2021宁波市第四届网络安全大赛（练习平台）RSA部分

原创

Umbrella_伞 于 2021-05-22 17:43:49 发布 221 收藏

分类专栏： CTF 文章标签： 安全 unctf rsa 密码学

本文链接：https://blog.csdn.net/qq_46075726/article/details/117167226

挺久没有做过CTF了，明天就要打比赛了，临时抱佛脚，撸两道rsa，找找手感。

公钥文件泄露

题目给了两个文件：pub.key和flag.enc:

pub.key

```
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhAMAzLFxkrkcYL2wch21CM2kQVFpY9+7+
/AvKr1rzQczdAgMBAAE=
-----END PUBLIC KEY-----
```

flag.enc（乱码）

```
ZF:9Mw 骓骼)P恂黩?沏悴[嘟??
```

解题思路挺多的，这个没啥好说的（虽然因为很久没有做过，卡了很久）

> 只想简单解题的话，直接找个在线解密的解密的网站（我就不提供了，自己找吧），然后把这两个文件放上去解密就行了。
>
> 稍微复杂点的化，就是用openssl可以查看到公钥(n，e），大数分解n，得到p，q，然后写个python脚本

```
openssl rsa -pubin -in pub.key -text
```

```
hGHRqgwYWUhUFTozP4q94Ut7uDfLVS0IO0qwdy4mpY6LS4+lLIQtVDUrbmKuF8+0
5vP5MG/qUqv8g574pi7n9yu+YeDGEAkLnDLRpGFUiwb1P1YsfFnJv460emT6bcZO
VhF8j/TTdOeEUTHpFQIDAQAB
-----END PUBLIC KEY-----
```


写个脚本，将上面的modulus里的的字符做个分割，组成16进制数。

```
from rsa import PublicKey, transform, core, common
import rsa
import gmpy2

"""第一种方法"""

temp = []
temp = hex("00:aa:08".split(':'))
n = ''
for i in temp:
    n += i
n = eval(n)
print(n)
# 将n进行大数分解，得到p，q
p = 285960468890451637935629440372639283459
q = 304008741604601924494328155975272418463
e = 65537
phi = (p-1)*(q-1) # 欧拉函数
d = gmpy2.invert(e,phi) #计算模逆元d
key = rsa.PrivateKey(n,e,int(d),p,q) # 有了n，e，d，p，q之后 使用rsa模块生成私钥
with open('./flag.enc','rb+') as f:
    f = f.read()
    print(rsa.decrypt(f,key)) # 解密文件

""" 第二中方法:直接使用rsa模块去查看n，e """
whit open("./pub.key","rb+") as k, open('./flag.enc','rb+') as f:
    k = k.read()
    public_key = PublicKey.load_pkcs1_openssl_pem(k) #通过rsa模块的内容去读取公钥的信息
    print(public_key) # 这里会输出一个 n 和 e 的元组
    # 将n进行大数分解，得到p，q
    p = 285960468890451637935629440372639283459
    q = 304008741604601924494328155975272418463
    e = 65537
 key = rsa.PrivateKey(n, e, int(d), p, q)

    f = f.read()
    print(rsa.decrypt(f,key))
"""
# 在使用第二种方法的是后，其实是有个坑的。因为我自己很久没有做rsa了。
# 很多东西都要重新开始学，这个脚本最初也是照着网上的脚本写的（网上的脚本贴贴在下面）
# 他用的是core.decrypt_int()方法去解密的。这没有什么问题，只是需要做个数据格式转换。
# 但是 PublicKey.load_pkcs1(PUBLIC_KEY) 这个地方就有坑了。
# rsa使用的密钥文件有两种格式：
# 第一种：
 -----BEGIN PUBLIC KEY-----
 .........
 -----END PUBLIC KEY-----
# 第二中
 -----BEGIN RSA PUBLIC KEY-----
 .........
 -----END RSA PUBLIC KEY-----
# PublicKey.load_pkcs1()方法都的是第二种密钥格式
# PublicKey.load_pkcs1_openssl_pem() 方法的的是第一种密钥格式
```

```
# 而且：
#
# 两种方法返回的数据是不同的！
# 两种方法返回的数据是不同的！
# 两种方法返回的数据是不同的！
#
# 重要的事情说三遍（你细品我现在的心情）
#
"""
```

