

L'ATTAQUES DES RSA

Jiayan

1^{er} juin 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 est un algorithme de cryptographie asymétrique, le plus utilisé actuellement ,L'algorithme RSA est théoriquement solide, avec une bonne sécurité, et peut être utilisé pour le cryptage des données, les signatures numériques et l'authentification d'identité, répondant ainsi aux différents besoins de sécurité des réseaux.

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 $e \in (\mathbb{Z}/\phi(n)\mathbb{Z})^*$
- 5.Calculer $d = e^{-1} \pmod{\phi(n)}$

Clé public 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 > 0:
        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 privée p,q,d:\n')
print('p:\n' % p)
print('q:\n' % q)
print('d:\n\n' % d)

print('cl publique n,e:\n')
print('n:\n' % n)
print('e:\n\n' % e)

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

C = pow_mod(M, e, n)
print('\non peut obtenir les chiffres\n%d\n'%C)

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

```

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 privée p,q,d :
[
p: 94149338725861697340012734566081976667395404380741718609550818133994612875244261732624311957018090441397701829430831664934288716942006753477400928350869383
q: 8506220613807594833004388056949819010257450706062356414468400440149198819498891638286371578119771695605169272395421417761686613032234654059744660048696283
d: 34461316583759965642448034787675846115554511768701168488019559215464561007084413902984087946407761710709748882070175407230896795903754228201432720332514369450349496713406099900946
75310403239177583641259992440553219519780802883313850768879955730631709490230056789382607645521308380528781

clé publique n,e :
n: 80085504584627844558656943535302513066692659459062747718465568837270086949303523537947138947513231202473555653134152723431775104889098914615437484894376679109444235941196242997938
94433808933891040274300460721228965629289766956670060223161076099550135655464835763452283703181438770603389
e: 60106737934023395190522709767261753457545899525106590418053276593012793823374350219425446213158463774485872799988806130071462107938365233620557335526626947093747997722812794848831
26156220168073522923552991912349845147277698139299932514463092956472977000609

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

on peut obtenir les chiffrés:
142535990978146012273573555831745204824912767797991930893083132126527244144070184238095414308365808645321355737582688314219484089240434497219998212373914386790963075467326415700625979
04492417697102041048524330421284942336406102718020695694189383251292402640278921373556403319720074518868

bon chiffré avec l'attaque wiener, d doit plus grand que 1/3(N quart de puissance), donc si d plus grande que ça, c'est bon chiffré
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'algorithme RSA. Comment factoriser n ? On présente deux exemples "méthode Fermat" et "pollard $p-1$ ".

2.1.2 Méthode Fermat

En arithmétique modulaire, la méthode de factorisation de Fermat est un algorithme de décomposition en produit de facteurs premiers d'un entier naturel.

Tous les entiers naturels impairs N peuvent être considérés comme la différence de deux nombres carrés : $N = a^2 - b^2$. Algébriquement, N peut aussi se factoriser 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

$$N = \left(\frac{c+d}{2}\right)^2 - \left(\frac{c-d}{2}\right)^2$$

Puisque N est entier impair, donc $\left(\frac{c+d}{2}\right)^2$ et $\left(\frac{c-d}{2}\right)^2$ sont des nombres entiers.

C'est-à-dire qu'après ces étapes, il est inévitable que le produit des deux facteurs apparaisse sous la forme $(a+b)(a-b)$. Par conséquent, le nombre d'origine peut être écrit comme la différence des deux carrés des deux nombres : $a^2 - b^2$. Par conséquent, nous devons pouvoir trouver a et b par la méthode de Fermat. En d'autres termes, s'il n'est pas trouvé, cela signifie que le nombre d'origine doit être un nombre premier.

Mais cette méthode a des limites, elle est très efficace lorsque les deux facteurs sont relativement proches

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))
```

```
m-moire — -zsh — 80x24
Last login: Tue May 31 14:37:08 on ttys000
jiangjiayan@jiangjiayandeMacBook-Pro ~ % cd /Users/jiangjiayan/Downloads/m-moire
[jiangjiayan@jiangjiayandeMacBook-Pro m-moire % chmod a+x code2.py ]
[jiangjiayan@jiangjiayandeMacBook-Pro m-moire % cd /Users/jiangjiayan/Downloads/m]
-moire
[jiangjiayan@jiangjiayandeMacBook-Pro m-moire % ./code2py ]
zsh: no such file or directory: ./code2py
[jiangjiayan@jiangjiayandeMacBook-Pro m-moire % ./code2.py ]
(97, 83)
[jiangjiayan@jiangjiayandeMacBook-Pro m-moire % ./code2.py ]
(1031842086266197177016996622962524380109761095403214775319159794628249092185260
8845177620433545531866534107032762636825490035901644019239241124126161846863, 10
31842086266197177016996622962524380109761095403214775319159794628249092156529839
4360557088265647814655856893312863845456980167994463292482487553351166905)
jiangjiayan@jiangjiayandeMacBook-Pro m-moire % █
```

