Primeritat i factorització

Artur Travesa

(versió 2024-07)

Capítol 10: Construcció certificada de claus RSA

10.0. Introducció

En el capítol 3 hem après a construir nombres naturals primers certificables de mida prefixada. Però, per exemple, per a la construcció de claus per al criptosistema RSA, cal que, a més a més, els primers satisfacin altres condicions extres.

L'objectiu d'aquest capítol és aprendre a construir nombres primers de mida (en bits) prefixada, de certificar que ho són, de primers, i fer-ho de manera que se satisfacin algunes condicions extres que detallarem més avall, a fi que s'assemblin més als especificats als estàndards per a les claus RSA per a certificats digitals (cf. [DS-S, A.1.1]).

I com a exemple, construirem una parella de nombres primers de 2048 bits cadascun de manera que el seu producte sigui un nombre natural de 4096 bits i tals que se satisfacin algunes condicions extres necessàries per a poder formar part d'una clau RSA de 4096 bits per a signatures digitals.

Observació. El sol fet de donar-los a conèixer, ja els invalida per a ser-ho de debò, de part d'una clau; però poden servir com a exemple.

10.0.0. Funcions que aprofitarem de capítols anteriors

10.0.0.0. Tests de primeritat i certificats de composició

In [1]:

```
def SolovayStrassenTest(nn,ff):
    if nn==1:
        return false
    if nn in [2,3,5,7]:
        return true
    if is_even(nn):
        return false
    if ff<1:
        return 'Cal fer alguna prova.'
    f=0
    n2=(nn-1)//2
    while f<ff:
        g=ZZ.random_element(2,nn-1)
        x=Mod(g,nn)^n2
        if x==1 or x==nn-1:
            y=Mod(kronecker(g,nn),nn)
            if y!=x:
                return false
        else:
            return false
        f=f+1
    return 'Indeterminat'

```

In [2]:

```
def MillerRabinTest(nn,ff):
    if nn==1:
        return false
    if nn in [2,3,5,7]:
        return true
    if is_even(nn):
        return false
    if ff<1:
        return 'Cal fer alguna prova.'
    v=0
    m=nn-1
    while is_even(m):
        v=v+1
        m=m//2
    f=0
    while f<ff:
        g=ZZ.random_element(2,nn-1)
        x=Mod(g,nn)^m
        if x!=1 and x!=nn-1:
            k=1
            x=x^2
            while (x!=nn-1 and k<v-1):
                x=x^2
                k=k+1
            if k>=v-1 and x!=nn-1:
                return false
        f=f+1
    return 'Indeterminat'

```

In [3]:

```
def SolovayStrassenCert(nn,ff):
    if nn==1:
        return [nn,false,1]
    if nn==2 or nn==3:
        return [nn,true,nn-1,[nn-1]]
    if nn==5:
        return [nn,true,2,[2]]
    if nn==7:
        return [nn,true,3,[2,3]]
    if is_even(nn):
        return [nn,false,2]
    if ff<1:
        return 'Cal fer alguna prova.'
    f=0
    n2=(nn-1)//2
    while f<ff:
        g=ZZ.random_element(2,nn-1)
        x=Mod(g,nn)^n2
        if x ==1 or x==nn-1:
            y=Mod(kronecker(g,nn),nn)
            if y!=x:
                return [nn,false,g]
        else:
            return [nn,false,g]
        f=f+1
    return 'Indeterminat'

```

In [4]:

```
def MillerRabinCert(nn,ff):
    if nn==1:
        return [nn,false,1]
    if nn==2 or nn==3:
        return [nn,true,nn-1,[nn-1]]
    if nn==5:
        return [nn,true,2,[2]]
    if nn==7:
        return [nn,true,3,[2,3]]
    if is_even(nn):
        return [nn,false,2]
    if ff<1:
        return 'Cal fer alguna prova.'
    v=0
    m=nn-1
    while is_even(m):
        v=v+1
        m=m//2
    f=0
    while f<ff:
        g=ZZ.random_element(2,nn-1)
        x=Mod(g,nn)^m
        if x!=1 and x!=nn-1:
            k=1
            x=x^2
            while (x!=nn-1 and k<v-1):
                x=x^2
                k=k+1
            if k>=v-1 and x!=nn-1:
                return [nn,false,g]
        f=f+1
    return 'Indeterminat'

```

10.0.0.1. Certificats de primeritat

In [5]:

```
def Certifica(pp,fppmu,ff):
    if pp==1:
        return [pp,false,1]
    if pp==2 or pp==3:
        return [pp,true,pp-1,[pp-1]]
    if is_even(pp):
        return [pp,false,2]
    if ff<1:
        return ["Cal fer alguna prova."]
    if len(fppmu)==0:
        lta1=factor(pp-1)
        lta=[lta1[i][0] for i in range(len(lta1))]
    else:
        lta=sorted(fppmu)
    l=len(lta)
    f=0
    while f<ff:
        g=ZZ.random_element(2,pp-2)
        if (s:=Mod(g,pp)^((pp-1)//2))==pp-1:
            i=1
            while i<= l-1 and Mod(g,pp)^((pp-1)//lta[i])!=1:
                i=i+1
            if i==l:
                return [pp,true,g,lta]
        else:
            if s!=1:
                return [pp,false,g]
        f=f+1
    return [pp,'Indeterminat']

```

In [6]:

```
def Pocklington(pp,tt,ff):
    if not pp in ZZ or pp<1:
        return ['Cal que el nombre P sigui enter positiiu.']
    if pp==1:
        return [pp,false,1]
    if pp==2 or pp==3:
        return [pp,true,pp-1,[pp-1]]
    if is_even(pp):
        return [pp,false,2]
    if ff<1:
        return 'Cal fer alguna prova.'
# Comprovació que la llista tt és de divisors de pp-1, i càlcul d
# però no que són primers.
    if false in [(r in ZZ and r>1) for r in tt]:
        return 'La llista T no és de nombres enters >1.'
# Si 2 no pertany a la llista tt, li afegim (per a millora del ca
    t=tt
    if not (2 in t):
        t=[2]+t
    x=prod(t)
    q,r=divmod(pp-1,x)
    if r:
        return 'La llista T no és correcta.'
    d=gcd(q,x)
    while d>1:
        q=q//d
        d=gcd(q,x)
    uu=q
    q=uu^2
    if q==pp:
        return [pp,false,uu]
    if q>pp:
        return 'U és massa gran.'
    t=sorted(t)
# Si hem arribat aquí, és que P, T, F i U són correctes (excepte
# potser, que alguns elements de T no siguin primers).
    l=len(t)
    f=0
    while f<ff:
        g=ZZ.random_element(2,pp-2)
        if (s:=Mod(g,pp)^((pp-1)//2))==pp-1:
            i=1
            while i<= l-1 and gcd((s:=Mod(g,pp)^((pp-1)//t[i]))-1
                i=i+1
            if i==l:
                return [pp,true,g,t]
        else:
            if s!=1:
                return [pp,false,g]
        f=f+1
    return [pp,'Indeterminat']

```

10.1. Restriccions per a les claus RSA per a signatura digital

D'acord amb els estàndards per a les claus RSA per a signatures digitals (cf. [DS-S,

A.1.1]), les longituds dels mòduls n de les claus, en bits, han de ser nombres parells, posem $2L$, i com a mínim de 2048 bits (o sigui, $2L \geq 2048$).

