L'ATTAQUES DES RSA

Jiayan

31 mai 2022

Résumé

Dans cette article ,on va présenter l'algo de chiffrement RSA et ses attaques comme des factorisation violente,module commun,aveuglant,attaque de wiener,attaque de coppersmith,attaque de hasard.......

1 Introduction de RSA

Le chiffrement RSA (nommé par les initiales de ses trois inventeurs) est un algorithme de cryptographie asymétrique, très utilisé dans le commerce électronique, et plus généralement pour échanger des données confidentielles sur Internet.

1.1 ALGO DE RSA

1.Choisir p,q deux grands nombres premiers aléatoires
2.Poser n=pq
3.Calculer$\phi(n) = (p-1)(q-1)$
4.$Choisir e \in (Z/\phi(n)z)*$
5.$Calculer d = e^{-1} \pmod{\phi(n)}$

Clé plublic	n (n=pq),e :e est premier avec $\phi(n)$
Clé privé	$d = e^{-1} \pmod{\phi(n)}$
Chiffré	$c = m^e \pmod{n}$
Dechiffré	$m = c^d \pmod{n}$

CODE DE RSA

```
import math
import random
import time
import math


def pow_mod(p, q, n):
    res = 1
    while q :
        if q & 1:
            res = (res * p) % n
        q >>= 1
        p = (p * p) % n
    return res

def gcd(a,b):
    while a!=0:
        a, b =b%a , a
    return b

def mod_1(x, n):
    x0 = x
```

```
    y0 = n
    x1 = 0
    y1 = 1
    x2 = 1
    y2 = 0
    while n != 0:
            q = x // n
            (x, n) = (n, x % n)
            (x1, x2) = ((x2 - (q * x1)), x1)
            (y1, y2) = ((y2 - (q * y1)), y1)
    if x2 < 0:
            x2 += y0
    if y2 < 0:
            y2 += x0
    return x2

def probin(w):
    list = []
    list.append('1') #1
    for i in range(w - 2):
        c = random.choice(['0', '1'])
        list.append(c)
    list.append('1') # 1
    # print(list)
    res = int(''.join(list),2)
    return res


def prime_miller_rabin(a, n):
    ensemblepremier=(2,3,5,7,11,13,17,19,23,29,31,37,41
                ,43,47,53,59,61,67,71,73,79,83,89,97)
    for y in ensemblepremier:
        if n%y==0:
            #print("%d %d"%(n,y))
            return False

def prime_test(n, k):
    while k > 0:
        a = random.randint(2, n-1)
        if not prime_miller_rabin(a, n):
            return False
        k = k - 1
    return True


def get_prime(w):
    while True:
        prime_number = probin(w)
        for i in range(50):
            u = prime_test(prime_number, 5)
            if u:
                break
            else:
                prime_number = prime_number + 2*(i)
        if u:
            return prime_number
        else:
            continue

if __name__=='__main__':
    p = get_prime(512)
    q = get_prime(512)
    n = p * q # n
    OrLa = (p-1)*(q-1)

    while True:
        e = random.randint (1,999.......999)
        if gcd(e, OrLa) == 1:
            break
        else:
            e = d-1
```

```
d = mod_1(e, OrLa)

print('cl␣privite␣p,q,d␣:\n')
print('p:␣%d\n' % p)
print('q:␣%d\n' % q)
print('d:␣%d\n\n' % d)

print('cl␣public␣n,e␣:\n')
print('n:␣%d\n' % n)
print('e:␣%d\n\n' % e)


M = int(input("entrer␣les␣chiffres␣"))


C = pow_mod(M, e, n)
print('\n␣on␣peut␣obtenir␣les␣chiffrs\n%d\n'%C)

if d < 1/3*pow(n,1/4):
   print("mauvais␣chiffr")
else:print("bon␣chiffr")
```