脚本在这里摘自 作者：窗户

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
import sys
#rsa
from rsa import PublicKey, common, transform, core
def f(cipher, PUBLIC_KEY):
 public_key = PublicKey.load_pkcs1(PUBLIC_KEY)
 encrypted = transform.bytes2int(cipher)
 decrypted = core.decrypt_int(encrypted, public_key.e, public_key.n)
  text = transform.int2bytes(decrypted)
  if len(text) > 0 and text[0] == '\x01':
    pos = text.find('\x00')
    if pos > 0:
     return text[pos+1:]
    else:
     return None
fn = sys.stdin.readline()[:-1]
public_key = sys.stdin.readline()[:-1]
x = f(open(fn).read(), open(public_key).read())
print x
```

Baby_RSA(我也不知道应该算什么分类)

题目直接给了一个加密脚本。或不多说，先上脚本：（原脚本里面没有任何注释，大部分的东西都在注释里了）

```
## 看着上面一堆的导入库，然后用到的却不多，头秃

import sympy # 这个是解题的核心库，pip装一下就好了
import random
from gmpy2 import gcd, invert # gmpy2，没啥好说的，rsa加解密的核心库
from Crypto.Util.number import getPrime, isPrime, getRandomNBitInteger, bytes_to_long, long_to_bytes
# Crypto 这个库有点恶心人。网上大部分教程都是直接 pip install pycryptodemo 完事了
# 然后你会发现，屁，照样报错，解决办法是：
# 你去python库安装的目录的Python38\Lib\site-packages
# （找不到的话CMD，where python，就能去到python的根目录里）
# 把 crypto 目录改成 Crypto 就能用了
# 原因我也不知道，还没有去查过库
from z3 import *



flag = b"nsfocus{xxxx}"  # 这个flag的样式是很重要（记号1）
base = 65537

# 这个函数全程没有用到过，我不知道他是干嘛用的，迷惑我们的？迷茫
def GCD(A):
```

```python
    B = 1
    for i in range(1, len(A)):
        B = gcd(A[i-1], A[i])
    return B

# 取 p 的函数，挺有意思的
def gen_p():
    P = [0 for i in range(16)]
    # 获取了一个随机的质数赋给 P[0] ,而后的每个个值都是前一个质数的下一个质素
    P[0] = getPrime(128)
    for i in range(1, 17):    # 做题的的时候眼抽了，没看到这里重新更新了P[]的大小
        P[i] = sympy.nextprime(P[i-1]) # 还很蒙蔽的说下面循环给17不是超界了

        # sympy.nextprime() 获取参数的下一个质数
        # 我很自然的想到了 获取前一个质数方法，然后发现真的有，这是这个题目的解题核心

    print("P_p :", P[9]) ##### 注意 这个P[9],这样我们就能把整个P[]都给不全了

    n = 1
    for i in range(17):
        n *= P[i]
    p = getPrime(1024)
    factor = pow(p, base, n)
    # 看到这里，有没有发现点什么？
    # 没错，就是在这个地方做了 rsa加密，加密了p。一个简单的欧拉函数(phi)变形的加密
    # 数学解题思路在下面
    print("P_factor :", factor)
        ##### 注意 这里返回的是 p 的下一个质数
    return sympy.nextprime(p) ##### 注意 这里返回的是 p 的下一个质数
        ##### 注意 这里返回的是 p 的下一个质数

           ##### 你没想错，我又踩坑了
# 取 Q 的函数，这个就真的有意思了。
def gen_q():
    sub_Q = getPrime(1024)
    Q_1 = getPrime(1024)
    Q_2 = getPrime(1024)

