2021年9月绿城杯，CRYPTO的RSA-2

原创

沐一 · 林 于 2021-10-07 22:07:58 发布 84 收藏

分类专栏： 密码学 CTF 文章标签： ctf

本文链接：https://blog.csdn.net/xiao__1bai/article/details/120640778

密码学 同时被 2 个专栏收录

51 篇文章 1 订阅

订阅专栏

CTF

167 篇文章 6 订阅

订阅专栏

2021年9月绿城杯，CRYPTO的RSA-2

.

.

打开文件，发现是分成两段的`RSA`加密，第一段加密`flag[:20]`，第二段加密`flag[20:]`。

```
from Crypto.Util.number import *
import gmpy2
from flag import flag
assert flag[:5]==b'flag{'

m1 = bytes_to_long(flag[:20])
p  = getPrime(512)
p1 = gmpy2.next_prime(p)
q  = getPrime(512)
q1 = gmpy2.next_prime(q)
n1 = p*q*p1*q1
print('n1 =',n1)
e = 0x10001
c1 = pow(m1,e,n1)
print('c1 =',c1)



m2 = bytes_to_long(flag[20:])
p2 = getPrime(1024)
q2 = getPrime(1024)
print('p2+q2 =',p2+q2)
```

```
print('q2*q2 =',p2*q2)
n2 = p2*p2*q2*q2*q2
print('n2 =',n2)
c2 = pow(m2,e,n2)
print('c2 =',c2)

#n1 = 6348779979606280884589422188738902470575876294634928314659473603635680262809639892915911577103896292161096152747547183299879905518361156608791032341299219438240614163962643581102160479943311199205034314915095296047424680329069509842569645604050623452801205267714399402786062261530779590578822627452733949866070044067700354593016958063785988905894325389162198214777770214601891400815217791032269535444264418232447658283429730864229490179377012613489635410351286614640687690337723903204267950446177519097879141859859112776284046325335303907612572515520734936975185473502469936798441322974140947271471611695481601586911
#c1 = 6201882078995455673376327652982610102807874783073703018551044780440620679217833227711395689114659144506630609087600915116940111002026241056808189658969089532597757995423694966667948250438579639890580690392400661711864264184444018345499567505424672090632235109624193289954785503512742400960515331371813467034511130432319427185134018830006918682733848618201088649690422818940385123599468595766345668931882249779415788129316594083269412221804774856038796248038700275509397599351533280014908894068141056694660319816046357462684688942519849441237878018480036145051967731081582598773076490918572392784684372694103015244826

#p2+q2 = 2747731467611384627081375823090973864377938917936913830338565243030108112941019334548244850105214689148461518198760435085418796375444442565207414184954793937771328309858565220085610884108628159132922886837616579191219300169569168658492611537210976713158834693489729257570780897151020322418185269259886455786778
#q2*q2 = 1851472427003096217256696594172322438637407629423265225870108578101877617284335592056603515733157952498010819073914195992652308214227367274184955247515627839713157136009901859201895978562778513012647798276521049854768036723072363442403600953934785434457353784862806146889216619986622798416784313979342968255924131707297937400291260754903943139826718481877150346811637961824931932478899632134076462459344310635410427447260117022983521963809324255754784006089252757694007716299006968701996694682621011231840826974929436658668273261437243421876872057791736872653020089755891268747008858377471176759958003766337892900021 7