A més a més, el mòdul n és el producte de dos nombres naturals primers diferents, p, q, tals que

$p-1$ és divisible per un nombre primer p_1,

$p+1$ és divisible per un nombre primer p_2,

$q-1$ és divisible per un nombre primer q_1,

$q+1$ és divisible per un nombre primer q_2,

i de manera que les longituds (en bits) dels primers p_i, q_i, satisfacin les restriccions següents, en cas que siguin certificables (o amb la darrera condició menys restrictiva si els primers no se certifiquen):

$len(n) = 2L$	len(p_i), len(q_i)	len(p_1)+len(p_2), len(q_1)+len(q_2) (si els primers se certifiquen)	len(p_1)+len(p_2), len(q_1)+len(q_2) (si els primers no se certifiquen)
$2048 \leq 2L < 3072$	> 140 bits	≤ 494 bits	≤ 1007 bits
$3072 \leq 2L < 4096$	> 170 bits	≤ 750 bits	≤ 1518 bits
$4096 \leq 2L$	> 200 bits	≤ 1005 bits	≤ 2030 bits

A més a més,

cadascun dels primers p i q ha de ser de L bits; de fet, p i q han de pertànyer a l'interval $2^{L-1}\sqrt{2} \leq p, q \leq 2^L - 1$, i

el valor absolut, $|p-q|$, de la seva diferència ha de ser com a mínim 2^{L-100}.

Observació. Encara que l'estàndard no ho especifica, les restriccions anteriors sobre la divisibilitat per primers p_i, q_i, són per a (intentar) evitar els mètodes de factorització $p-1$ de Pollard, i $p+1$, de Williams; la restricció sobre el valor absolut de la diferència és per a (intentar) evitar el mètode de factorització de Fermat, i la restricció sobre la mida dels primers p_i, q_i, i encara més, sobre l'interval al qual han de pertànyer, permet assegurar que el seu producte, n, és de $2L$ bits.

10.2. Primers de 112 bits

L'objectiu final és construir un parell de nombres primers p, q, de 2048 bits cadascun, de la forma

$$p = 2m_p p_0 p_1 + 1, \qquad q = 2m_q q_0 q_1 + 1,$$

de manera que p_0, p_1, q_0, q_1, siguin primers, que p_1, q_1 siguin de més de 200 bits cadascun (o sigui, $p_1, q_1 > 2^{199}$), que $p < (p_0 p_1)^2$, $q < (q_0 q_1)^2$, i que m_p sigui divisible per un nombre primer p_2 de més de 200 bits tal que $len(p_1) + len(p_2) \leq 1005$ bits, i que m_q sigui divisible per un nombre primer q_2 de més de 200 bits tal que $len(q_1) + len(q_2) \leq 1005$ bits, i que se satisfacin les altres restriccions per a les longituds i la diferència entre p i q.

Com que hi ha una restricció per a la diferència entre p i q, serà convenient construir-los simultàniament, imposant aquesta condició en la construcció.

D'altra banda, la construcció dels nombres m_p i m_q ha de ser tal que es garanteixi la condició que $p+1$ i $q+1$ siguin divisibles per primers p_2, q_2, respectivament, de la longitud predeterminada.

10.2.0. Construcció d'una primera llista de nombres primers

Es tracta de construir una llista, que anomenarem lp112, de 32 nombres primers de 112 bits i certificar-los, però sense emmagatzemar, ni escriure, els certificats. (En farem servir 24, d'aquests nombres, però en tenim més per si algun presentés problemes.)

Observació. Els nombres d'aquesta llista són prou petits perquè puguin ésser certificats fàcilment amb la funció Certifica; per això no emmagatzemarem els certificats.

Per a construir aquesta llista, definirem una funció Primer112( ) que calculi un nombre primer d'aquesta mida a partir d'una cercar a l'atzar entre els nombres senars de 112 bits. Un cop feta la tria, li passarem el test de Solovay-Strassen, de manera que si el nombre triat és compost, no passarà el test i en triarem un altre. Això ho farem successivament un màxim de, posem, 500 vegades. Un cop en tinguem un que passi el test (que, per tant, deu ser primer), el certificarem com a primer i el retornarem. Si no és primer o si hem arribat al final sense trobar-ne cap en el nombre màxim fixat de tries, retornarem 0.

Després, aplicarem aquesta funció per a fer una llista de 32 nombres.

In [7]:
```
def Primer112():
    fita=500
    n=0
    while not SolovayStrassenCert((q:=(2*ZZ.random_element(2^110
        n=n+1
    if n<fita and Certifica(q,[],50)[1]:
        return q
    return 0

```

In [8]:
```
lp112=[Primer112() for i in range(32)]
```

Encara que no cal, podem veure'n certificats (que fem de nou); notem que, com que no coneixem prou primers que divideixin p-1, és més útil la funció Certifica que la funció Pocklington.

In [9]:
```
1 [Certifica(lp112[i],[],50) for i in range(len(lp112))]
```

Out[9]:
```
[[4926282733217301591518313572336131,
  True,
  311050894807554827263067467236032 1,
  [2, 3, 5, 32119, 17041774281118830984492784 03]],
 [2848340922237610515858970910850853,
  True,
  2439746627857008411850320632926437,
  [2, 3, 7, 593, 57181822095832540670098990421]],
 [3265043531133321766370921465502469,
  True,
  3772908935807860119733497276464 32,
  [2, 3, 13, 17, 101, 256931, 1712371, 27663149, 333854197]],
 [4164256990730443618450953899909417,
  True,
  5902922243009957327121372629678 26,
  [2, 1870257178117, 2783211474506325236 81]],
 [4036064878503773287392481507784849,
  True,
  2254609206109395383927444839855789,
```

10.3. Primers de 480 bits

Notem que el producte de quatre nombres de 112 bits és un nombre de (aproximadament) 448 bits. Sigui $np1$ el producte de quatre d'aquests nombres de $lp112$; concretament, prendrem el primer, el quadrat del segon i el tercer; o sigui, farem

$$np1 := lp112[0] \cdot lp112[1]^2 \cdot lp112[2].$$

Anàlogament, considerarem

$$np2 := lp112[4] \cdot lp112[5]^2 \cdot lp112[6],$$
$$nq1 := lp112[8] \cdot lp112[9]^2 \cdot lp112[10],$$
$$nq2 := lp112[12] \cdot lp112[13]^2 \cdot lp112[14].$$

Farem servir aquests nombres per a la construcció dels primers p_1, p_2, q_1 i q_2.