Avec l'exemple "8051" dans le poly ,on obtiens le résultat (97,83),avec l'autre exemple 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 $n \neq p$. D'après le petit théorème de Fermat, on a

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

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

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

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

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 $M = \text{ppcm}(1, \dots, B)$, on a

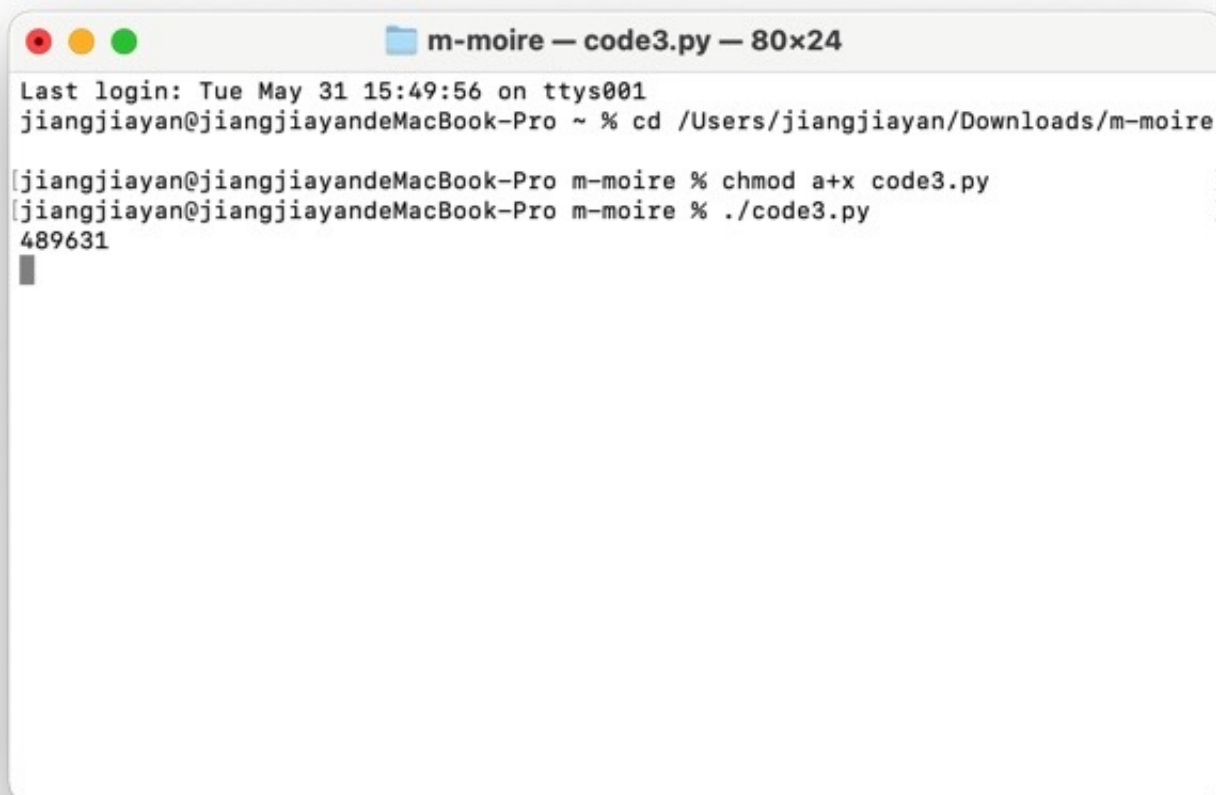
$$a^M = 1 \pmod{P} \text{ pour tout } a \text{ premier avec } p.$$