#n2 = 4058822704559530408036038504108223850704429273134446581529603290563352555694378761071265167546081076876276349357912983127101814159154620755741081743245513931552767493293308529927759917397191244522653223581458087958531721134952440642420062267588099239078202515862124149969340028803165819443464171802691065232793325387731310611286128331427463512473481739846505937356219469495784126483431264092627889038608961110371499064654147057735159952690445834266044496859119760682036136476164820524104144468114582079905441317946228550966112436207409358349493270624946195424040882708701552550717308212941223448622809200284186836589583746369920095991578276765725872979403777640199530924494117141584240361748671949248367149083456257922550683149688154253051959543893248279686785323415966440942097752610248038519310188378516108026957370715662683855150602445548065022430589450196858344234680712692074077978059365087164591514968942429291261157829191272189686477295041026662904554248000926657409608013870968346648956829056936347844434956349850753080550251105116516082719279552018272080242221336424735577522285821464860303474367918747084421252913437497573751098228795731687817996460239474960143182316798215743489045924539437072894279011715648526811675805263679441726868090142019300228903553875362055548850692636662464129188135326861713096899125898300216530018697196366166647660099838904888056519931728042834980282444832989850278849223338187302621720298192165467384014209583960336066604947610056126833622590250493280060546413619227559388673674649795527028054142359 3
#c2 = 2559109016854482176174602417872466083959094819045132922748116857649071724229452073986560206108255875975119645211772064742659826156857244094237003970293282194136679214017342848834493220357633429264825555117127482882165709766710679287220008257931996331050372143550062314601295447461315084808342512698755459465179747774182865523824355026697221675259378873483637314436321763961249239722880821520586228127877409631761591885440399262072096917378815121548990881274917986180314493716958745200809700894071009136118394226824527115446187210281360275443993974756650711651936282125572417909305104199473085640149399677127617234331304575591675108269314988592210549181822501284451926493313762292902491861947753852153354855178973969893306721230557848041616360913718989179720927755741116964356854039230303671995214055443533885167144095286515107738322030529500163281644214402243776308913314188692426577424729030666982508586235173233639561727610037423715958075999959302875693935484067733346728163243576703315005243926250105929903521292804154625993311856425111958897000901687385547855658825013896993859998819849456724117239945374170984048695318976428911831287058099311563671072413980970825636021272812778639441167642782843156904627968748136821513756150077748038050155161657783249952129565523736018415988915183776635311618532031777464529420104477282809907491707789663190965467161255720765383034489764411593632212835149455100465298155075879128543480981687238190040144074357810458230521548888856316605456880214592139972667375272282064680749465729910419012394567564 7
```

