【Neepuctf】Crypto部分writeup

原创

mortal15 于 2021-05-24 00:21:15 发布 126 收藏
分类专栏： Crypto

本文链接：https://blog.csdn.net/a5555678744/article/details/117203218


Crypto 专栏收录该内容
21 篇文章 2 订阅
订阅专栏

Crypto方向

1.RSA

```
from Crypto.Util.number import *
from sympy import nextprime
import gmpy2
import random

def encode (p1,p2,e):
    not_hint = (p1 + 1) * (p2 + 1)
    S = gmpy2.invert(e, not_hint)
    not_p = S%(p1+1)
    return not_p

flag = b'Neepu{********************}'
flag = bytes_to_long(flag)
p = getPrime(512)
q = getPrime(512)
n = p*q
e = nextprime(random.randint(1,1000))
d = gmpy2.invert(e, (p-1)*(q-1))
c = pow(flag, e, n)
print(c)
print(n)

m = encode(p, q, e)
c1 = pow(m, 7, n)
c2 = pow(m+e, 7, n)
print(c1)
print(c2)
```


先看一下代码，分析一下逻辑，要想得到flag，不仅要知道n的分解方式，还需要知道e是多少，那么把目标分解为两步，而这两者都和encode(p,q,e)有关。

分析到encode(p,q,e)里面，发现这个函数里面，S是e的模逆，(m*e-1)%(p+1)==0

那么就知道一定要先搞定m和e就可以搞定p了

求e的过程

```
c1 = pow(m, 7, n)
c2 = pow(m+e, 7, n)
```

这是明显的padding Oracle攻击可以用富兰克林算法搞定

注意以下脚本并非python脚本而是sagemath脚本，需要在sagemathNotebook中运行

算法代码如下：

```
 Franklin-Reiter attack against RSA.
# If two messages differ only by a known fixed difference between the two messages
# and are RSA encrypted under the same RSA modulus N
# then it is possible to recover both of them.

# Inputs are modulus, known difference, ciphertext 1, ciphertext2.
# Ciphertext 1 corresponds to smaller of the two plaintexts. (The one without the fixed difference added to
def franklinReiter(n,e,r,c1,c2):
    R.<X> = Zmod(n)[]
    f1 = X^e - c1
    f2 = (X + r)^e - c2
    # coefficient 0 = -m, which is what we wanted!
    return Integer(n-(compositeModulusGCD(f1,f2)).coefficients()[0])

  # GCD is not implemented for rings over composite modulus in Sage
  # so we do our own implementation. Its the exact same as standard GCD, but with
  # the polynomials monic representation
def compositeModulusGCD(a, b):
    if(b == 0):
        return a.monic()
    else:
        return compositeModulusGCD(b, a % b)

def CoppersmithShortPadAttack(e,n,C1,C2,eps=1/30):
    """
    Coppersmith's Shortpad attack!
    Figured out from: https://en.wikipedia.org/wiki/Coppersmith's_attack#Coppersmith.E2.80.99s_short-pad_at
    """
    import binascii
    P.<x,y> = PolynomialRing(ZZ)
    ZmodN = Zmod(n)
    g1 = x^e - C1
    g2 = (x+y)^e - C2
    res = g1.resultant(g2)
    P.<y> = PolynomialRing(ZmodN)
    # Convert Multivariate Polynomial Ring to Univariate Polynomial Ring
    rres = 0
    for i in range(len(res.coefficients())):
        rres += res.coefficients()[i]*(y^(res.exponents()[i][1]))

    diff = rres.small_roots(epsilon=eps)
    recoveredM1 = franklinReiter(n,e,diff[0],C1,C2)
    print(recoveredM1)
    print("Message is the following hex, but potentially missing some zeroes in the binary from the right e
    print(hex(recoveredM1))
```

```
    print("Message is one of:")
    for i in range(8):
        msg = hex(Integer(recoveredM1*pow(2,i)))
        if(len(msg)%2 == 1):
            msg = '0' + msg
        if(msg[:2]=='0x'):
            msg = msg[:2]
        print(binascii.unhexlify(msg))

def testCoppersmithShortPadAttack(eps=1/25):
    from Crypto.PublicKey import RSA
    import random
    import math
    import binascii
    M = "flag{This_Msg_Is_2_1337}"
    M = int(binascii.hexlify(M),16)
    e = 3
    nBitSize =  8192
    key = RSA.generate(nBitSize)
    #Give a bit of room, otherwhise the epsilon has to be tiny, and small roots will take forever
    m = int(math.floor(nBitSize/(e*e))) - 400
    assert (m < nBitSize - len(bin(M)[2:]))
    r1 = random.randint(1,pow(2,m))
    r2 = random.randint(r1,pow(2,m))
    M1 = pow(2,m)*M + r1
    M2 = pow(2,m)*M + r2
    C1 = Integer(pow(M1,e,key.n))
    C2 = Integer(pow(M2,e,key.n))
    CoppersmithShortPadAttack(e,key.n,C1,C2,eps)

def testFranklinReiter():
    p = random_prime(2^512)
    q = random_prime(2^512)
    n = p * q # 1024-bit modulus
    e = 11

    m = randint(0, n) # some message we want to recover
    r = randint(0, n) # random padding

    c1 = pow(m + 0, e, n)
    c2 = pow(m + r, e, n)
    print(m)
    recoveredM = franklinReiter(n,e,r,c1,c2)
    print(recoveredM)
    assert recoveredM==m
    print("They are equal!")
    return True
e=7
n=919952729271050811226591920110560204683055707485558496503099668872368713181568553186665404616696692478667
c1=101860667855118297591641948032098191722249661192276686384133501996626832851892860777365371612040191479l
c2=461821039942991455620228120234384957976860771044774726314941502220384044194141007276671712900986242141l3
CoppersmithShortPadAttack(e,n,c1,c2,1 / 80)#最后这个参数老是要调一调
```