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.



```
m-moire — code3.py — 80x24
Last login: Tue May 31 15:49:56 on ttys001
jiangjiayan@jiangjiayandeMacBook-Pro ~ % cd /Users/jiangjiayan/Downloads/m-moire
[jiangjiayan@jiangjiayandeMacBook-Pro m-moire % chmod a+x code3.py
[jiangjiayan@jiangjiayandeMacBook-Pro m-moire % ./code3.py
489631
█
```

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 un nouveau message

Pour obtenir m , Eva choisit 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 l'algorithme RSA, si $x = r^e \pmod{n}$, donc $r = x^d \pmod{n}$. Eva envoie $y = xc \pmod{n}$ à Alice, car Alice ne voit jamais y , elle fait une signature de y et obtient

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

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

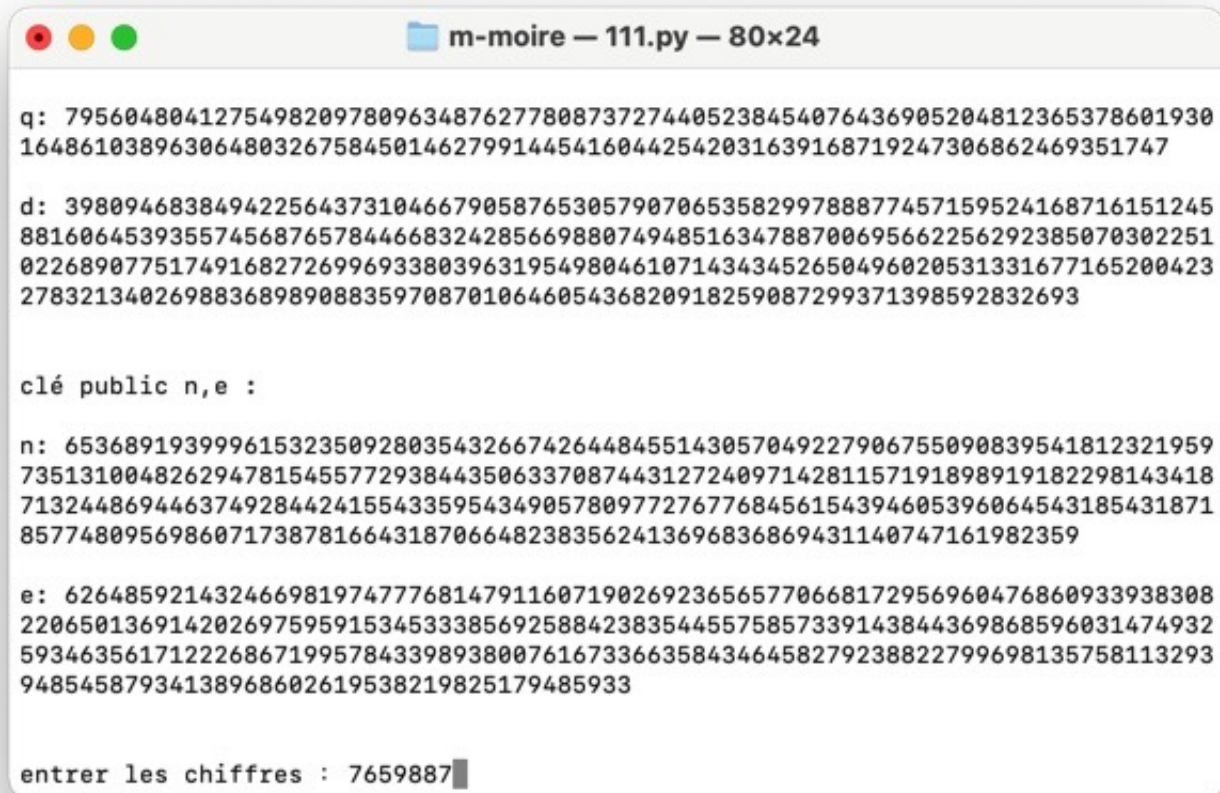
```
##dchiffre##
def multi(x, c, n):
    res = 0
    while c :
        if c & 1:
            res = (res + x) % n
        c >>= 1
        x = (x + x) % n
    return res

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")
mfina1=u //r
print("mfina1",mfina1)
```