.

.

（这里积累第一个经验）

先看第一段，`gmpy2.next_prime(p)`函数的意思是取邻近p的下一个素数，那么很明显`p和p1`，`q和q1`都是两两相邻的素数，相差很小，`n1`是`2048bit`级别的，没法硬钢。

后半段的`n2`和前半段的加密逻辑差别很大，所以不是两个n的`模不互素`。也不是像上一道题RSA1一样的`密文与模数`不互素。

```
m1 = bytes_to_long(flag[:20])
p  = getPrime(512)
p1 = gmpy2.next_prime(p)
q  = getPrime(512)
q1 = gmpy2.next_prime(q)
n1 = p*q*p1*q1
print('n1 =',n1)
e = 0x10001
c1 = pow(m1,e,n1)
print('c1 =',c1)
```


.

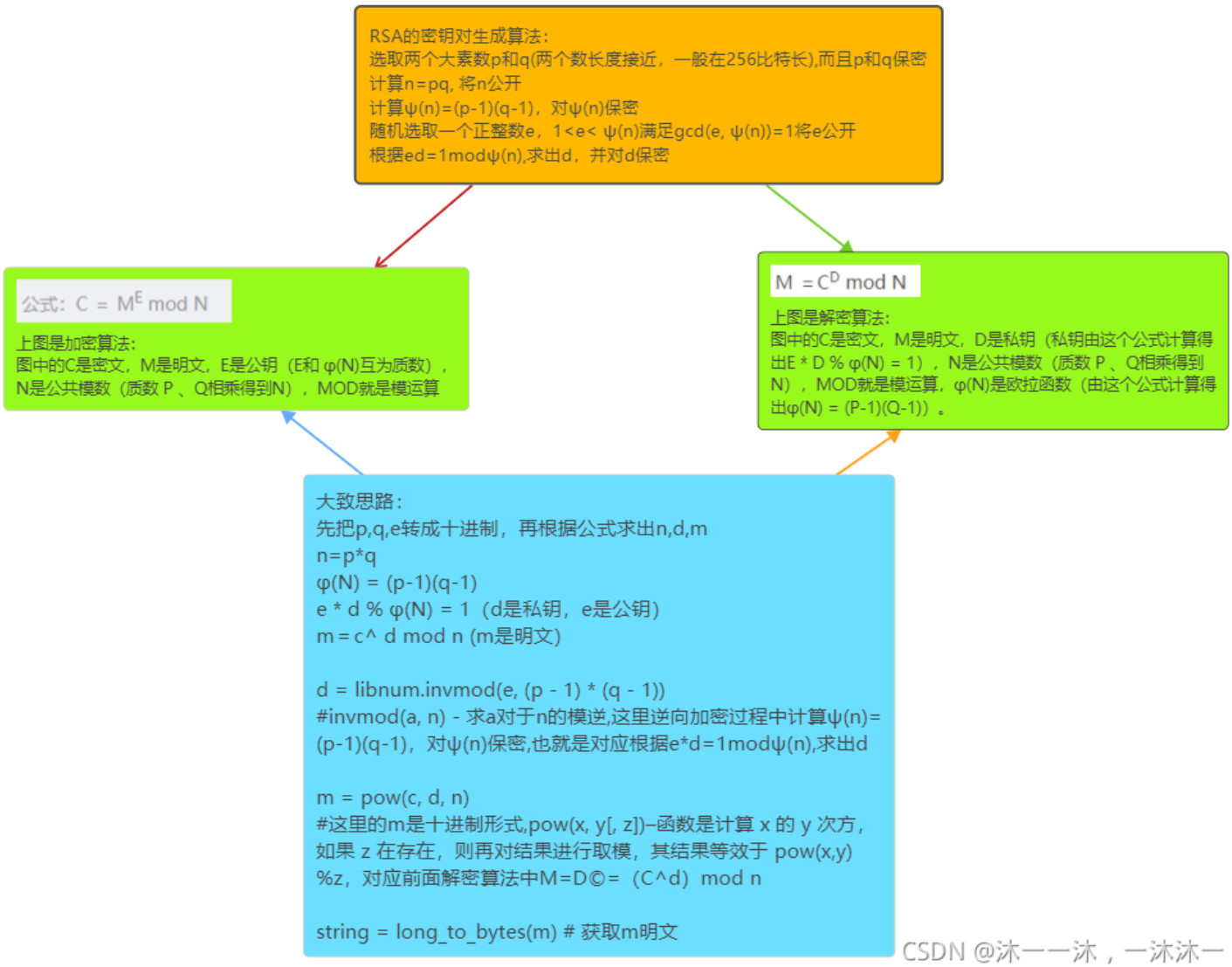
.