C'est une résultat si on veut chiffrer un nombre '8876579'par RSA,on peut entrer les message par le clavier pour chiffrer .

```
Last login: Tue May 31 14:31:45 on ttys000
jiangjiayan@jiangjiayandeMacBook-Pro ~ % ./rsanouvel.py
zsh: no such file or directory: ./rsanouvel.py
jiangjiayan@jiangjiayandeMacBook-Pro ~ % cd /Users/jiangjiayan/Downloads/m-moire/code4\ choisir\ chiffre
jiangjiayan@jiangjiayandeMacBook-Pro code4 choisir chiffre % chmod a+x rsanouvel.py
jiangjiayan@jiangjiayandeMacBook-Pro code4 choisir chiffre % ./rsanouvel.py
clé privite p,q,d :

p: 9414933872586169734001273456608197666739540438074171860955081813399461287524426173262431195701809044139770182943031664934288716942006753477400928350869383

q: 8506220613807594833004388056949819010257450706062356414468400440149198819498891638286371578119771695605169272395421417761686613032234654059744660048696283

d: 34461316583759965642448034787675846115554511768701168488019559215464561007084413902984087946407761710709748882070175407230896795903754228201432720332514369450349496713406099900946
75310403239177583641259992440553219519780802883313858768879955730631709490230056789382607645521308380528781


clé public n,e :

n: 80085504584627844558656943535302513066692659459062747718465568837270086949303523537947138947513231202473555653134152723431775104889098914615437484894376679109444235941196242997938
9443380893389104027430046072122896562928976695667006022316107609955013565546483576345228370318143877060338 9

e: 60106737934023395190522709767261753457545899525106590418053276593012793823374350219425446213158463774485872799988806130071462107938365233620557335526626947093747997722812794848831
2615622016807352292355299191234984514727769813929993251446309295647297700060 9


entrer les chiffres : 8876579 on peut entrer les message par la clavier

 on peut obtenir les chiffrés:
1425359909781460122735735555831745204824912767797991930893038132126527244144070184230954143083658086453213557375826883142194840892404344972199982123739143867909630754673264157006259 79
0449241769710204104852433042120494233640610271802069569418938325129240264027892137355640331972007451886 8

bon chiffré avec l'attaque wiener, d doit plus grand que 1/3(N quart de pussiance),donc si d plus grande que ça,c'est bon chiifré
jiangjiayan@jiangjiayandeMacBook-Pro code4 choisir chiffre %
```

2 Nombre d'attaques arithmétiques

2.1 Factorisation violente

2.1.1 Factorisation n directement

Si n n'est pas très grand ,on peut factoriser n directement pour obtenir p et q .Et après avec p et q c'est facile de calculer la clé privée d pour déchiffrer l'algo rsa. Comment factoriser n ? On présente deux exemples "méthode Fermat" et "pollard p-1"

2.1.2 Méthode Fermat

L'intuition est la suivante. Tout entier naturel impair N se décompose en la différence de deux carrés : N=a^2-b^2. Algébriquement, cette différence se factorise en (a + b)(a − b) et, si ni a + b ni a − b n'est égal à 1, alors ce sont des facteurs non triviaux de N.

Il existe une telle représentation pour tout nombre impair composé. Si N = cd est une factorisation de N, alors