I també considerarem

$$np00 := lp112[16] \cdot lp112[17]^2 \cdot lp112[18],$$
$$nq00 := lp112[20] \cdot lp112[21]^2 \cdot lp112[22],$$
$$np01 := lp112[24] \cdot lp112[25]^2 \cdot lp112[26],$$
$$nq01 := lp112[28] \cdot lp112[29]^2 \cdot lp112[30],$$

nombres que farem servir com a auxiliars per a la construcció de p_0 i de q_0.

In [10]:
```
1 [np1,np2,nq1,nq2,np00,nq00,np01,nq01]=[lp112[4*i]*lp112[4*i+1]^2
```

Tot i que no cal, mirem-nos-els.

In [11]:
```
1 [np1,np2,nq1,nq2,np00,nq00,np01,nq01]
```

Out[11]:
```
[13049451221822828079264937326288765106437006100712547409683245668640282000111593227548944156667600368278293332382650121543005469218335
1,
 21383915132753820882225225650672222713063599638020565238560776402599481436826100601576687470538234230749607619274956255237575428153349
3,
 13003775296546139860970148806381708790458977378399623611274671756443833821829938808370848054752619974293788456866936299589862270889258
7,
 15846764350010484781402157061558955038192382206122014340858472969797745063917092937522976200797457840022295525301747414008551729237133
1,
 13551525781697463121368839633950603933705123259280792466407850978863910752273876687111534667115323296095755476695843122722553600515076
9,
 21316756986518791556832999019395250453733176137783133550183638599497426332683313024273983475996089598587874277243800548269386862409259,
 99786854351728531109639826143960641430572910726823475743367454663275140465502105597885641864927146752277014772521167453495894592152659,
 39603038528565027278926962147497253947037861711431203643388587282366384376951305658518777674427201598557989490515706006809286393640504041]
```

Òbviament, aquests nombres no són ni de 480 bits, ni (probablement) primers; però a partir d'ells podem calcular fàcilment nombres primers de 480 bits de la manera següent. Ens fixem en $np1$; per als altres, es fa el mateix.

Per a tot nombre natural $kp1$, el nombre $p_1 := 2 * kp1 * np1 + 1$ és senar; i el seu nombre de bits depèn d'un interval on es pot triar $kp1$; de fet, si volem que p_1 sigui de 480 bits, cal que

$$2^{479} \leq p_1 = 2 * kp1 * np1 + 1 < 2^{480}$$

i, per tant, hem de prendre $kp1$ com un nombre enter de l'interval

$$\frac{2^{479} - 1}{2 \cdot np1} \leq kp1 < \frac{2^{480} - 1}{2 \cdot np1}.$$

Notem que la longitud de l'interval és

$$\frac{2^{480} - 1}{2 \cdot np1} - \frac{2^{479} - 1}{2 \cdot np1} = \frac{2^{480} - 2^{479}}{2 \cdot np1} = \frac{2^{478}}{np1} \geq \frac{2^{478}}{2^{4 \cdot 112}} = 2^{30},$$

de manera que hi ha molts valors de $kp1$ per poder triar a l'atzar.

I, anàlogament, per a les parelles $(kp2, p_2)$, $(kq1, q_1)$, i $(kq2, q_2)$.

Ara, la tria dels valors de $kp1$, $kp2$, $kq1$, $kq2$ en els intervals adequats produeix nombres p_1, p_2, q_1 i q_2 de la quantitat desitjada de bits; però (segurament) no són primers!

Calculem les fites per als intervals.

In [12]:
```
infkp1=ceil((2^479-1)/(2*np1))
supkp1=floor((2^480-1)/(2*np1))
infkp2=ceil((2^479-1)/(2*np2))
supkp2=floor((2^480-1)/(2*np2))
infkq1=ceil((2^479-1)/(2*nq1))
supkq1=floor((2^480-1)/(2*nq1))
infkq2=ceil((2^479-1)/(2*nq2))
supkq2=floor((2^480-1)/(2*nq2))
```

Triem valors aleatoris.

In [13]:
```
kp1=ZZ.random_element(infkp1,supkp1)
kp2=ZZ.random_element(infkp2,supkp2)
kq1=ZZ.random_element(infkq1,supkq1)
kq2=ZZ.random_element(infkq2,supkq2)
```

Encara que no cal, els podem veure.

In [14]:
```
kp1, kp2, kq1, kq2
```

Out[14]: (7871743302, 3985833862, 9926574322, 8474742605)

Amb aquests valors de $kp1$, $kp2$, $kq1$ i $kq2$, definim nombres p_1, p_2, q_1, q_2, als quals aplicarem el test de primeritat de Solovay-Strassen a fi de comprovar que no són primers, o de tenir gran certesa que ho són.

I repetirem la tria a l'atzar dels valors $kp1$, $kp2$, $kq1$ i $kq2$, fins que el test no detecti que el nombre és compost. En aquest cas, podrem certificar-lo com a primer amb el certificat de Pocklington, perquè els productes de primers que hem pres per a $np1$, $np2$, $nq1$ i $nq2$ són més grans que l'arrel quadrada del primer corresponent que volem certificar (de fet, amb el primer del qual hem considerat el quadrat i un els altres dos, ja en tenim prou).

Això produirà, doncs, els nombres primers p_1, p_2, q_1 i q_2 que volem, amb certificats (que, si no cal, podem obviar).

Per a aquesta tasca, definim una funció Primer480(ltanp) que automatitzi aquest procés i es pugui aplicar de manera senzilla a cadascun dels casos. Aquí, ltanp és (una part significativa de) la llista de primers que divideixen $p - 1 = 2 \cdot k \cdot np$; i la funció retorna p, el valor k i un certificat de primeritat de p.

Observació. La funció està adaptada a la construcció que fem dels nombres np i no és general; per exemple, construeix el nombre np tal com ho hem fet a partir de la llista, però no cap altre nombre construït a partir dels seus elements.