（这里积累第二个经验）

然后查了资料后发现关键在`p和p1`，`q和q1`都是两两相邻的素数这里，也就是说`p,q,p1,q1两两互素`。单纯的`factordb.com`和`yafu`分解n肯定是分解不出来的，毕竟是`2048bit`嘛。所以我们需要一些算法，比如前面`模不互素`的`欧几里得算法`这样，这里我们要用的是`费马分解`算法。

费马分解算法的特征就是`n`是`4`个数的乘积，分解n之后我们会得到`p、p1、q、q1`四组隔开的排列组合，但是我们的脚本可以把组合限定成`p * q1，  p1 * q`,和`p * q ，p1 * q`这样。然后通过`gcd(pq1,pq) = p`、`gcd(p1q,p1q) = q`求出两组各一个数，然后就可以求出`ϕ(n)=ϕ(p)·ϕ(p1)·ϕ(q)·ϕ(q1)=(p-1)·(q-1)·(p1-1)·(q1-1)`了。

然后后面的参考以前积累的`RSA`解密流程做即可：



.

.

费马因子分解代码：

（取自https://blog.csdn.net/weixin_56678592/article/details/120555169?utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link）

```python
import gmpy2
from gmpy2 import *
from Crypto.Util.number import *
import sympy
e = 0x10001
n1 = 6348779979606280884589422188738902470575876294634928314659473603635680262809639892915911577103896292161096
1527475471832998799055183611566087910323412992194382406141639626435811021604799433111992050343149150952960474246
8032906950984256964560405062345280120526771439940278606226153077959057882262745273394986607004406770035459301695
8063785988905894325389162198214777770214601891400815217791032269535444264418232447658283429730864229490179377012
6134896354103512866146406876903377239032042679504461775190978791418598591127762840463253353039076125725155207349
3697518547350246993679844132297414094727147161169548160586911
c1 = 6201882078995455673376327652982610102807874783073703018551044780440620679217833227711395689114659144506630609
0908760091511694011100202624105680818965896908953259775799542369496666794825043857963989058069039240066171186426
4184444018345499567505424672090632235109624193289954785503512742400960515331371813467034511130432319427185134018
8300069186827338486182010886496904228189403851235994685957663456689318822497794157881293165940832694122218047748
5603879624803870027550939759935153328001490889406814105669466031981604635746268468894251984944123787801848003614
5051967731081582598773076490918572392784684372694103015244826
def fermat_factorization(n):
    factor_list = []
    get_context().precision = 2048 #这里应该是n1的数量级吧，len(bin(n1))就等于2048bit。
    sqrt_n = int(sqrt(n))
    c = sqrt_n
    while True:
        c += 1
        d_square = c**2 - n
        if is_square(d_square):
            d_square = mpz(d_square)
            get_context().precision = 2048
            d = int(sqrt(d_square))
            factor_list.append([c+d,c-d])
        if len(factor_list)==2:
            break
    return factor_list

factor_list = fermat_factorization(n1)
[X1,Y1] = factor_list[0] #费马函数分解最大的特征就是输出两组互相交叉的p * q1， p1 * q,和p * q ，p1 * q，我们必须用gcd(X1,X2)和gcd(Y1,Y2)求出对应的p、q
[X2,Y2] = factor_list[1]
assert X1*Y1 == n1
assert X2*Y2 == n1
p1 = gcd(X1,X2)
q1 = X1 // p1  #这是python向下取整除运算符,我真的第一次发现python有这个运算符。
p2 = gcd(Y1,Y2)
q2 = Y1 // p2

phi1 = (p1-1)*(q1-1)*(p2-1)*(q2-1) #求φ(n)
d1 = invert(e,phi1)  #常规RSA解密流程求d
print(long_to_bytes(gmpy2.powmod(c1,d1,n1)),end='') ##常规RSA解密流程求明文
#flag{Euler_funct1ons
```