在sagemathNotebook上跑出来结果如下：

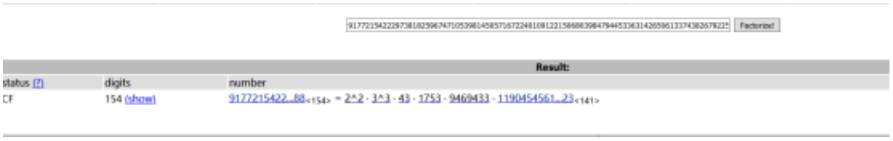
这里解出来的就是m了

得到m后求e就是在1到1000之间爆破了，脚本比较简单，就不贴在这里了。

爆破出来e=71

得到m和e之后m*e-1的值也就确定了，而这个数是(p+1)的整数倍，换言之，（p+1）必定是m*e-1的一个因数。

分解m*e-1如下：



写脚本得到m*e-1的因数表，其中减1能被n整除的就是那个p+1，很快就可以爆破出来

```
p=917721542229738182596747105398145057167224010912215068039847944533631426506133743826792253108992470761890
```

相应的q=n//p

```
q=100243122443861526304438928321116873147014244638154659769849204101122168716564932345074157785644256025701
```

整个解密脚本如下：

```
from Crypto.Util.number import *
from sympy import nextprime
import gmpy2
n=919952729271050811226591920110560204683055707485558496503099668872368713181568553186665404616696692478667
m=0x277cafa456beba0f916b725a92f28868138cb99a2d457a0d9ea6212bec789d6e87a1152278c5fd21df5f57b2a7c8247aee8709a
e=71
c=78543767285872349029076059073458316000847341792088805258173041942425687239313215276670106926320359779626

p=917721542229738182596747105398145057167224010912215068039847944533631426506133743826792253108992470761890
q=100243122443861526304438928321116873147014244638154659769849204101122168716564932345074157785644256025701
d = gmpy2.invert(e, (p-1)*(q-1))
print(long_to_bytes(gmpy2.powmod(c,d,n)))


#运行得到flag：Neepu{Have-a-g00d-day12138}
```

2.中国古代加密

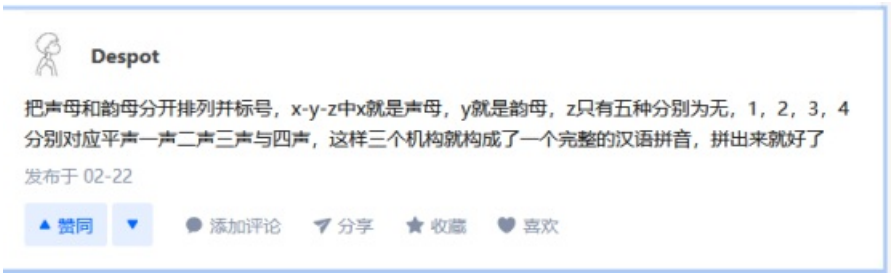
直到这个题出了第三个hint，这题才开始能做了起来。

现在拿不到原题了，只能文字描述解题过程

题目给的第一首词是乾隆年间写的一首词，这首词有它产生的背景，是祝贺一个140岁的古稀老人生日时写下的，前后两片都是指向一个数字--141，结合下面描述说让flag有了头尾，猜想flag的数字部分的头部和尾部应该都是141。

下面一首诗给了对应表之后，反应过来应该是反切码：

但是这还不够下面的描述中这首诗让flag有声有调，这其实只有声，搜了半天反切码，搜到一条回答让人顿时开悟



知乎上一位答主回答相关问题时提到了音调的识别问题

然后通过排列组合，有多个对应项的：令ch取8，an取10，u取12

就得到了最后的flag

```
Neepu{141181832310414124141}
```