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 voir les résultats comme les images suivantes



```
m-moire — 111.py — 80x24

q: 79560480412754982097809634876277808737274405238454076436905204812365378601930
16486103896306480326758450146279914454160442542031639168719247306862469351747

d: 39809468384942256437310466790587653057907065358299788877457159524168716151245
88160645393557456876578446683242856698807494851634788700695662256292385070302251
02268907751749168272699693380396319549804610714343452650496020531331677165200423
27832134026988368989088359708701064605436820918259087299371398592832693

clé public n,e :

n: 65368919399961532350928035432667426448455143057049227906755090839541812321959
73513100482629478154557729384435063370874431272409714281157191898919182298143418
71324486944637492844241554335954349057809772767768456154394605396064543185431871
85774809569860717387816643187066482383562413696836869431140747161982359

e: 62648592143246698197477768147911607190269236565770668172956960476860933938308
22065013691420269759591534533385692588423835445575857339143844369868596031474932
59346356171222686719957843398938007616733663584346458279238822799698135758113293
94854587934138968602619538219825179485933

entrer les chiffres : 7659887
```

on veut chiffrer un message "7659887"


```
m-moire — 111.py — 80x24

clé public n,e :

n: 65368919399961532350928035432667426448455143057049227906755090839541812321959
73513100482629478154557729384435063370874431272409714281157191898919182298143418
71324486944637492844241554335954349057809772767768456154394605396064543185431871
85774809569860717387816643187066482383562413696836869431140747161982359

e: 62648592143246698197477768147911607190269236565770668172956960476860933938308
2206501369142026975959153453385692588423835445575857339143844369868596031474932
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"

```

m-moire — -zsh — 80x24
94854587934138968602619538219825179485933

entrer les chiffres : 7659887

on peut obtenir les chiffrés:
18265827378560416823001563455613688905521804995763584988620192070761080358252116
90563636868413236561090791696518507534279223096400041645720079100192651849919541
54507429601954678359751921911718472105256248001483855403077784615537855924983953
788795347328838606654429146819862137759094164212821578684178736038

bon chiffré
eva choisi un nombre aléatoire:199853
x: 37915335614682229044463605655310774366859974659798166715160738981435509343301
64943775995093881835111645565080406551041277177329382570148051831334317979666025
23815348114052572798961948402107039649581927163841630594053879866052840023651404
84490104397040578738327047404771765154027507718310073466251529631489517
y: 21925168947226483383163482179420296343113134997786731566803531925436415793882
46456319808644991424419345709894697773036023751940806826151333370653911889691122
96328980869470382097842667048478654226384138681190838038988138036953635060665541
02305246202634309506937377052726012442465056369432594039242346356842165
u: 1530851396611
mfinal 7659887
jiangjiayan@jiangjiyandeMBP m-moire %

```

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 chiffrer le message m, et puis on obtient c_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 $\text{pgcd}(e_1, e_2) = 1$

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

$$\begin{aligned}
 & (c_1^{s_1} * c_2^{s_2}) \pmod{n} \\
 &= ((m^{e_1} \pmod{n})^{s_1}) * ((m^{e_2} \pmod{n})^{s_2}) \pmod{n} \\
 &= m^{e_1 s_1 + e_2 s_2} \pmod{n} \\
 &= m_1 \pmod{n} = m
 \end{aligned}$$

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 chiffrés\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 $e_2 = 17$ $n = 65368919399961532350928035432667426448455143057049227906755090839541812321959735131004826294781545577293844350633708744312724097142811571918989191822981434187132448694463749284424155433595434905780977276776845615439460539606454318543187185774809569860717387816643187066482383562413696836869431140747161982359$)



```
jiangjiayan@jiangjiayandeMacBook-Pro mémoire code % ./code7.py
message:199853
e:13
n:653689193999615323509280354326674264484551430570492279067550908395418123219597
35131004826294781545577293844350633708744312724097142811571918989191822981434187
13244869446374928442415543359543490578097727677684561543946053960645431854318718
5774809569860717387816643187066482383562413696836869431140747161982359

on peut obtenir les chiffrés:
811406970222751526149926697009402913092473579284322883038824941369373

jiangjiayan@jiangjiayandeMacBook-Pro mémoire code % ./code7.py
message:199853
e:17
n:653689193999615323509280354326674264484551430570492279067550908395418123219597
35131004826294781545577293844350633708744312724097142811571918989191822981434187
13244869446374928442415543359543490578097727677684561543946053960645431854318718
5774809569860717387816643187066482383562413696836869431140747161982359