.
结果：

```
└─$ python 2.py
b'flag{Euler_funct1ons'
```

.
.
.
（这里积累第三个经验）
接下来分析后半段，后半段给了`x+y`和`x*y`的值，按照初中数学的逻辑直接列个方程就可以解出`x`和`y`的值了，但是我看别人博客各种算法来解这个简单的逻辑，什么`欧拉函数`，因式方程看得头都晕。

然后翻着翻着突然发现一个秀儿，他`p2`和`q2`求法是这样的，因为`n2=p2*p2*q2*q2*q2`，所以`q2 = n2 // (p2_mul_q2*p2_mul_q2)`、`p2 = p2_mul_q2 // q2`，真的我都鼓掌了，太棒了，简单题就简单做啊，简单的逻辑就应该这样求才对啊。

所以直接套用脚本即可，这里要注意的是`ϕ(n)`这里，因为因子只有`p2`和`q2`，所以`ϕ(n)=p2*(p2 - 1)*q2*q2*(q2 - 1)`，只用对单个因子减1即可。

```
import libnum
from Crypto.Util.number import long_to_bytes

n2=405882270455953040803603850410822385070442927313444658152960329056335255569437876107126516754608107687627634935791298312710181415915462075574108174324551393155276749329330852992775991739719124452265322358145808795853172113495244064242006226758809923907820251586212414996934002880316581944346417180269106523279332538773131061128612833142746351247348173984650593735621946949578412648343126409262788903860896111037149906465414705773515995269044583426604449685911976068203613647616482052410414446811458207990544131794622855096611243620740935834949327062494619542404088270870155255071730821294122344862280920028418683658958374636992009599157827676572587297940377764019953092449411714158424036174867194924836714908345625792255068314968815425305195954389324827968678532341596644094209775261024803851931018837851610802695737071566268385515060244554806502243058945019685834423468071269207407797805936508716459151496894242929126115782919127218968647729504102666290455424800092665740960801387096834664895682905693634784443495634985075308055025110511651608271927955201827208024222133642473557752228582146486030347436791874708442125291343749757375109822879573168781799646023947496014318231679821574348904592453943707289427901171564852681167580526367944172686809014201930022890355387536205554885069263666246412918813532686171309689912589830021653001869719636616664766009983890488805651993172804283498028244483298985027884922333818730262172029819216546738401420958396033606660494761005612683362259025049328006054641361922755938867367464979552702805414235 93

p2_add_q2=274773146761138462708137582309097386437793891793691383033856524303010811294101933454824485010521468914846151819876043508541879637544444256520741418495479393777132830985856522008561088410862815913292288683761657919121930016956916865849261153721097671315883469348972925757078089715102032241818526925988645578778

p2_mul_q2=185147242700309621725669659417232243863740762942326522587010857810187761728433559205660351573315795249801081907391419599265230821422736727418495524751562783971315713600990185920189597856277851301264779827652104985476803672307236344240360095393478543445735378486280614688921661998662279841678431397934296825592413170729793740029126075490394313982671848187715034681163796182493193247889963213407646245934431063541042744726011702298352196380932425575478400608925275769400771629900696870199669468262101123184082697492943665866827326143724342187687205779173687265302008975589126874700885837747117675995800376633789290002 17

e = 0x10001

c2 = 255910901685448217617460241787246608395909481904513292274811685764907172422945207398656020610825587597511964521177206474265982615685724409423700397029328219413667921401734284883449322035763342926482555511712748288216570976671067928722000825793199633105037214355006231460129544746131508480834251269875545946517974777418286552382435502669722167525937887348363731443632176396124923972288082152058622812787740963176159188544039926207209691737881512154899088127491798618031449371695874520080970089407100913611839422682452711544618721028136027544399397475665071165193628212557241790930510419947308564014939967712761723433130457559167510826931498859221054918182250128445192649331376229290249186194775385215335485517897396989330672123055784804161636091371898917972092775574111696436585403923030367199521405544353388516714409528651510773832203052950016328164421440224377630891331418869242657742472903066698250858623517323363956172761003742371595807599995930287569393548406773334672816324357670331500524392625010592990352129280415462599331185642511195889700090168738554785565882501389699385999881984945672411723994537417098404869531897642891183128705809931156367107241398097082563602127281277863944116764278284315690462796874813682151375615007774803805015516165778324995212956552373601841598891518377663531161853203177746452942010447728280990749170778966319096546716125572076538303448976441159363221283514945510046529815507587912854348098168723819004014407435781045823052154888885631660545688021459213997266737527228206468074946572991041901239456756 47

q2 = n2 // (p2_mul_q2*p2_mul_q2)
p2 = p2_mul_q2 // q2
d2 = libnum.invmod(e, p2*(p2 - 1)*q2*q2*(q2 - 1))  #invmod(a, n) - 求a对于n的模逆,这里逆向加密过程中计算ψ(n)=(p-1)(q-1)，对ψ(n)保密,也就是对应根据e*d=1modψ(n),求出d
m = pow(c2, d2, n2)   # 这里的m是十进制形式,pow(x, y[, z])--函数是计算 x 的 y 次方，如果 z 在存在，则再对结果进行取模，其结果等效于 pow(x,y) %z，对应前面解密算法中M=D(C)=（C^d）mod n
string = long_to_bytes(m)  # 获取m明文
print(string)
```

.
结果：

```
└─$ python 1.py
b'_1s_very_interst1ng}'
```

.
.
.
总结：

> 1：
> （这里积累第一个经验）
> 先看第一段，`gmpy2.next_prime(p)`函数的意思是取邻近p的下一个素数，那么很明显`p和p1`，`q和q1`都是两两相邻的素数，相差很小，`n1`是`2048bit`级别的，没法硬钢。
> .
> 后半段的`n2`和前半段的加密逻辑差别很大，所以不是两个n的`模不互素`。也不是像上一道题RSA1一样的`密文与模数`不互素。
> .

