MRCTF 2020 Crypto writeup

原创

Slightwindsec 于 2020-03-29 23:35:03 发布 2076 收藏 1

分类专栏： CTF 文章标签： 密码学 python

本文链接：https://blog.csdn.net/qq_41956187/article/details/105189556

文章首发于我的博客：https://am473ur.com/

MRCTF 2020 Crypto writeup

古典密码知多少

一张图，标准银河字母+圣堂武士+猪圈变形，在网上找密码表对照解出 FGCPFLIRTUASYON ,栅栏栏数3，FLAGISCRYPTOFUN。

flag：MRCTF{CRYPTOFUN}

天干地支+甲子

网上找天干地支对应的数字表，都加上 60 转 ascii 得到 Goodjob。

flag：MRCTF{Goodjob}

keyboard

对照九键键盘，重复次数就是某个按键的第几个字母，mobilephone。

flag：MRCTF{mobilephone}

vigenere

复制到在线网站就可以解了....比如这个 ，解开看最后一行。

flag：mrctf{vigenere_crypto_crack_man}

babyRSA

这题利用 P_p 和 P_factor 计算出 _P ，用 Q_1 ， Q_2 ， sub_Q 计算出 _Q ，然后解 RSA 就可以了。

P_p 往前找 9 个素数，约隔 1500 出现一个素数，递减枚举即可得到 P[0] ，得到 P 列表，然后就是解一个多素因子 n 的 RSA ，得到 _P .

_Q 就更简单了...把出题人故意写的 sub_Q ** Q_2 % Q_1 改成 pow(sub_Q,Q_2,Q_1) 就可以了。快速幂取模算法当然远比计算出结果再取模快得多。

exp：

```python
import sympy
import random
from gmpy2 import gcd, invert
from Crypto.Util.number import *
from z3 import *
base = 65537
Ciphertext =  17091872405163671414608621877494510476440948857917616735746743308408427921897950499683941222168544
91757922647656430908587059997070488674220330847871811836724541907666983042376216411561826640060734307013458794925
02568406280458943984302729028203499961791512423183852459360708037730098515217982819956947424167865155977176339559
59669714020607253768812979012647205398739153828000708220300634802912572965020766136237193619678956265845877831253
33505938858959644541233578654340925901963957980047639114170033936570060250438906130591377904182111622236567507022
71117645730147654346160052499304530072843281567207739987966827647183 2

#计算_P
P_p =20602792684730861271967757255499114342 1
P_factor =21367174276590898078711657997628960059586470457413446917311179096523362990951388470415844694640991047572
75843426418485978589422091511146273062863933902597002396988694874690808812671828030624880434691382527863818226
46126962323295676431679988602406971858136496624861228526070581338082202663895710929460596143281673761666804565161
43596395765501201105193618053658148849905951794630865013530042867248681964527996969351903940789294167278436286865
32436327279282796985881776941717972546448645541628486962107636811972797581308117237001546182807641233963123300
32986093579531909363210692564988076206283296967165522152288770019720928264542910922693728918198338839
t=0
while t<9:
      P_p-=2
      if isPrime(P_p):t+=1
#print(P_p)
#20602792684730861271967757255499114290 9

P = [0 for i in range(17)]
P[0] = 20602792684730861271967757255499114290 9
for i in range(1, 17):
      P[i] = sympy.nextprime(P[i-1])
phi_n=1
n=1
for i in range(17):
      phi_n *= P[i]-1
      n *= P[i]
p=pow(P_factor,invert(base,phi_n),n)
_P=sympy.nextprime(p)

#计算_Q
def gen_q():
      Q_1=10376643984946558808462504949579385763455651706456348843314822452463810597116105176312771843806286254818481474760129949405281366285145974012749955778539871448190946163199602004831579016796769993296797448448120987966417300958523146978514162898202184788394587120143015507125780316352361286311396749596957860552 1
      Q_2=15101073427691693979059146127898148644254803503235079730649610513635872358695312348408786017643862984368846267168177751365294755532560741485851456605351324308362781068608489026112064116198761443511488756549186612050784456621056162050396120585140938604119432672843707399537232243303515351975701739606306646974 3
      sub_Q=16899252979359331575789599510143024199495363833091931480013053680980182497111203957256238944958435064392439198480097819370779590995647299263100429047927352511695946185622726223260008917695081072947505826033217762696128600987663034094509362995930280318966890412389099106911382624149778366699575139136102894965 1
      Q = pow(sub_Q,Q_2,Q_1)
      return sympy.nextprime(Q)
_Q = gen_q()

#解RSA
_E = base
_M = pow(Ciphertext, invert(_E,(_P-1)*(_Q-1)), _P * _Q)
M=long_to_bytes(_M)
print(M)
```

```
print(M)
flag: MRCTF{sti11_@_b@by_qu3st10n}
```