$$\left(\tfrac{c+d}{2}\right)^2 - \left(\tfrac{c-d}{2}\right)^2$$

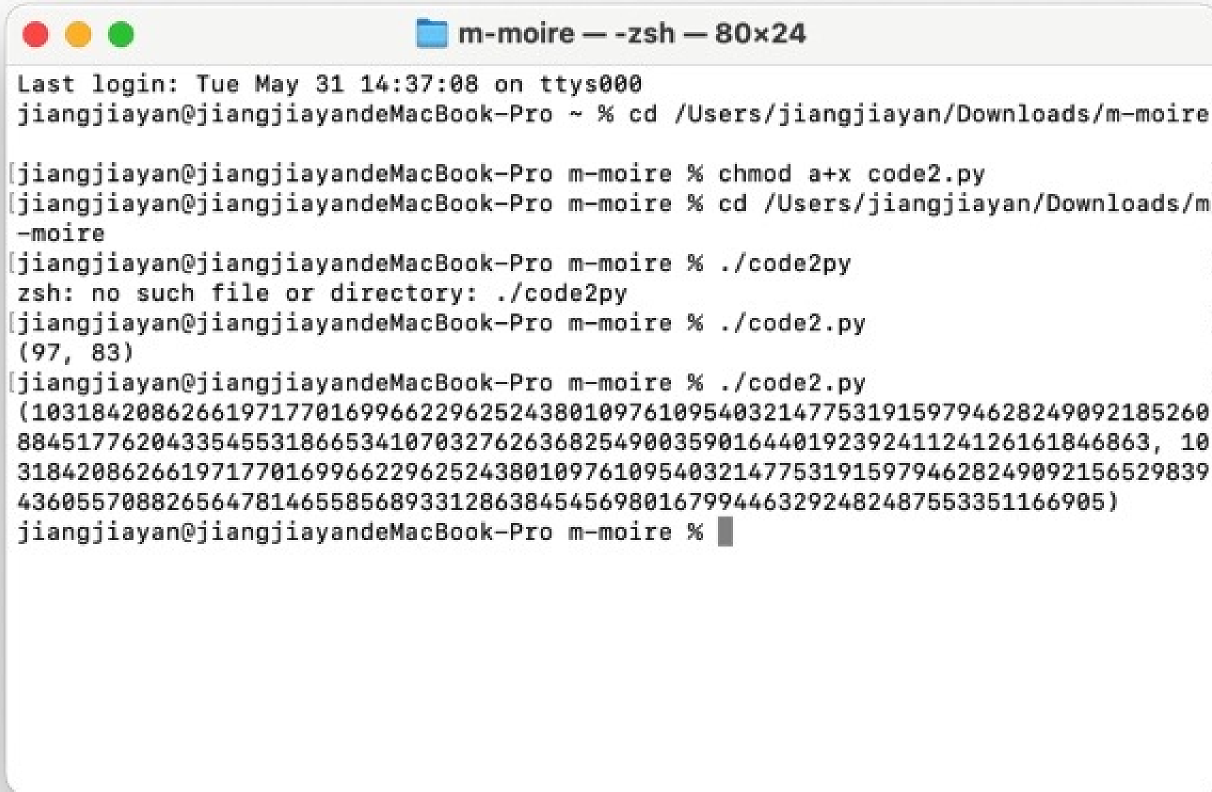
Puisque N est impair, c et d le sont aussi et ces « moitiés » sont des nombres entiers. Notons qu'un multiple de 4 donne aussi une différence de deux carrés, en posant c et d comme nombres pairs.

Dans sa forme la plus simple, la méthode de factorisation de Fermat peut être plus lente que la factorisation par essais de divisions. Néanmoins, la combinaisons des deux méthodes est plus efficace qu'uniquement l'une ou l'autre.

Il est effectué par le code 2

code de fermat

```
from math import *
def fermat(n, verbose=True):
    a = isqrt(n) # int(ceil(n**0.5))
    b2 = a*a - n
    b = isqrt(n) # int(b2**0.5)
    count = 0
    while b*b != b2 and count<1000000:
        a = a + 1
        b2 = a*a - n
        b = isqrt(b2) # int(b2**0.5)
        count += 1
    p=a+b
    q=a-b
    #print(p)
    #print(q)
    return p, q
print(fermat(8051))
```



Avec l'example "8051" dans le poly ,on obtiens le résultat (97,83),avec l'autre example qu'on obtenu par le premier code rsa,on peut aussi le factoriser par méthode fermat