    # 上面去了三个质数，没啥好说的，下面的这个表达式

    #做个记号，后面要用（记号2）
    Q = sub_Q ** Q_2 % Q_1  ##### 没错就是这里,整个题目最精妙的地方

    # 下面的这些都没啥好说的
    print("Q_1: ", Q_1)
    print("Q_2: ", Q_2)
    print("sub_Q: ", sub_Q)

    return sympy.nextprime(Q)  # 这个地方返回的也是 Q 的下一个 质数

if __name__ == "__main__":
    _E = base
    _P = gen_p()
    _Q = gen_q()
    # 这个地方是用来保证 _E 和 phi（欧拉函数）互质用的（方便求d）
    assert (gcd(E, (_P - 1) * (_Q - 1)) == 1)

    M = bytes_to_long(flag)  # 做了个数据类型转换，方便加密

    # 最终的加密
```

```
    _C = pow(M, _E, P * Q)    ### 这个地方小坑， P 是 _P, Q 是 _Q
            ### 我当时傻乎乎以为是上面的 return 里面的p和q
    print("Ciphertext = ", _C)


# 下面是一大串数据，看看就好了
'''
P_p : 206027926847308612719677572554991143421
P_factor : 213671742765908980787116579976289600595864704574134469173111790965233629909513884704158446946409910475727584342641848597858942209151114627306286393390259700239698869487469080881267182803062488043469138252786381822646126962323295676431679988602406971858136496624861228526070581338082202663895710929460596143281673761666804565161435963957655012011051936180536581488499059517946308650135300428672486819645279969693519039407892941672784362868653243632727928279698588177694171797254644864554162848696210763681197279758130811723700154618280764123396312330032986093579531909363210692564988076206283296967165522152288770019720928264542910922693728918198338839
Q_1:  103766439849465588084625049495793857634556517064563488433148224524638105971161051763127718438062862548184814747601299494052813662851459740127499557785398714481909461631996020048315790167967699932967974484481209879664173009585231469785141628982021847883945871201430155071257803163523612863113967495969578605521
Q_2:  151010734276916939790591461278981486442548035032350797306496105136358723586953123484087860176438629843688462671681777513652947555325607414858514566053513243083627810686084890261120641161987614435114887565491866120507844566210561620503961205851409386041194326728437073995372322433035153519757017396063066469743
sub_Q:  168992529793593315757895995101430241994953638330919314800130536809801824971112039572562389449584350643924391984800978193707795909956472992631004290479273525116959461856227262232600089176950810729475058260332177626961286009876630340945093629959302803189668904123890991069113826241497783666995751391361028949651
Ciphertext =  345623589099899619561032857607081860821214611511405481456027050930754187121990192431790628578334550245040790329751574990992524705765647909915632671853514996929804468773071797890974260467518835020187979846056972195540296609855691105457928921653324488355642760778712906352434147867527223651542479559424654280980312382566018143515151700992698915229123958004645008359970690898294185984012966296090785469067558255181604932863141234420395671652662097558543008112248486048018657161884184536763767013699669161321608151292096720066891837553781297306414109593351344034238966188378088890844885124575865978213408940779494887391 0
'''
```

ok，看到这里，基本上代码都读懂了，那我们就开始填坑了。

第一个坑 —— get_p() 函数：

```
def gen_p():
    P = [0 for i in range(16)]
    P[0] = getPrime(128)
    for i in range(1, 17):
        P[i] = sympy.nextprime(P[i-1])
    n = 1
    print("P_p :", P[9])
    for i in range(17):
        n *= P[i]
    p = getPrime(1024)
    factor = pow(p, base, n)
    print("P_factor :", factor)
    return sympy.nextprime(p)
```

上面也说了这是一个简单的欧拉函数的加密。

首先，已知 P[9],也就是P_p，我们就能给整个P[]数组给初始化。所以 n 的值也就能够计算出来了。