Easy_RSA

这题还是计算出 _P 和 _Q 解 RSA 的题。

> 先是利用 n 和 phi_n，算出 p 和 q，n=p*q，phi_n=(p-1)*(q-1)=p*q-p-q-1，所以 p+q=n-phi_n-1，令 k=n-phi_n-1,把 p=k-q 带入到 n=p*q 中就可以作为二次函数利用单调性二分法求解了，时间复杂度 O(logn) 相当快，脚本在这里。这样 _P 就有了。
>
> 另一个问题是用给出的 n 和 e*d 分解出 p 和 q，这个算法可以直接用…
>
> ### Algorithm
>
> An RSA modulus N product of large distinct primes can be factored given (N, e, d) per:
>
> 1. Compute $f \leftarrow e\,d - 1$, and express f as $2^s\,t$ with t odd
> 2. Set $i \leftarrow s$ and $a \leftarrow 2$
> 3. Compute $b \leftarrow a^t \bmod N$, and if $b = 1$ then
>   - set a to the next prime, and proceed at 3
> 4. If $i \neq 1$ then
>   - compute $c \leftarrow b^2 \bmod N$, and if $c \neq 1$ then
>   - set $b \leftarrow c$, decrease i, and proceed at 4
> 5. If $b = N - 1$ then
>   - set a to the next prime, and proceed at 3
> 6. Compute and output $p \leftarrow \gcd(b - 1, N)$, and $q \leftarrow N/p$.
>
> For standard RSA where N has 2 distinct factors, we have fully factored N. Otherwise, p or/and q won't be prime, just re-run the algorithm from step 2 replacing N by any still unfactored component, until all the prime factors of N have been pulled out.
>
>

exp:

```
import base64
from random import randint
import sympy
from Crypto.Util.number import bytes_to_long, getPrime, getRandomNBitInteger,isPrime, long_to_bytes
from gmpy2 import gcd, invert

Ciphertext = 40855937355228438525361161524441274634175356845950884889338630813182607485910094677909779126550263304194796000904384775495000943424070396334435810126536165332565417336797036611773382728344687175253081047586602838685027428292621557914514629024324794275772522013126464926990620140406412999485728750385876868115091735425557755502739403341664303264477433964465401168671663976051235335571906579522220116721983178096130822578047848246729441082854348841225876444649481523876618572845441669189885946253208343721379310482375914731761363788141978758192074515143039452671279060844296010653753912188051426983069634173750771744894696202 1
_E = 65537
q = 11897508595485866064256258415213926142249334853259340030796012731724951176154203045191256136268736105319137530718041393172135525189535093637678165767489680138880637975075726437739660817423507502185461432800989740882423580016736920420368093829880375296498335829829969927342559638226886923713972475421444355638 3
p = 11815357834556225055076705773138578296306373458632111257986974765000144847363386030514228150486252192826465208763007074055151414447275508390668351959059272819038803078609426303224991061641917361742015064571572722208025156079396184767165938884288329623744941477235779809926616292547131169236900678271556688895 71
assert (p < q)
factor2 = 2021 * p + 2020 * q
_P = sympy.nextprime(factor2)
```

```
Q_n = 20714298338160449749545360743688018842877274054540852096459485283936802341271363766157976112525034004319938054034934880860956966585051684483662535780621673316774842614701726445870630109196016676725183412879870463432277629916669130494040403733295593655306104176367902352484367520262917943100467697540593925707162162616635533550262718808746254599456286578409187895171015796991910123804529825519519278388910483133813330902530160448972926096083990208243274548561238253002789474920730760001104048093295680593033327818821255300893423412192265814418546134015557579236219461780344691279876695651389303085251899448974217539 47
Q_E_D = 10077207922229813458611615685074281785540812771696289192925986874667257260233391895807558267175249361825951828633612277270333018303722110505829865349079433788509849907358382183253279830951353838317523342953346734839038932322519880529495048480206814859090290722115096853906798043283131037636820277321226632011267069973750105483164628658514228141923757222271397564684355502473185568857383410871187440614954007825377434970815806305575493281267578612370076828804844532619988098371750453882549810378930487368219105305036680682580260265867426844084457795549936840401911491393447716042842866284701228951665531068011963860031522828429893520 1
f, s, tem = Q_E_D-1, 0, 1
while f % 2 == 0:
    f = f // 2
    s += 1
i, a, t = s, 2, f
b = pow(a, t, Q_n)
while b == 1:
    a = sympy.nextprime(a)
    b = pow(a, t, Q_n)

while i != 1:
    c = pow(b, 2, Q_n)
    if c != 1:
        b = c
        i -= 1
    else:
        break
if b == Q_n-1:
    a = sympy.nextprime(a)
    b = pow(a, t, Q_n)
    while b == 1:
        a = sympy.nextprime(a)
        b = pow(a, t, Q_n)

p = gcd(b-1, Q_n)
q = Q_n//p
factor2 = 2021 * p - 2020 * q
if factor2 < 0:
    factor2 = (-1) * factor2
_Q = sympy.nextprime(factor2)

_M = pow(Ciphertext, invert(_E, (_P-1)*(_Q-1)), _Q*_P)
m = long_to_bytes(_M)
print(m)
```