2.1.3 Pollard p-1

Soit n un entier divisible par un nombre premier p, avec n p. D'après le petit théorème de Fermat, on a

$$a^{p-1} = 1 \pmod{P}$$

cela implique que pour tout muliple M de p-1,on a

$$a^{M} = 1 \pmod{P}$$

car $a^{k(p-1)} = (a^{p-1})^k = 1^k = 1 \pmod{P}$

Si p − 1 est B-superlisse pour un certain seuil B, alors p − 1 divise le plus petit commun multiple des entiers de 1 à B. Donc, si l'on pose M = ppcm(1, . . . , B), on a

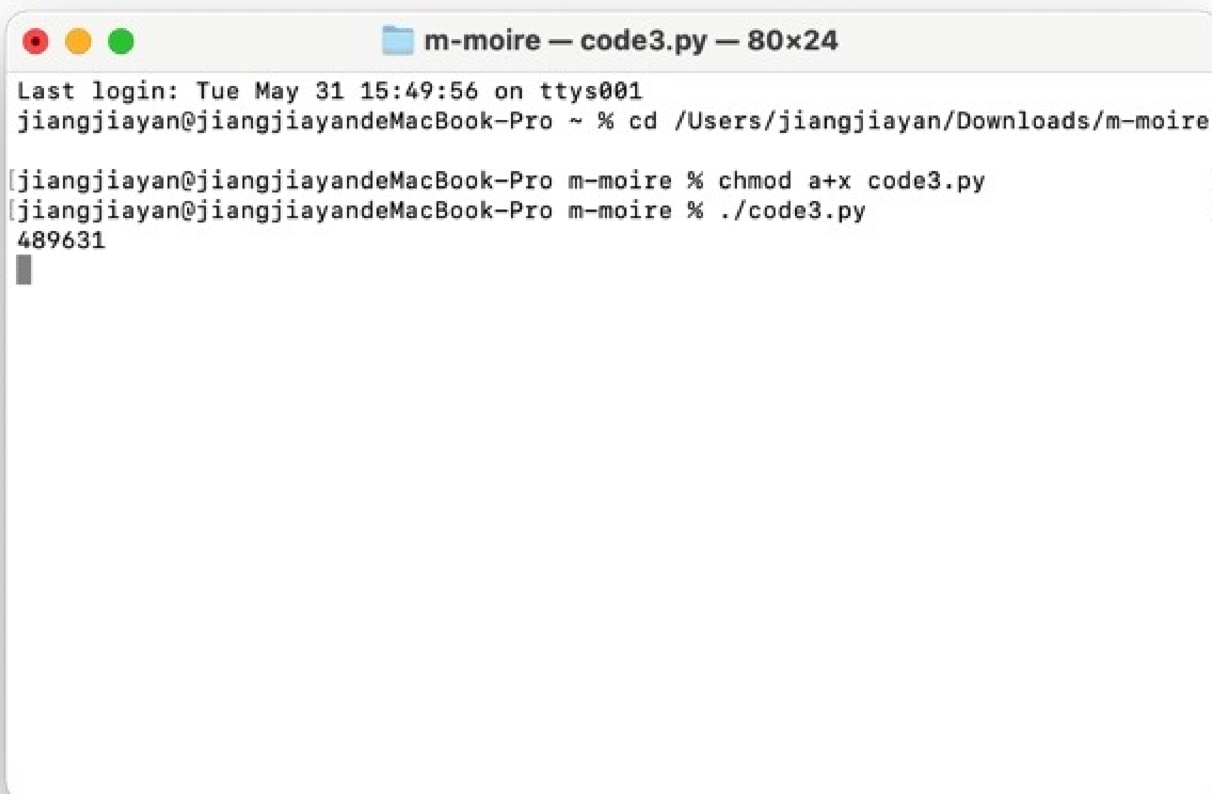
$$a^M = 1 \pmod{P} \text{ pour tout a premier avec p.}$$

Autrement dit, p divise a^M-1 et donc le pgcd de n et a^M-1 est supérieur ou égal à p. En revanche, il est possible que ce pgcd soit égal à n lui-même auquel cas, on n'obtient pas de facteur non trivial.