```
import sympy
import gmpy2
P = []
P[9] = P_p
for i in range(8,-1,-1):
 P[i] = sympy.prevprime(P[i+1])
for i in range(10,17):
    P[i] = sympy.nextprime(P[i-1])
n = 1
for i in range(1,17):
    n *= P[i]
```

接下来就到了欧拉函数部分了，已知n，e，要求d，最简单的方法就是通过欧拉函数。那么，欧拉函数是多少？

提醒一下，碰到简单n，e题目的时候，我们通过将将n分解曾p，q两个质数，然后用 phi = （p-1)*(q-1) 来计算。那这个地方呢？也去分解n?

其实，出题的时候就已经给你分好了。P[]数组里面的值都是质数！

而 n = P[1] * p[2] * …*P[16]

所以 phi(n) = phi(P[1]) * phi(P[2]) * … * phi(P[16])

所以 phi(n) = (P[1] - 1)*(P[2] -1)*(P[3] - 1)* … * (P[16] - 1)

所以 欧拉函数也有了，p，不久放出来了吗！（咳，有点味道）

```
phi = 1
for i in range(1,17):
 phi *= (P[i] - 1)
d = gmpy2.invert(base, phi)
p = pow(factor, d, n)
return sympy.nextprime(p)
```

OK,接下来第二个坑（天坑）

get_q()函数

```
def gen_q():
    sub_Q = getPrime(1024)
    Q_1 = getPrime(1024)
    Q_2 = getPrime(1024)
    Q = sub_Q ** Q_2 % Q_1
    print("Q_1: ", Q_1)
    print("Q_2: ", Q_2)
    print("sub_Q: ", sub_Q)
    return sympy.nextprime(Q)
```

看到上面的算法，你第一件想到的事情是什么？我想到的是sub_Q,Q_1,Q_2的结果都给了，跑不久完事了。如果你也是这样想的，那恭喜你，和我一样掉坑了，等到宇宙爆炸的都不一定能拿到结果。

让我们来看看（记号2），这里的运算时乘方，不是乘法，注意 是乘方，乘方，乘方。

一个1024位的数字做1024为数值次的乘方。怎么算？拿头算？

而这个地方精妙的就是后面的取模了。这就在告诉我们，Q你是拿不到了，但Q肯定是 Q_1 > Q > 0 之间的质数。

没了。

所以，到了这里，就是在变相的告诉你，最后的加密用到的q。而且q要么很小，靠近0，要么很大，靠近Q_1。

那思路不久清晰了吗：

p 已知了，Q_1已知，q自己去爆破 (用sympy.nextprime() 方法和sympy.prevprime()方法) ,e 已知了。

所以代码可以撸起来了。

```
import gmpy2
import sympy
from Crypto.Util.number import long_to_bytes
p = return sympy.nextprime(p)
q = Q_1  # 或者 q = 1
c = Ciphertext
Q_2 = 1
while True:
 n = p * q
 phi = (p-1) * (q-1)
 d = gmpy2.invert(e, phi)
 m = pow(c, d, n)
 m = long_to_bytes(m).decode('utf-8','ignore')
 print(m)
 q = sympy.prevprime(q)  # q = sympy.nextprime(q) 和前面的q=1配套使用
```

这样就可以爆破出结果了，不过还有一个问题，就是它没有终止条件。所以，这个时候就要用到（记号1）的flag格式了

```
# 只需要在 m = long_to_bytes(m).decode('utf-8','ignore')之后的任意位置加一个if判断就行了
if "nsfocus" in m:
 break
```

为了高效的爆出结果，我们肯定是选在两头都跑（1和Q_1）,所以可以加个多线程上去。不过我比较菜，不会多线程，所以我选择开两个文件，嘿嘿，技术不够，脚本来凑。

完整脚本：

```
import sympy
import gmpy2
from Crypto.Util.number import long_to_bytes

e = 65537
P = [0 for i in range(17)]
P_p = 20602792684730861271967757255499114342 1
P[9] = P_p
P_factor = 213671742765908980787116579976289600595864704574134469173111790965233629909513884704158446946409910475727584342641848597858942209151114627306286393390259700239698869487469080881267182803062488043469138252786381822646126962323295676431679988602406971858136496624861228526070581338082202663895710929460596143281673761666804565161435963957655012011051936180536581488499059517946308650135300428672486819645279969693519039407892941672784362868653243632727928279698588177694171797254644864554162848696210763681197279758130811723700154618280764123396312330032986093579531909363210692564988076206283296967165522152288770019720928264542910922693728918198338839

n = P[9]
phi = P[9] - 1
for i in range(8,0,-1):
    P[i] = sympy.prevprime(P[i+1])
    n *= P[i]
    phi *= (P[i] - 1)

for i in range(10,17,1):
    P[i] = sympy.nextprime(P[i-1])
    n *= P[i]
    phi *= (P[i] - 1)
```