In [15]:
```
def Primer480(ltanp):
    np=ltanp[0]*ltanp[1]^2*ltanp[2]
    infkp=ceil((2^479-1)/(2*np))
    supkp=floor((2^480-1)/(2*np))
    kp=ZZ.random_element(infkp,supkp)
    p=2*kp*np+1
    fita=1000
    while fita >0 and SolovayStrassenTest(p,50)==false:
        kp=ZZ.random_element(infkp,supkp)
        p=2*kp*np+1
        fita=fita-1
    if fita==0:
        return 0, 'Cal tornar-hi!'
    else:
        cert=Pocklington(p,ltanp[0:2],50)
    return [p,kp,cert]
```

10.3.1. Construcció dels nombres primers p1, p2, q1 i q2, de 480 bits

Apliquem la funció per a construir p_1, p_2, q_1 i q_2, amb els seus respectius certificats.

In [16]:
```
[p1,kp1,certp1]=Primer480(lp112[0:3])
```

In [17]:
```
[q1,kq1,certq1]=Primer480(lp112[4:7])
```

In [18]:
```
[p2,kp2,certp2]=Primer480(lp112[8:11])
```

In [19]:
```
[q2,kq2,certq2]=Primer480(lp112[12:15])
```

Tot i que no cal, els podem veure, així com també que se satisfan les restriccions de la seva mida.

In [20]:
```
p1,p2,q1,q2
```

Out[20]:
```
(232081754371461561782985341722174228198997830685835738907052842679
244915635107874158223307830973933694093843264680991707166772723639 5
556370396467,
 232418220695730897703073734869086823896795388888803151709009121915
827578018301041819541596866243867397595791881530317974348414736143 5
041240538487,
 163503535747899429661910965600667592028410737349481344313490404446
471480856546455244052010759831792205000163295563919024958333634015 7
187154500811,
 278907012381353841472636110617248465146988780416597882580296646665
382952125072897431421795250374265654752381696561207937797817719496 7
477932932943)
```

In [21]:
```
2^479<p1<2^480
```

Out[21]:
```
True
```

In [22]:
```
2^479<p2<2^480
```

Out[22]: True

In [23]:
```
2^479<q1<2^480
```

Out[23]: True

In [24]:
```
2^479<q2<2^480
```

Out[24]: True

10.4. Primers de 1024 bits

10.4.0. Construcció dels primers p0 i q0

Anàlogament a la construcció dels nombres primers p_1, p_2, q_1 i q_2 anteriors, podem construir nombres primers p'_0, p'_1, q'_0, q'_1, també de 480 bits, a fi de construir $p_0 := 2 \cdot kp_0 \cdot p'_0 \cdot p'_1$, i $q_0 := 2 \cdot kq_0 \cdot q'_0 \cdot q'_1$, de 1024 bits a partir del càlcul dels valors kp_0 i kq_0, a l'atzar, com més amunt (en l'interval adequat).

Repetim, doncs, el procediment. Només cal aplicar la funció Primer480( ) a la llista adequada en cada cas.

In [25]:
```
[pp0,kpp0,certpp0]=Primer480(lp112[16:19])
```

In [26]:
```
[pq0,kpq0,certpq0]=Primer480(lp112[20:23])
```

In [27]:
```
[pp1,kpp1,certpp1]=Primer480(lp112[24:27])
```

In [28]:
```
[pq1,kpq1,certpq1]=Primer480(lp112[28:31])
```

Per a la construcció de p_0, q_0, serà útil repetir la funció Primer480 però per a 1024 bits i a partir de dos primers de 480 bits.

In [29]:
```
def Primer1024(ltanp):
    np=ltanp[0]*ltanp[1]
    infkp=ceil((2^1023-1)/(2*np))
    supkp=floor((2^1024-1)/(2*np))
    kp=ZZ.random_element(infkp,supkp)
    p=2*kp*np+1
    fita=2000
    while fita >0 and SolovayStrassenTest(p,50)==false:
        kp=ZZ.random_element(infkp,supkp)
        p=2*kp*np+1
        fita=fita-1
    if fita==0:
        return 0, 'Cal tornar-hi!'
    else:
        cert=Pocklington(p,ltanp,50)
    return [p,kp,cert]
```

In [30]:
```
[p0,kp0,certp0]=Primer1024([pp0,pp1])
```

In [31]:
```
[q0,kq0,certq0]=Primer1024([pq0,pq1])
```

10.5. Primers de 2048 bits

Notem que fins ara hem construït nombres primers prou grans i de manera que si escrivim $p = 2m_p \cdot p_0 \cdot p_1$, i anàlogament per a q, els nombres $p-1$ i $q-1$ són divisibles per un primer de 480 bits.

Resta trobar els valors de m_p i m_q de manera que p i q siguin primers, de la mida volguda, per als quals se satisfaci la restricció sobre la seva diferència, i tals que $p+1$ i $q+1$ siguin divisibles per algun primer de 480 bits (els p_2, q_2 que hem calculat més amunt).

Considerem els nombres enters y_p, y_q, dels intervals $1 \leq y_p \leq p_0p_1 - 1$, $1 \leq y_q \leq q_0q_1 - 1$, inversos de $p_0p_1 \pmod{p_2}$ i $q_0q_1 \pmod{q_2}$, respectivament; és a dir, tals que $y_pp_0p_1 \equiv 1 \pmod{p_2}$ i $y_qq_0q_1 \equiv 1 \pmod{q_2}$.

Ara es tracta de trobar nombres naturals t_p i t_q adequats, de manera que per a $m_p := t_pp_2 - y_p$, $m_q := t_qq_2 - y_q$, els nombres $p := 2m_pp_0p_1 + 1$, $q := 2m_qq_0q_1 + 1$, satisfacin les condicions volgudes.

Notem que per a tots els nombres de la forma $m_p := t_pp_2 - y_p$, tenim que
$$p + 1 = 2m_pp_0p_1 + 2 = 2(t_pp_2 - y_p)p_0p_1 + 2 \equiv -2y_pp_0p_1 + 2 \equiv 0 \pmod{p_2};$$
i anàlogament per a $q+1$:
$$q + 1 = 2m_qq_0q_1 + 2 = 2(t_qq_2 - y_q)q_0q_1 + 2 \equiv -2y_qq_0q_1 + 2 \equiv 0 \pmod{q_2};$$
és a dir, $p+1$ i $q+1$ són divisibles per p_2, q_2, respectivament.

Així, només cal construir els valors t_p i t_q de manera que se satisfacin les altres condicions (mida dels p i q, valor absolut de la diferència, i que siguin primers).

10.5.0. Construcció de yp i yq

In [32]:
```
[yp,yq]=[Integer(Mod(p0*p1,p2)^(-1)),Integer(Mod(q0*q1,q2)^(-1))
```

In [33]:
```
[yp,yq]
```

Out[33]:
```
[48420549999768651963622640506468767333190354153882106893246351699979659279700488002214087435653184623668866542168237566783997437869034070635388 1,
 32985980291428067640466904398679730360991235615291233736476553768162390275918298347320769540445583764672643692330079575924848942577828083588737 7]
```

10.5.1. Construcció de p

De manera similar com hem calculat valors $kp1$, $kp2$, $kq1$, $kq2$, i altres, ara calcularem els valors t_p i t_q.