on peut obtenir les chiffrés:
12944384999932638720444011550017532613476634057732930088004571117309210193946331
35266821613

jiangjiayan@jiangjiayandeMacBook-Pro mémoire code %
```

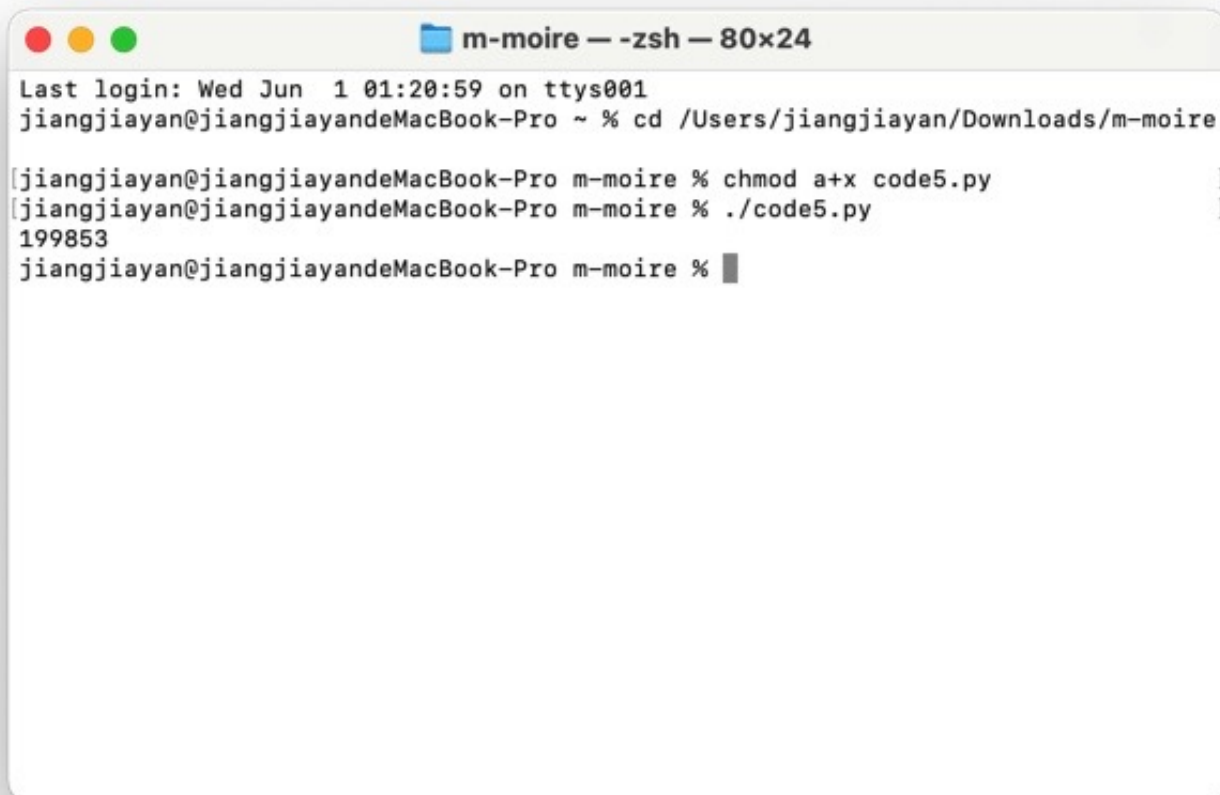
Maintenant ,on teste l'attaque module commun.

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=65368919399961532350928035432667426448455143057049227906755090839541812321959735131004826294781
5455772938443506337087443127240971428115719189891918229814341871324486944637492844241554335954349
0578097727677684561543946053960645431854318718577480956986071738781664318706648238356241369683686
9431140747161982359
c1 = 811406970222751526149926697009402913092473579284322883038824941369373
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"