code de pollard p-1

```
import math
import sys
def pollard(n):
    m = 2
    ans=[]
    max = n
    for i in range(max):
        if(i>0):
            m = pow(m,i,n)
            #print(i)
            if (math.gcd(n,m-1) != 1):
                return math.gcd(n,m-1)
print(pollard(8886599965778657))
print(pollard
     (1064698090990178296911919042837031547499229214034913208259686254345816117126402401237096514170599
75174436713148689510987484478893557111612866409462466323328333162215160821865949027654556723484731121754
831966739068484567634939921443349831401149010917400862761219810128409364342169426640567577168804416260
082449))
```

Mais comme l'image suivant ,il est facile de factoriser les nombre petit ,pour les nombres grands il va prendre beaucoup de temps ,jusqu'à j'ai terminé cette mémoire ,le deuxième exemple n'est pas factorisé.



2.2 Aveuglant

2.2.1 Introduction

RSA est faible dans l'attaque à chiffres choisis .L'attaquant déguise le chiffre intercepté en un nouveau message chiffré et il sera signé par la clé privée.

2.2.2 L'attaque aveuglante

1.Déguiser une nouveau message
Pour obtenir m,Eva choisis un nombre aléatoire r,r<n.Elle calcule

$$x = r^e \pmod n$$

2.Elle encapsule le message déguisé dans y.

$$y = xc \pmod n$$

3.On fait t=$r^{-1} \pmod n$
Par algo RSA,si x=$r^e \pmod n$, donc r=$x^d \pmod n$. Eva envoyai $y = xc \pmod n$ à Alice,car Alice vu jamais y,elle fait un signature de y et obtenir

$$u = y^d \pmod n$$

Maintenant Eva calcule

$$tu \pmod n = r^{-1} y^d \pmod n$$

$$=r^{-1} x^d c^d \pmod n$$

$$c^d \pmod n = m$$

Après ça ,Eva obtient le message clair m
code d'aveuglant

```
##dchiffr##
    def multi(x, c, n):
        res = 0
        while c :
            if c & 1:
                res = (res + x) % n
            c >>= 1
            x = (x + x) % n
        return res

r=int(input("eva choisi un nombre alatoire:"))#eva choisi un nombre alatoire
x=pow_mod(r,e,n)
print("x:",x)
y=multi(x,C,n)
print("y:",y)
u=pow_mod(y,d,n)
print("u:",u)
#if M==mfinal;
#mfinal=u/r;
#print("mfinal",mfinal)
```

```
#else
#print("euueur")
mfinal=u //r
print("mfinal",mfinal)
```