I a fi de poder calcular els nombres p i q d'acord amb la restricció que el valor absolut de la diferència sigui més gran que $2^{2048-100} = 2^{1948}$, primerament calcularem p i, després podrem calcular q en tenir en compte el valor de p.

Per comoditat de lectura (i també de càlcul), posarem $a := 2^{2047}\sqrt{2}$ i $b := 2^{2048}$.

Notem que volem que se satisfacin les condicions sobre la mida de p i de q:

$$a < p = 2(t_p p_2 - y_p)p_0 p_1 + 1 < b,$$
$$a < q = 2(t_q q_2 - y_q)q_0 q_1 + 1 < b;$$

per tant, ha de ser

$$\frac{a-1}{2p_0p_1} + y_p < t_p p_2 < \frac{b-1}{2p_0p_1} + y_p;$$

o sigui,

$$\frac{a-1+2p_0p_1y_p}{2p_0p_1} < t_p p_2 < \frac{b-1+2p_0p_1y_p}{2p_0p_1};$$

és a dir,

$$\frac{a-1+2p_0p_1y_p}{2p_0p_1p_2} < t_p < \frac{b-1+2p_0p_1y_p}{2p_0p_1p_2}.$$

I, anàlogament,

$$\frac{a-1+2q_0q_1y_q}{2q_0q_1q_2} < t_q < \frac{b-1+2q_0q_1y_q}{2q_0q_1q_2}.$$

Observació. Més avall canviarem aquestes fites per a t_q.

Ara, en tenir en compte les mides dels primers p_0, p_1, p_2, notem que això permet triar t_p en un interval de longitud

$$\frac{b-1+2p_0p_1y_p}{2p_0p_1p_2} - \frac{a-1+2p_0p_1y_p}{2p_0p_1p_2} = \frac{b-a}{2p_0p_1p_2} = \frac{2^{2046}(2-\sqrt{2})}{p_0p_1p_2}$$
$$> 2^{2046-1024-480-480}(2-\sqrt{2}) = 2^{62}(2-\sqrt{2}) > 2^{61}.$$

És a dir, podem triar t_p a l'atzar en un interval prou gran i esperar que alguna tria proporcioni un nombre primer de la mida volguda.

Comencem per definir un parell de nombres, sq(= a), i sq2(= r) per a comparar i calcular

les fites dels intervals.

In [34]:
```
sq=2^2047*sqrt(2)
```

In [35]:
```
sq2=(2^2048+sq)/2
```

Escrivim les fites de l'interval per al càlcul de p.

In [36]:
```
inftp=ceil((sq-1+2*p0*p1*yp)/(2*p0*p1*p2))
suptp=floor((2^2048-1+2*p0*p1*yp)/(2*p0*p1*p2))
```

Calculem, successivament, t_p, m_p, i p.

In [37]:
```
tp=ZZ.random_element(inftp,suptp)
```

In [38]:
```
mp=tp*p2-yp
```

In [39]:
```
p=2*mp*p0*p1+1
```

Tot i que no cal, els podem veure.

In [40]:
```
mp
```

Out[40]:
```
70525041791230960465640787609975168332258136956696841251423698645893944120615989218450281002018969426263439645143173734291823484249186964348090070130088448734126966
```

In [41]:
```
p
```

Out[41]:
```
30003052846086477380543156356675274723518778689593895411413388285079821666012008970259517026973828447345021733459531498410336064418896204461926438733455922908142230293075466886283103092541364962311024418760553736892719807650735887654765982386947343646902166614710978181271946153052533109214409562178724347834361195157715886863777470334015890664599700078316236495589648754058844507669073109472025693414184930703943856887696546728594018790888504627615804841520937571342913833045210642262723212830432781032939487013168022881700280185744008164167309645161416576529927590008849287803887487271774997050981909335712573315 09
```

In [42]:
```
SolovayStrassenTest(p,50)
```

Out[42]:
```
False
```

Com era d'esperar, p no és primer; cal repetir el càlcul de p; ho automatitzem.

In [43]:
```
fita=2000
```

In [44]:
```
while SolovayStrassenTest(p,50)==false and fita>0:
    tp=ZZ.random_element(inftp,suptp)
    mp=tp*p2-yp
    p=2*mp*p0*p1+1
    fita=fita-1
if fita==0:
    primerp=0
    print('Cal tornar-hi!')
else:
    primerp=[p,Pocklington(p,[p0],100)]

```

Tot i que no cal, el podem veure, amb un certificat.

In [45]:
```
primerp
```

Out[45]:
```
[29400638643435206084507793777103461295382084302981661859971123074766441685011995556555124128990673450088382929028831163259761414320181980084932176579612282953531588011298961094993183775166716957210791889057926062753819429066463024106581122200456975238221925357396936902221759773243169402657955358200983618923906462420416014512272124698830310426416828359503179832579759733094590196875602063074151386056508300351275232420182244643849880197979496704649268570194990411860534757406446917912207000898195823235489480130580317935839392925517481396349865617508343609254787144282553943655848564406603091909714782265955085194721,
 [29400638643435206084507793777103461295382084302981661859971123074766441685011995556555124128990673450088382929028831163259761414320181980084932176579612282953531588011298961094993183775166716957210791889057926062753819429066463024106581122200456975238221925357396936902221759773243169402657955358200983618923906462420416014512272124698830310426416828359503179832579759733094590196875602063074151386056508300351275232420182244643849880197979496704649268570194990411860534757406446917912207000898195823235489480130580317935839392925517481396349865617508343609254787144282553943655848564406603091909714782265955085194721,
  True,
  787360055988506028922703992491349127629974964800732747424648944013557089380864787098409754259065932165848728790914776617830919995286075703952076091914025951962637621855847447671543573197071330339216113974624311964716525493785946637413093624037840875215784557606735283001148013255455291303160354394620503234172347749892005734115052709633526406502097135892487682442135971964263603623693328980027673722406619581670032263843405778131795299033394265860385710899144632257476372846705577994835310548194710180098941084159628164683585504885170008930882831501963914943296857971330988186555098947050336789960541413267401184278,
  [2,
   9165393182906098189233458295210239199691597782873375508829057159432218042260057224339416162434720684721807599626585780008561108073344697702861722734314912729561943473341250155242934557499410531960067539296308870891039031919903111080499651260852232347205665690706980542461243999371352771027244823470527801895 7]]]
```

I (òbviament) se satisfan les condicions volgudes per a p.

```
In [46]:  1  (p-1)%p1
Out[46]: 0

In [47]:  1  (p+1)%p2
Out[47]: 0

In [48]:  1  sq<p<2^2048
Out[48]: True
```

10.5.2. Construcció de q

Per a la tria del valor de q, cal triar t_q de manera similar a t_p. Però ara, podem recalcular l'interval a fi que se satisfaci la propietat que $|p - q| \geq 2^{1948}$.

