialRing(Zmod(N))
f = x - a
x = f.small_roots(X = 2^1000,beta = 0.4)
x = x[0]
k_phi = e1*e2*x - (e2 - e1)
p_ = gcd(k_phi,N)

p = gmpy2.iroot(int(p_),r - 1)[0]
# print(p)
q = N // (p**r)
# print(q)
n = p * q
phi_n = (p - 1) * (q - 1)
d = gmpy2.invert(e,phi_n)
# print(n)
m = pow(c,d,n)
print(long_to_bytes(m))
msg = bytes.decode(long_to_bytes(m))
flag = hashlib.md5(msg.encode()).hexdigest()
print("d3ctf{" + flag + "}")
```

d3qcg

keywords: 二元coppermsith

Description

```
from Crypto.Util.number import *
import random
from random import randint
from gmpy2 import *
from secret import flag
import hashlib
assert b'd3ctf' in flag
Bits = 512
UnKnownBits = 146

class QCG():
    def __init__(self,bit_length):
        p = getPrime(bit_length)
        a = randint(0,p)
        c = randint(0,p)
        self._key = {'a':a,'c':c,'p':p}
        self.secret = randint(0,p)
        self.high = []
    def Qnext(self,num):
        return ((self._key['a'])*num**2+self._key['c'])%self._key['p']

    def hint(self):
        num = self.secret
        for i in range(2):
            num = self.Qnext (num)
            self.high.append(num>>UnKnownBits)
    def get_key(self):
        return self._key
    def get_hint(self):
        return self.high

Q1 = QCG(Bits)
print(Q1.get_key())
#{'a': 359151868029071994359613719079636629637448453638238006185223706464796944258139196781545754785896918719889
867011565111659872793974216575379880445835939710 1, 'c': 699682475294399463180251592112538252004491709517200922200
008137186174413557674474280679851039262117388263045674002431310102721980952053819505890388173958 33, 'p': 7386537
185240346459857715381835501419533088465984777861268951891482072249822526223542514664598394978163933836402581547 4
188219544070626403857564484084313 47}
Q1.hint()
print(Q1.get_hint())
#[675235839991023912866466486748270120898886505767153331474173629197063491373375704302862023618386823091427898 33
, 700071056797299678777916013607007326611244704739447926802538265697396193915724001484555276216763138017993184 22
]
enc = bytes_to_long(hashlib.sha512(b'%d'%(secret)).digest())^bytes_to_long(flag)
print(enc)
# 617661530281224716512583237899489083795270487484957178097139331850241718794508971891111637084033487357476204 54
299201502444138173893049692946240019455271 25
```

Analysis

`flag` 经过与 `secret` 异或，我们这里把 `secret` 称之为 `s0`；总而言之，我们需要知道 `s0` 的大小，而 `s0` 又通过了 `Qnext()` 函数，生成了 `s1`， `s2`

生成的过程如下

$$s \equiv$$

```
            self.high.append(num>>UnKnownBits)
```

假设已知的高位分别为`h1,h2`，低位数据（低128位）分别为`l1,l2`，就有

$$s \equiv a^2$$

> 对一个ρ阶的多项式f，可以：
>
> - 在模n意义下，快速求出n以内的根

大概测试一下`l1,l2`与`p`的相对大小

```
import gmpy2
p = 73865371852403464598577153818355014195330884659847778612689518914820722498225262235425146645983949781639338364025815474188219544070626403857564484084313 47
e = 2
assert 2 ** 128 < gmpy2.iroot(p,e)[0]
```

所以是可以进行`coppermsith`方法进行破解求得这个同余式的根`l1,l2`的

但是`sagemath`提供的`small_roots()`函数（专门用于求`coppermsith`的）只能求解一元多项式

这里显然是一个二元多项式，所以在`github`上找一找成品 GitHub - defund/coppersmith: Coppersmith's for generic multivariate polynomials

> 关于`coppermsith`的实现代码中`m,d`的取值要求
>
> - `m` - Determines how many higher powers of `f` and `N` to use. Defaults to 1.
> - `d` - Determines how many variable shifts to use. Defaults to `f.degree()`.

这里我们选择`m=4,d=4`（~~我的建议是取值可以调试，如果因为取值无法解出方程会报错的~~）

求解出来之后，那么`s1,s2`已知

$$s \equiv a^2$$

关于`Tonelli_Shanks`方法的相关推导可见 (11条消息) Tonelli-Shanks算法_python_M3ng@L的博客-CSDN博客

Solving code

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


a = 35915186802907199435961371907963662963744845363823800618522370646479694425813919678154575478589691871988986701156511165987279397421657537988044583593971 01
c = 69968247529439946318025159211253825200449170951720092200008137186174413557674474280679851039262117388263045674002431310102721980952053819505890388173958 33
p = 73865371852403464598577153818355014195330884659847778612689518914820722498225262235425146645983949781639338364025815474188219544070626403857564484084313 47

num1,num2 = 67523583999102391286646648674827012089888650576715333147417362919706349137337570430286202361838682 30
```

```
9142789833, 70007105679729967877791601360700732661124470473944792680253826569739619391572400148455527621676313801799318422
enc = 617661530281224716512583237899489083795270487484957178097139331850241718794508971891111637084033487357476204542992015024441381738930496929462400194552712 5
kbits = 146
s1_high,s2_high = num1 << 146,num2 << 146



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 []

P.<l1,l2> = PolynomialRing(Zmod(p))
f = a*(s1_high + l1)^2 + c - (s2_high + l2)
l1,l2 = small_roots(f,[2^146,2^146],m=4,d=4)[0]
s1 = s1_high + l1
d_a = gmpy2.invert(a,p)
```

```python
# s0 = gmpy2.iroot(int((s1 - c)*d_a),2)[0]

def Legendre(n,p):
    return pow(n,(p - 1) // 2,p)
def Tonelli_Shanks(n,p):
    assert Legendre(n,p) == 1
    if p % 4 == 3:
        return pow(n,(p + 1) // 4,p)
    q = p - 1
    s = 0
    while q % 2 == 0:
        q = q // 2
        s += 1
    for z in range(2,p):
        if Legendre(z,p) == p - 1:
            c = pow(z,q,p)
            break
    r = pow(n,(q + 1) // 2,p)
    t = pow(n,q,p)
    m = s
    if t % p == 1:
        return r
    else:
        i = 0
        while t % p != 1:
            temp = pow(t,2**(i+1),p)
            i += 1
            if temp % p == 1:
                b = pow(c,2**(m - i - 1),p)
                r = r * b % p
                c = b * b % p
                t = t * c % p
                m = i
                i = 0
        return r

s0 = Tonelli_Shanks(int((s1 - c)*d_a % p),p)
# flag = str_xor(enc,s0)
print(s0)
# s0 = 33453614052034629810418479143744538685991060606658122297844627347647422470489576550056124745875558397537486048827087416879261475364585674117891781293982 05
flag = enc ^^ bytes_to_long(hashlib.sha512(b'%d'%(s0)).digest())
print(flag)
print(long_to_bytes(flag))
```