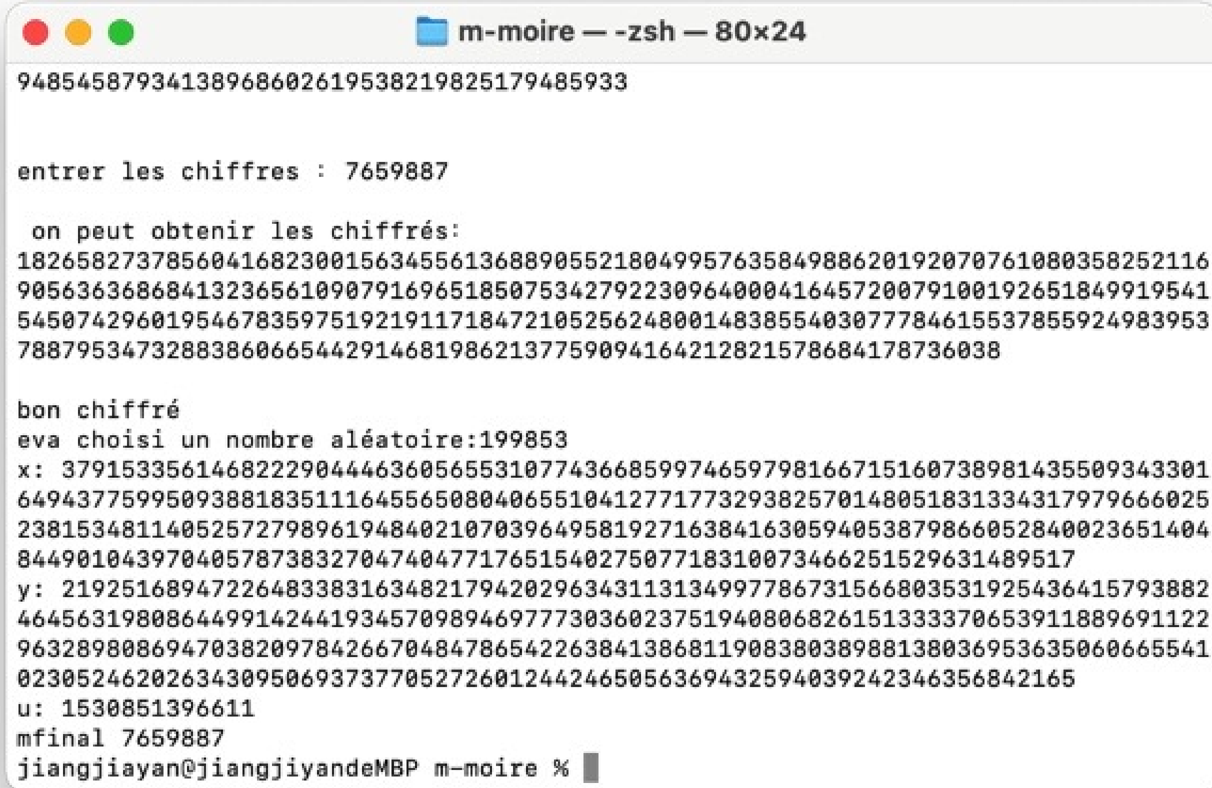
il ne peut pas être exécuté indépendamment ,on doit l'ajouter après "code rsa" pour faire le deuxième chiffrement .on peut voir cette code complète sur le lien https://github.com/jiangjiayan/moire

on peut voire les résultats comme les images suivantes

on veut chiffrer un message "7659887"

```
m-moire — 111.py — 80×24

clé public n,e :

n: 65368919399961532350928035432667426448455143057049227906755090839541812321959
73513100482629478154557729384435063370874431272409714281157191898919182298143418
71324486944637492844241554335954349057809772767768456154394605396064543185431871
857748095698607173878166431870664823835624136968368694311407471619823 59

e: 62648592143246698197477768147911607190269236565770668172956960476860933938308
22065013691420269759591534533385692588423835445575857339143844369868596031474932
59346356171222686719957843398938007616733663584346458279238822799698135758113293
94854587934138968602619538219825179485933


entrer les chiffres : 7659887

 on peut obtenir les chiffrés:
18265827378560416823001563455613688905521804995763584988620192070761080358252116
90563636868413236561090791696518507534279223096400041645720079100192651849919541
54507429601954678359751921911718472105256248001483855403077784615537855924983953
788795347328838606654429146819862137759094164212821578684178736038

bon chiffré
eva choisi un nombre aléatoire:
```

on peut voir que "7659887"est déjà bien chiffré par RSA,maintenant une attaquante Eva entrer un nombre aléatoire ,par exemple "199853"



et au final ,on obtient le même message "7659887".

2.3 Attaque par le module commun

Pour communiquer dans un groupe de personnes,on pourrait envisager l'utilisation d'un module RSA n commun avec des paires de clé distinctes(d_i,e_i).Si on utilise(n,e_1)et(n,e_2)pour chiffre le message m, et puis on obtientc_1,c_2.