```
m-moire — -zsh — 80x24
Last login: Wed Jun 1 01:20:59 on ttys001
jiangjiayan@jiangjiayandeMacBook-Pro ~ % cd /Users/jiangjiayan/Downloads/m-moire
[jiangjiayan@jiangjiayandeMacBook-Pro m-moire % chmod a+x code5.py
[jiangjiayan@jiangjiayandeMacBook-Pro m-moire % ./code5.py
199853
jiangjiayan@jiangjiayandeMacBook-Pro m-moire %
```

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.

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{a_4 + \dots}}}}$$

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

$$\frac{72}{28} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{3}}}$$

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 obtiendra le pgcd de deux nombres.

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

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

$$= pq - (p+q) + 1$$

$$= N - (p+q) + 1$$

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 $d \equiv e^{-1} \pmod{n}$, et puis on a une égalité :

$$ed \equiv 1 \pmod{n}$$

c'est à dire :

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

On divise par d des deux côtés de l'équation et 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 $\phi(n) < N$, on peut l'écrire aussi

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

Évidemment, $\frac{1}{d*\phi(n)}$ 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

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

donc

$$ed=1+k(n-(p+q)+1)$$

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

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

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

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

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

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

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

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 réduite $\frac{e}{N}$

D'autre 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 $p^* \phi(N) = Np - p^2 - pq + p$

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

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

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

2.5 Hastad Attaque

Hastad Attaque également s'appelle attaque de diffusion. Si on chiffre le même message avec le même exposant de chiffrement e à plus de e personnes différentes, RSA peut faire l'objet d'une attaque de diffusion

Par exemple : Alice envoie le même message crypté RSA m à trois destinataires, en utilisant des modules différents n_1, n_2, n_3 , ils sont premiers. Mais ils utilisent le même exposant $e=3$, Eva intercepte trois chiffrés et connaît les clés publiques des trois destinataires

À l'instant, on peut alors utiliser une attaque de Hastad pour récupérer le message m sans factoriser N .

Afin de comprendre son principe d'attaque, on rappelle le théorème des restes chinois :

Théorème : En mathématiques, le théorème des restes chinois est un résultat d'arithmétique modulaire traitant de résolution de systèmes de congruences.

Soient n_1, n_2, \dots, n_k , des entiers deux à deux premiers entre eux, c'est-à-dire que $\text{pgcd}(n_i, n_j) = 1$, lors que $i \neq j$, donc les entiers (a_1, a_2, \dots, a_k) , il existe un entier x unique module $n = \prod_{i=1}^k n_i$

$$x = a_1 \pmod{n_1}$$

.....

$$x = a_k \pmod{n_k}$$

Avec lemme chinois et algo de Gauss, Eve peut trouver la solution de x . Dans $0 \leq x < n_1 n_2 n_3$, on a

$$x = c_1 \pmod{n_1}$$

$$x = c_2 \pmod{n_2}$$

$$x = c_3 \pmod{n_3}$$

On sait que $m^3 < n_1 n_2 n_3$, donc $x = m^3$ elle prend la racine cubique de x et puis elle peut obtenir le message m .

Comment prévenir et éviter cette attaque

1. Utilisez un grand exposant e , Cela rendra difficile l'utilisation de la méthode d'attaque Hastad.

2. Ajoutez quelques bits aléatoires au message, au moins 64 bits. Assurez-vous que chaque chiffré de message ajoute un nombre aléatoire différent.

3 pollard p-1

- pour un entier n que vous souhaitez factoriser
- Si on peut trouver un entier s qui n'est pas premier avec n , on peut trouver un facteur de n directement en trouvant $\text{pgcd}(s, n)$
- Comment obtenir s ?
- si p est un facteur premier de n
 p est inconnu, et on souhaite construire un nombre s avec un facteur p , vous pouvez trouver le facteur de n en trouvant $\text{pgcd}(s, n)$.
- $x = 1 \pmod{p} \rightarrow \text{est premier avec } \text{pgcd}(x - 1, n)$
- Avec petit fermat, $2^{p-1} = 1 \pmod{p}$
- Mais p est inconnu!
- on construit un multiple de $(p-1)$
- Supposer les puissances de chaque nombre premier q , (q est premier avec $(p-1)$), on a $q \leq B$
 On peut obtenir un multiple de $(p-1)$: $B!$
- donc $x = 2^{B!} \pmod{n}$
- p est premier avec n , $x = 2^{B!} = 1 \pmod{p}$
 avec petit Fermat $(p-1)$ est premier avec $B!$
- $s = x - 1$
- $d = \text{pgcd}(s, n)$, d est un facteur non trivial de n . ($s \neq 0$)