Notem que podem bipartir l'interval $[a := 2^{2047}\sqrt{2}, b := 2^{2048}]$ pel seu punt mitjà,

$$r := \frac{a+b}{2} = \frac{2^{2047}\sqrt{2} + 2^{2048}}{2} = 2^{2046}(2+\sqrt{2}),$$

mirar en quina de les dues meitats hi ha p, i imposar que q estigui a l'altra (i prou separat de p).

Més concretament, volem imposar que se satisfaci que si $a < p < r$, llavors sigui $r + 2^{1948} < q < b$, i que si $r < p < b$, llavors sigui $a < q < r - 2^{1948}$.

Per tant, només cal adaptar adequadament els intervals de cerca de t_q.

Observació. És (molt) probable que una tria a l'atzar de t_p i t_q en l'interval predit ja faci que se satisfaci la condició sobre el valor absolut de la diferència entre p i q. I en cas (molt poc probable) que no se satisfés, podríem tornar a calcular un dels primers (o tots dos). Amb el procediment que hem descrit, ens assegurem que això succeeixi. A canvi, però, perdem la propietat que tots dos nombres siguin "prou" aleatoris, perquè imposem que estiguin en meitats complementàries de l'interval $[a, b]$.

Així, les noves fites són:

En el cas que $a < p < r$, cal que

$$\frac{a + 2^{1948} - 1 + 2q_0q_1y_q}{2q_0q_1q_2} < t_q < \frac{b - 1 + 2q_0q_1y_q}{2q_0q_1q_2};$$

i en el cas que $r < p < b$, cal que

$$\frac{a - 1 + 2q_0q_1y_q}{2q_0q_1q_2} < t_q < \frac{2^{2046}(\sqrt{2}+1) - 1 + 2q_0q_1y_q - 2^{1948}}{2q_0q_1q_2};$$

Calculem-les.

In [49]:
```
if p<sq2:
    inftq=ceil((sq+2^1948-1+2*q0*q1*yq)/(2*q0*q1*q2))
    suptq=floor((2^2048-1+2*q0*q1*yq)/(2*q0*q1*q2))
else:
    inftq=ceil((sq-1+2*q0*q1*yq)/(2*q0*q1*q2))
    suptq=floor((2^2048-1+2*q0*q1*yq-2^1948)/(2*q0*q1*q2))
```

Encara que no cal, les podem veure.

In [50]:
```
inftq,suptq
```

Out[50]:
```
(20726225375333666824, 29311309022598263965)
```

Calculem, successivament, t_q, m_q, i q.

In [51]:
```
tq=ZZ.random_element(inftq,suptq)
```

In [52]:
```
mq=tq*q2-yq
```

In [53]:
```
q=2*mq*q0*q1+1
```

Encara que no cal, els podem veure.

In [54]:
```
tq
```

Out[54]:
```
27405407247517551763
```

In [55]:
```
mq
```

Out[55]:
```
76435602584994221107035173467933919976058663000039116574357556529094926250069072978446882373615893028181895793964505437084297794639679604896046379563312680574541132
```

In [56]:
```
q
```

Out[56]:
```
30215665623188436043350014209458195345685622766598833484656235497912063414199585573329723227029379817887598258436674420878862610181067605739118145569427081139495582669931656738819757398822732242032449452355598920793737021685177057233368770535553167200751111182331430358524955560947909959415109999301718170985977260023350303248766276760408618516015926941770280534034373910994208185964991499318439590361728157388336006516325281739514801968213947385191191618922319813565290074614595053575399231326371761122814753890896851552092985821816140894875386614161856591667205832724648719673546157995894407363841307844063174705049
```

In [57]:
```
SolovayStrassenTest(q,50)
```

Out[57]:
```
False
```

Com era d'esperar, q no és primer; cal repetir el càlcul de q; ho automatitzem.

In [58]:
```
fita=2000
```

In [59]:
```
while SolovayStrassenTest(q,50)==false and fita>0:
    tq=ZZ.random_element(inftq,suptq)
    mq=tq*q2-yq
    q=2*mq*q0*q1+1
    fita=fita-1
if fita==0:
    primerq=0
    print('Cal tornar-hi!')
else:
    primerq=[q,Pocklington(q,[q0],100)]

```

In [60]:
```
primerq
```

Out[60]:
```
[22999992224019330879405103047317882376951148948021019343591690350
50248478958032307849782569104989243380301365271999171703397858943624
33534833613441225852967850461486534272631739980477413036745665841358
79073839515528099635688012558550058049496957507738618831356368109278
18090634956260748817736793152874789313985231912069808476902323229633
17708884581462366985286551825291427567452493692333889238830844245808
45423497252933220262674504036118569264051143550671396974037721386069
41981736879239346171697737230566488472016496382233125897659507623000
56700103735178016314827265412478938153112107113457179897920856466450
5132207,
 [22999992224019330879405103047317882376951148948021019343591690350450
24847895803230784978256910498924338030136527199917170339785894362433
53483361344122585296785046148653427263173998047741303674566584135879
07383951552809963568801255855005804949695750773861883135636810927818
09063495626074881773679315287478931398523191206980847690232322963317
70888458146236698528655182529142756745249369233388923883084424580845
42349725293322026267450403611856926405114355067139697403772138606941
98173687923934617169773723056648847201649638223312589765950762300056
70010373517801631482726541247893815311210711345717989792085646645051
32207,
  True,
  13426398065142319098865038629620778627520959950662972764801085863366
65804797654409906626457577221011060239589412491532675852278985593013
07174493311597999095094415623320721157767658453082458560983851982921
53838496043719941809779554739678054587134326937434394936623645038067
66636956476314679143360111220675212653089586156236497961897829850964
91831206531117405274180508492047124137902001808259872271373372202541
90899694096482941044030293788514947360802181712314975204999851987922
27323711180247079049498644974173474472093635740580615882552123192434
48053213816901253236880810358234605360570568007308902703616572840290
8194,
  [2,
   12088692421152671559884058783392804300877858166105154631965891918225
24493643024419371183495569043082417222368346789841220606307313784853
61268409805111217573007496632872126609207956861941475248732124781799
43499775233557651015766862502101367461188641217271824821570380446306
48288034525577664250660071254197635871]]]
```

I se satisfan les condicions volgudes per a q.