Car e_1,$e_2$sont clé public,donc maintenant on a c_1,c_2,e_1,e_2,n.e_1,$e_2$sont premières entre eux,donc pgcd(e_1,e_2)=1

Avec la théorème de bézout,$e_1 s_1$+$e_2 s_2$=1,donc

$$(c_1^{s_1} * c_2^{s_2}) \pmod n$$

$$= ((m^{e_1} \pmod n)^{s_1}) * ((m^{e_2} \pmod n)^{s_2}) \pmod n$$

$$= m^{e_1 s_1 + e_2 s_2} \pmod n$$

$$= m_1 \pmod n) = \mathrm{m}$$

Maintenant, avec l'attaque du module commun, on peut obtenir le message m.

code de module commun

Pour tester cette attaque,on doit écrire un code pour chiffrer un message par même module ,et après on obtient c_1 , c_2 ,(e_1,e_2 sont premiers)

code chiffré par même N

```
import math
import random
import time
import math


def pow_mod(p, q, n):
    res = 1
    while q :
        if q & 1:
            res = (res * p) % n
        q >>= 1
        p = (p * p) % n
    return res

M = int(input("message:"))
e = int(input("e:"))
n = int(input("n:"))
C = pow_mod(M, e, n)
print('\n on peut obtenir les chiffrs\n%d\n'%C)
```

On chiffre un exemple :message est "199853",on veut le chiffrer deux fois par même N et e différent (e_1 =13 et e_2=17 n=65368919399961532350928035432667426448455143057049227906755090839541812321959735131004826294781545577293844350633708744312724097142811571918989191822981434187132448694463749284424155433595434905780977276776845615439460539606454318543187185774809569860717387816643187066482383562413696836869431140747161982359)

Maintenant ,on teste l'attaque module commun.

code moudle commun

```
import libnum
import gmpy2

def common_modulus(n, c1, c2, e1, e2):
    assert(libnum.gcd(e1, e2))
    _, s1, s2 = gmpy2.gcdext(e1, e2)#
    #s1<0c1^s1==(c1^-1)^(-s1)c1^-1c1n
    if s1 < 0:
        s1 = -s1
        c1 = gmpy2.invert(c1, n)
    if s2 < 0:
        s2 = -s2
        c2 = gmpy2.invert(c2, n)
    return pow(c1, s1, n)*pow(c2, s2, n)%n
n=653689193999615323509280354326674264484551430570492279067550908395418123219597351310048262947815455772938443506337087443127240971428115719189891918229814341871324486944637492844241554335954349057809772767768456154394605396064543185431871857748095698607173878166431870664823835624136968368694311407471619823 59
c1 = 81140697022275152614992669700940291309247357928432288303882494136 9373
c2 = 1294438499993263872044401155001753261347663405773293008800457111730921019394633135266821613
e1 = 13
e2 = 17
print(common_modulus(n, c1, c2, e1, e2))
```

Dans l'image suivante ,après l'attaque de module commun,on a déjà obtenu le message"199853"

2.4 Faible exposant privée (Attaque de wiener)

On sait que la sécurité de l'algorithme RSA repose sur la difficulté de factoriser de grands entiers. Selon RSA,e$\in$ $(Z/\phi(n)$z)* .si on choisit e mauvais, il rendra cet algorithme de chiffrement non sécurisé.

Comme l'attaque de Wiener,si d est trop petit comme d<$\frac{1}{3}N^{\frac{1}{4}}$. l'attaque Wiener peut casser l'algorithme de chiffrement RSA.

(On va expliquer pourquoi d<$\frac{1}{3}N^{\frac{1}{4}}$ plus tard)

Tout d'abord, on doit introduire' la fraction continue' . Une fraction continue est une expression de la forme.

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 \cfrac{1}{a_3 + \cfrac{1}{a_4 + ...}}}}$$

Pour tout entier, on peut le décomposer en fractions continuespar exemple,(72,28)

$$\tfrac{72}{28} = 2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{3}}}$$

Quand on décompose une fraction en fraction continue, en fait, on utilise l'algorithme d'Euclide constamment. Par conséquent, on arrive à une conclusion :il existe toujours un nombre fini de fraction continue pour tous les nombres rationnels. Pour la division d'algorithme d'Euclide de deux nombres, on obtenira le pgcd de deux nombres.