flag：MRCTF{Ju3t_@_31mp13_que3t10n}

real_random

感觉这题主要代码就是下面截取的部分，对 x 进行若干次 (b * x + c) % m 运算，最后会得到 gen_N(flag[t])，所以只要知道第几个 i 是满足random[t][i] = gen_N(flag[t]) 就可以了，尝试了几次发现得到的 i 是和 c 是无关的，当然也是和 gen_N(flag[t]) 无关。

```
x = ((gen_N(flag[t]) - c) * _b) % m
for i in range(2**d):
    x = (b * x + c) % m
for i in range(times):
    x = (b * x + c) % m
    random[t][i] = x
```

既然 i 的值仅与 p , q , d 等变量有关，那么就可以自定义一个数 K 来代替 flag[t]，本地利用给出的 p , q , d 算出其他变量(p q 很容易用 M 枚举出来)，一模一样的跑一遍和服务器上一样的程序，本地得到的 i 就一定是服务器上的 i，send 上去就可以了。

得到的是 (flag * 16^2) + flag 的结果，对 16^2 取模就是 flag 。

exp：

```python
from pwn import *
from Crypto.Util.number import*
from gmpy2 import*
import re

host, port = ('38.39.244.2', 28101)
r = remote(host, port)

null = r.recvline().decode()
null = r.recvline().decode()

def gen_N(flag):
    return (flag * 16**2) + flag

def reM(M):
    p = getPrime(6)
    q = getPrime(6)
    while (p-1)*(q-1) != M:
        p = getPrime(6)
        q = getPrime(6)
    m = p * q * 2 ** 5
    b = 4 * p * q + 1
    return (b, m)

def Slove(M, d):
    b, m = reM(M)
    c = getPrime(10)
    _b = invert(b, m)
    times = getPrime(19)
    x = ((gen_N(102) - c) * _b) % m
    for i in range(2**d):
        x = (b * x + c) % m
    for i in range(times):
        x = (b * x + c) % m
        if x == gen_N(102):
            return i
            break

ans = b''
while True:
    r. recvuntil('m:  ')
    M = r.recv(4).decode()
    r. recvuntil('d:  ')
    d = r.recv(2).decode()
    M = int(M)
    d = int(d)
    r. recvuntil('Now guess where the flag is ^_^ ')
    I = Slove(M, d)
    print('I=', str(I))
    r. send(str(I)+'\n')
    r. recvuntil('\n')
    k = int(r. recvline().strip(). decode())
    k %= 16**2
    ans += long_to_bytes(k)
    print(ans)
```

flag：MRCTF{lt_13_n0t_r3@11y_r@nd0m}