In [61]:
```
1 q=primerq[0]
```

In [62]:
```
1 q
```

Out[62]:
```
22999992224019330879405103047317882376951148948021019343591690350450248478958032307849782569104989243380301365271999171703397858943624335348336134412258529678504614865342726317399804774130367456658413587907383951552809963568801255855005804949695750773861883135636810927818090634956260748817736793152874789313985231912069808476902323229633177088845814623669852865518252914275674524936923338892388308442458084542349725293322026267450403611856926405114355067139697403772138606941981736879239346171697737230566488472016496382233125897659507623000567001037351780163148272654124789381531121071134571798979208564664505132207
```

In [63]:
```
1 (q-1)%q1
```

Out[63]:
```
0
```

In [64]:
```
1 (q+1)%q2
```

Out[64]:
```
0
```

In [65]:
```
1 sq<q<2^2048
```

Out[65]:
```
True
```

In [66]:
```
1 abs(p-q)>2^1948
```

Out[66]:
```
True
```

In [67]:
```
1 2^4095<p*q<2^4096
```

Out[67]:
```
True
```

In [68]:
```
1 n=p*q
```

In [69]:
```
1 n
```

Out[69]:
```
676214460180211988791566002161482376401099079441667667904655738109812138407641460974019079372818484728725400842183419326284499603619696840113254666975329695874249696469138745913591790377356436892924653247688918015060775266464869544314309178877354650131537925530317571934919628438509644149222987553081277629636215884513557436807376891144626414259423089267591792069491690281664564481032893819609818090350173092489455345222662900753008957475549872099183217378918879961615028012820727173275148073774009932997662988779101414054426129705487939224706838156379141506674231849056212022405134716713979590238998339596025248962954159661597906457525093233529055273770426489479060223109639548242120379174533820596614027705228571940418209598120253460691012961376915204638217432434939663388118421990178723293221119107842478896497182400216187965781021419486152317565806727158425733868571589928319435844161162712316696612821101631042672561493581050004791398216349925543124125741075809841585805429582556349177941365734609949648120195209980606930181653040002526878401266995087092187331020253591024297006865878160227785077610460644644034860770525405019799027594373663903789926134757291081635665476108973751146607578322753546323637679113713086026443479247
```

10.6. Els valors obtinguts per a una possible clau

In [70]:
```
1 p
```

Out[70]:
```
294006386434352060845077937771034612953820843029816618599711230747664416850119955565551241289906734500883829290288311632597614143201819800849321765796122829535315880112989610949931837751667169572107918890579260627538194290664630241065811222004569752382219253573969369022217597732431694026579553582009836189239064624204160145122721246988303104264168283595031798325797597330945901968756020630741513860565083003512752324201822446438498801979794967046492685701949904118605347574064469179122070008981958232354894801305803179358393929255174813963498656175083436092547871442825539436558485644066030919097147822659550851947 21
```

In [71]:
```
1 q
```

Out[71]:
```
229999922240193308794051030473178823769511489480210193435916903504502484789580323078497825691049892433803013652719991717033978589436243353483361344122585296785046148653427263173998047741303674566584135879073839515528099635688012558550058049496957507738618831356368109278180906349562607488177367931528747893139852319120698084769023232296331770888458146236698528655182529142756745249369233388923883084424580845423497252933220262674504036118569264051143550671396974037721386069419817368792393461716977372305664884720164963822331258976595076230005670010373517801631482726541247893815311210711345717989792085646645051322 07
```

In [72]:
```
1 p0
```

Out[72]:
```
916539318290609818923345829521023919969159778287337550882905715943221804226005722433941616243472068472180759962658578000856110807334469770286172273431491272956194347334125015524293455749941053196006753929630887089103903191990311108049965126085223234720566569070698054246124399937135277102724482347052780189 57
```

In [73]:
```
1 p1
```

Out[73]:
```
232081754371461561782985341722174228198997830685835738907052842679244915635107874158223307830973933694093843264680991707166772723639555637039646 7
```

In [74]:
```
1 p2
```

Out[74]:
```
232418220695730897703073734869086823896795388888803151709009121915827578018301041819541596866243867397595791881530317974348414736143504124053848 7
```

In [75]:
```
1 q0
```

Out[75]:
```
120886924211526715598840587833928043008778581661051546319658919182252449364302441937118349556904308241722236834678984122060630731378485361268409805111217573007496632872126609207956861941475248732124781799434997752335576510157668625021013674611886412172718248215703804463064828803452557766425066007125419763587
```

In [76]:
```
1 q1
```

Out[76]:
```
163503535747899429661910965600667592028410737349481344313490404446471480856546455244052010759831792205000163295563919024958333634015718715450081 1
```

In [77]:
```
1 q2
```

Out[77]:
```
278907012381353841472636110617248465146988780416597882580296646665382952125072897431421795250374265654752381696561207937797817719496747793293294 3
```

In [78]:
```
1 n
```

Out[78]:
```
676214460180211988791566002161482376401099079441667667904655738109812138407641460974019079372818484728725400842183419326284499603619696840113254666975329695874249696469138745913591790377356436892924653247688918015060775266464869544314309178877354650131537925530317571934919628438509644149222987553081277629636215884513557436807376891144626414259423089267591792069491690281664564481032893819609818090350173092489455345222662900753008957475549872099183217378918879961615028012820727173275148073774009932997662988779101414054426129705487939224706838156379141506674231849056212022405134716713979590238998339596025248962954159661597906457525093233529055273770426489479060223109639548242120379174533820596614027705228571940418209598120253460691012961376915204638217432434939663388118421990178723293221119107842478896497182400216187965781021419486152317565806727158425733868571589928319435844161162712316696612821101631042672561493581050004791398216349925543124125741075809841585805429582556349177941365734609949648120195209980606930181653040002526878401266995087092187331020253591024297006865878160227785077610460644644034860770525405019799027594373663903789926134757291081635665476108973751146607578322753546323637679113713086026443479247
```

Per a completar una clau, calen els exponents públic, e, i privat, d, de manera que $de \equiv 1 \pmod{lcm(p-1, q-1)}$. I es recomana triar d a l'atzar, i prèviament al càlcul de p i q; óbviament, obviarem aquest pas previ i construirem d i e ara.

Notem que d i e han de ser, en particular, senars.