```
d = gmpy2.invert(e,phi)



final_p = pow(P_factor, d, n)
final_p = sympy.nextprime(final_p)

Q_1 = 10376643984946558808462504949579385763455651706456348843314822452463810597116105176312771843806286254818481474760129949405281366285145974012749955778539871448190946163199602004831579016796769993296797448448120987966417300958523146978514162898202184788394587120143015507125780316352361286311396749596957860552 1

final_q = 1  #final_q = Q_1

Ciphertext =  345623589099899619561032857607081860821214611511405481456027050930754187121990192431790628578334550245040790329751574990992524705765647909915632671853514996929804468773071797890974260467518835020187979846056972195540296609855691105457928921653324488355642760778712906352434147867527223651542479559424654280980312382566018143515151700992698915229123958004645008359970690898294185984012966296090785469067558255181604932863141234420395671652662097558543008112248486048018657161884184536763767013699669161321608151292096720066891837553781297306414109593351344034238966188378088890844885124575865978213408940779494887391 0

while True:
    n = final_p * final_q
    phi = (final_p - 1) * (final_q - 1)
    try:
        d = gmpy2.invert(e,phi)
        m = pow(Ciphertext, d, n)
        m = long_to_bytes(m).decode('utf-8',"ignore")
        if "nsfocus" in m:
            print("m: ",m)
            break
    except:
        pass

    final_q = sympy.nextprime(final_q) # final_q = sympy.prevprime(final_q)
```

其实，这题目就真的是在考数学。一手拓展欧几里得算法求逆元，然后同模余的运算法则。会了这一题，应该算是rsa算法有了初步的认知了。

先上解题代码：

```
import gmpy2
from gmpy2 import gcd
from Crypto.Util.number import long_to_bytes
n =  17083941230213489700426636484487738282426471494607098847295335339638177583685457921198569105417734668692072
72775913935820766724870395243668018315332760614742193236588998334728204643915617668576514362063710734787040194694650162053166557366806834908041080799658229750588994620505287900202893612531531225647058362291364631977912555969127091606458868499738245141274743272296691951341370998735303837547717838512545356711196525972148499715679935561764213156909581030407713105358848305724434074275180493549408768736341692131404154709311856576760966703385958312527532207761757678324785371851616674385826529113535389523998112 1

gift =  21354926537766862125533295605609672853033089368258873559119169174547721979606822401498211381772168335865090909698924197759584060879940545850228941659507684277415457362479184102558048945220857206429525796384184838002433683127025664581967085085436350513509995727871882362432756316098752536170156644140320588229194694432844534030640762327650242484355433265974188517515863085145401245713091527875597129502093578255768961322780451121779102660197410139951065794848687682510844533384171154835151328695947121620523620834141639546813062591370575810366574418974284325759240189509611418225542513692622483688999773378861901141 04
c =  11896048663970780362228781837489466486466282266282525774268696785272105376496162776613405022882264737932390195829622289249358353406571295574018064222013763168773667801755802797093469517945242168621127603773017861402501708822613876791515803039151959944370904767218422806768104905816096919472743978664055791890321098068209891550854674116923626647321385265228978238829509815935768090750750110601100943820432504952197382099637084090991156423394140353352175855468352060270942119322431162691631220150197779269032493785754953460073099974345959080880071039856065496031922388307631138393107747419702153992404456267795192911174

e_orign = 54722
e = 27361

for k in range(1,8):
    d = gmpy2.invert(e,k*gift)
    m = gmpy2.powmod( c, d, n)
    m = gmpy2.iroot(m,2)

    m = long_to_bytes(m[0]).decode('utf-8','ignore')

    print('m : ',m)
```

先自己试着去理解一以下把，解析后面再补。我困了。该睡觉了。