Après l'introduction de fraction continue, on retourne l'algo RSA. N=pq et$\phi(n)$=(p-1)*(q-1)

$$\phi(n)\text{=(p-1)*(q-1)}$$

$$\text{=pq-(p+q)+1}$$

$$\text{=N-(p+q)+1}$$

p et q sont grands nombres premiers,pq est plus grand que (p+q),donc on peut considérer

$$\phi(n) \approx N$$

On sait que d=$e{-}1 \pmod n$,et pius on a une égalité :

$$\text{ed}=1 \pmod n$$

c'est à dire :

$$\text{ed-1=k*}\phi(n)$$

On divise par d des deux côtés de l'équationet obtiens une autre l'équation comme suivant

$$\frac{e}{\phi(n)} - \frac{k}{d} = \frac{1}{d*\phi(n)}$$

Avec $\phi(n) \approx N$,Car (n)N,on peut l'ecrire aussi

$$\frac{e}{N} - \frac{k}{d} = \frac{1}{d*\phi(n)}$$

Évidemment ,d*$\phi(n)$est grand ,donc $\frac{1}{d*\phi(n)}$ est trop petit .C'est à dire que $\frac{e}{N}$ est juste un peu plus grand que $\frac{k}{d}$.Car e et N on déjà connu,avec décomposer $\frac{e}{N}$par fraction continue ,l'un d'eux sera égal $\frac{k}{d}$.

Pourquoi $\frac{k}{d}$est l'un d'étendue de fraction continue $\frac{e}{N}$?

Il y a une théorème important de fraction continue :

The important result is that if p et q are two integers with

$$\left|\alpha - \frac{p}{q}\right| \leq \frac{1}{2q^2}$$

then $\frac{p}{q}$is a convergent in the continued fraction expansion of α

Maintenant on utilise cette théorème à l'algo de RSA. Supposons que q < p < 2q. Puisque N=pq > q^2 ,alors q < $\sqrt{N}$ D'autre part, on a

Si e est une clé publique et d la clé privée ,alors on a

$$\phi(N)\text{=(p-1)(q-1)=N-p+q)+1.}$$

donc

$$\text{ed=1+k(n-(p+q)+1)}$$

ce qui donne en divisant par dN les relations successives suivantes :

$$\frac{e}{N}=\frac{k}{d}+\frac{1+k-k(p+q)}{dN}$$

$$\left|\frac{e}{N} - \frac{k}{d}\right|=\frac{k(p+q)-k-1}{dn}$$

$$\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{k(p+q)}{dN}$$

$$\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{kq(\frac{p}{q}+1)}{dN}$$

$$\left|\frac{e}{N} - \frac{k}{d}\right| < \text{3kq } \frac{3kq}{dN}$$

$$\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{3kd}{\sqrt{N}}$$

donc si :

$$d \leq \frac{n^{0.625}}{\sqrt{6}}$$

Et en conséquence : $\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{1}{2d^2}$

Comme on connaît e et N,L'attaque peut développer e et N en fraction continue .Avec une réduit $\frac{e}{N}$

D'autres part on peut calculer $\phi(N)$ par relation(factorisation de N),

$$\phi(N)=\frac{ed-1}{k}$$

$\phi(N)$=(p-1)(q-1)=pq-p-q+1=N-p-q+1

En multipliant par p ou q ,on trouve p*$\phi(N)$ =Np- p^2 -pq+p

Donc p*$\phi(N)$ -Np+ p^2 +pq -p =0

Donc p^2 -(N-$\phi(N)$ +1)*p+N=0 p et q sont solutions de l'équation x^2 -(N- $\phi(N)$ +1)*x+N=0

$\delta = (N - \phi + 1)^2$ -4N,q= $\frac{N-\phi(N)+1-\sqrt{\delta}}{2}$,p=$\frac{N-\phi(N)+1+\sqrt{\delta}}{2}$