2：

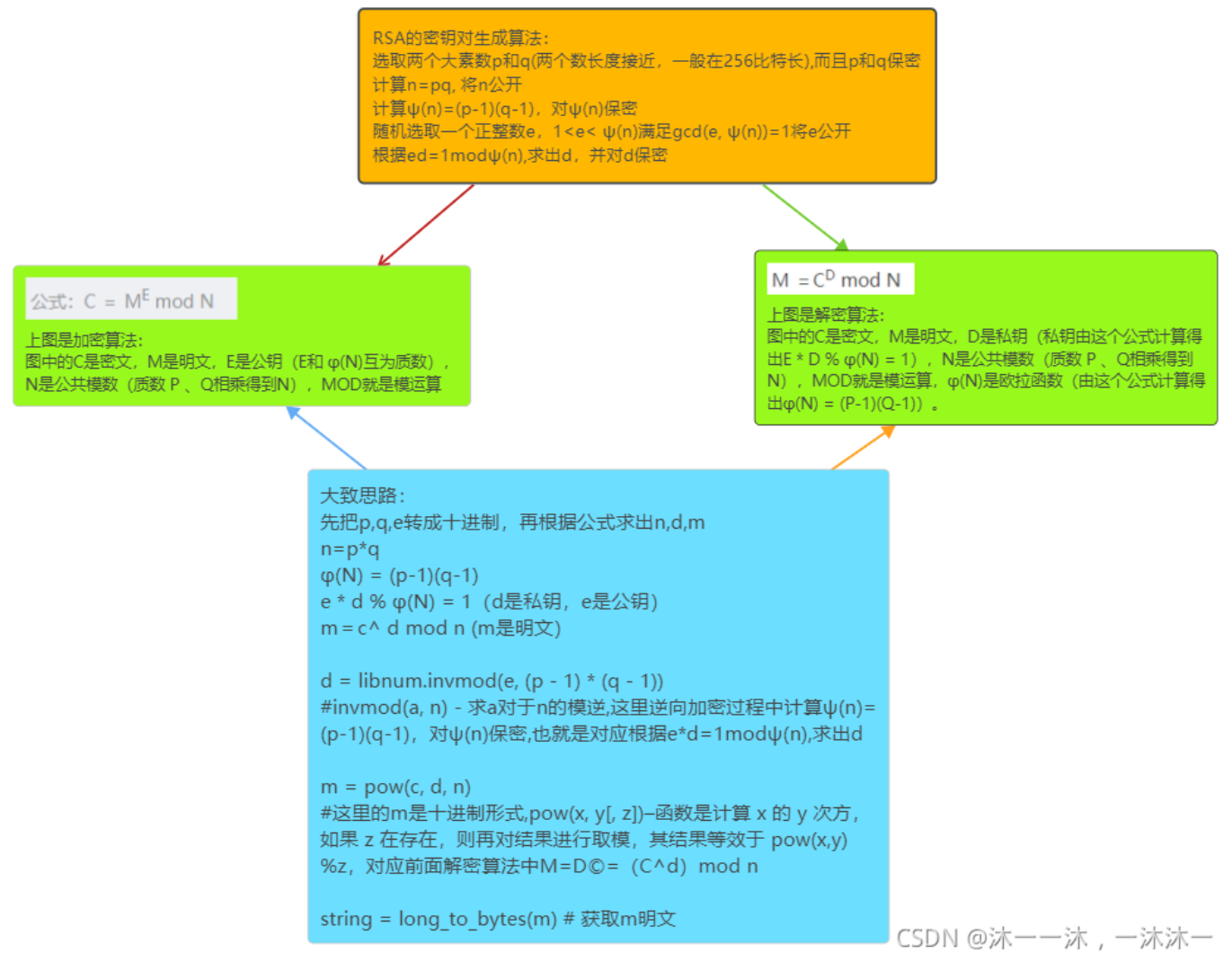
（这里积累第二个经验）

然后查了资料后发现关键在 p和p1 ， q和q1 都是两两相邻的素数这里，也就是说 p,q,p1,q1两两互素 。单纯的 factordb.com 和 yafu 分解n肯定是分解不出来的，毕竟是 2048bit 嘛。所以我们需要一些算法，比如前面 模不互素 的 欧几里得算法 这样，这里我们要用的是 费马分解 算法。

费马分解算法的特征就是 n 是 4 个数的乘积，分解n之后我们会得到 p、p1、q、q1 四组隔开的排列组合，但是我们的脚本可以把组合限定成`p

- q1， p1 * q ,和 p * q ， p1 * q 这样。然后通过 gcd(pq1,pq) = p 、 gcd(p1q,p1q) = q 求出两组各一个数，然后就可以求出 φ(n)=φ§·φ(p1)·φ(q)·φ(q1)=(p−1)·(q−1)·(p1−1)·(q1−1)`了。

然后后面的参考以前积累的 RSA 解密流程做即可：



3：

（这里积累第三个经验）

接下来分析后半段，后半段给了 x+y 和 x*y 的值，按照初中数学的逻辑直接列个方程就可以解出 x 和 y 的值了，但是我看别人博客各种算法来解这个简单的逻辑，什么 欧拉函数 ，因式方程看得头都晕。

.

然后翻着翻着突然发现一个秀儿，他 p2 和 q2 求法是这样的，因为 n2=p2*p2*q2*q2*q2 ，所以 q2 = n2 // (p2_mul_q2*p2_mul_q2)、p2 = p2_mul_q2 // q2 ，真的我都鼓掌了，太棒了，简单题就简单做啊，简单的逻辑就应该这样求才对啊。

.

所以直接套用脚本即可，这里要注意的是 ϕ(n) 这里，因为因子只有 p2 和 q2 ，所以 ϕ(n)=p2*(p2 - 1)*q2*q2*(q2 - 1) ，只用对单个因子减1即可。

解毕！敬礼！