In [79]:
```
m=lcm(p-1,q-1)
```

In [80]:
```
m
```

Out[80]:
```
338107230090105994395783001080741188200549539720833833952327869054906069203820730487009539686409242364362700421091709663142249801809848420056627333487664847937124848234569372956795895188678218446462326623844459007530387633232434772157154589438677325065768962765158785967459814219254822074611493776540638814818107942256778718403688445572313207129711544633795896034745845140832282240516446909804909045175086546244727672611331450376504478737774936049591608689459439980807514006410363586637574036887004966498831494389550707027213064852743969612353419078189570753337115924528106011202567358356989795119499169798012624481450879515365225960280590168352316965049046628114028770953038367408451844505281896366104560503566454623474762651909711562863926844056554444602474560721534425378041109556768517940414065279272697241313988461625438975737196013425444018802109900004136503928243890717642844007060656152058633230672704739844407076627844677836152787613587738807330319112906583429206386365742271850903838321961042273840790250355507111018278347663249127983550491592625334538783698308128168240687096256905899565143082056787375237197357278401211492354760927420363400453392162336267970138029086778407233937270471534034292986985209861127703426576160
```

In [84]:
```
d=2*ZZ.random_element(2,(m-2)//2)+1
```

In [85]:
```
d
```

Out[85]:
```
118215289282183358215084286305844693430761522268582022942533190788528070029970372052672434499183468696734266376047511801688913192717580764869468812945708063214430267826271103201123979063399784292388656462682042582572555934203099812161142549736862131369290772877474637485659924450250786213120447989355221450752740700287683002281240391516326638923564707559798369801746207371467653639815961541559258465887150649456960465730248110852695579374134227113508439277930954357381652420006050140630400443631137673483714697086455652281922992404537399241711957337758566875421850250524988642140910155659845875161763693500240019673166475256935266276836129499818895856282177600419505190116808462938967074297980338559312646075685447725629332294624228526083019618074349717923042647133456194059745461483866187648644481398103133040255546934457198693739036667793493946467666349241934589251863603831231463737539320418902877142286059956793906773474266227486371452042924697579086368902899694711844656493489115373173184119701575912738404064480519587769596013756668311600178761294412521113536323482428448405707497634405755024733304641520948582341927525294229555643200038194090568961670022417402780433754132490576595593704098033650166996212952426061634613475 71
```

In [86]:
```
e=Mod(d,m)^(-1)
```

In [87]:
```
1 e
```

Out[87]:
```
105389528762407066430051754038301139443623925672101689828295000020892099123709434965338413632530240778397572228384592751300612997799044558826680353165334903849568935115234208199221313782995264340765780256291389964303653066695461768671407111902675110272888991949561751887881330751444133498675801072660782579851366873423040363829807843021841723603454724411824762530999069507963905516722391370818139864715872054176133486826375918858838838622069954825059948870435067255148341044934173947697720327739679492186266790619992583255756252573000426849961541504913110289124833682057065772172374818594429592083620377419405511311888390152890666624136233220182712229318638305833978193824791949326022983317342433451296664370883007483192726779835590116614603620904979112173558550621127805164320267776838005660287082529597664421855615168637241743926870112455116945789083540662250394328895026978199459534233687371814777268658375495853609633237139426779953002819276797907052440018487929730520256380633083936303368791767491821989936413316591469026243625503597760832911679342105687611989198396565874039406343759056288971216362989199068013646859381619151116141027912336688630106642048969131949439887989788261678630317926740041646191327941044640524023074011
```

Escrivim les claus:

In [88]:
```
1 ClauPublicaRSA4096=[n,e]
```

In [89]:
```
1 ClauPrivadaRSA4096=[n,d]
```

In [90]:
```
1 ClauPublicaRSA4096
```

Out[90]:
```
[67621446018021198879156600216148237640109907944166766790465573810981213840764146097401907937281848472872540084218341932628449960361969684011325466697532969587424969646913874591359179037735643689292465324768891801506077526646486954431430917887735465013153792553031757193491962843850964414922298755308127762963621588451355743680737689114462641425942308926759179206949169028166456448103289381960981809035017309248945534522266290075300895747554987209918321737891887996161502801282072717327514807377400993299766298877910141405442612970548793922470683815637914150667423184905621202240513471671397959023899833959602524896295415966159790645752509323352905527377042648947906022310963954824212037917453382059661402770522857194041820959812025346069101296137691520463821743243493966338811842199017872329322111910784247889649718240021618796578102141948615231756580672715842573386857158992831943584416116271231669661282110163104267256149358105000479139821634992554312412574107580984158580542958255634917794136573460994964812019520998060693018165304000252687840126699508709218733102025359102429700686587816022778507761046064464403486077052540501979902759437366390378992613475729108163566547610897375114660757832275354632363767911371308602644347924 7,
 1053895287624070664300517540383011394436239256721016898282950000208920991237094349653384136325302407783975722283845927513006129977990445588266803531653349038495689351152342081992213137829952643407657802562913899643036530666954617686714071119026751102728889919495617518878813307514441334986758010726607825798513668734230403638298078430218417236034547244118247625309990695079639055167223913708181398647158720541761334868263759188588388386220699548250599488704350672551483410449341739476977203273967949218626679061999258325575625257300042684996154150491311028912483368205706577217237481859442959208362037741940551131188839015289066662413623322018271222931863830583397819382479194932602298331734243345129666437088300748319272677983559011661460362090497911217355855062112780516432026777683800566028708252959766442185561516863724174392687011245511694578908354066225039432889502697819945953423368737181477726865837549585360963323713942677995300281927679790705244001848792973052025638063308393633036879176749182198993641331659146902624362550359776083291167934210568761198919839656587403940634375905628897121636298919906801364685938161915111614102791233668863010664204896913194943988798978826167863031792674004164619132794104464052402307401 1]
```

In [91]:
```
1 ClauPrivadaRSA4096
```

Out[91]:
```
[67621446018021198879156600216148237640109907944166766790465573810981213840764146097401907937281848472872540084218341932628449960361969684011325466697532969587424969646913874591359179037735643689292465324768891801506077526646486954431430917887735465013153792553031757193491962843850964414922298755308127762963621588451355743680737689114462641425942308926759179206949169028166456448103289381960981809035017309248945534522266290075300895747554987209918321737891887996161502801282072717327514807377400993299766298877910141405442612970548793922470683815637914150667423184905621202240513471671397959023899833959602524896295415966159790645752509323352905527377042648947906022310963954824212037917453382059661402770522857194041820959812025346069101296137691520463821743243493966338811842199017872329322111910784247889649718240021618796578102141948615231756580672715842573386857158992831943584416116271231669661282110163104267256149358105000479139821634992554312412574107580984158580542958255634917794136573460994964812019520998060693018165304000252687840126699508709218733102025359102429700686587816022778507761046064464403486077052540501979902759437366390378992613475729108163566547610897375114660757832275354632363767911371308602644347924 7,
 1182152892821833582150842863058446934307615222685820229425331907885280700299703720526724344991834686967342663760475118016889131927175807648694688129457080632144302678262711032011239790633997842923886564626820425825725559342030998121611425497368621313692907728774746374856599244502507862131204479893552214507527407002876830022812403915163266389235647075597983698017462073714676536398159615415592584658871506494569604657302481108526955793741342271135084392779309543573816524200060501406304004436311376734837146970864556522819229924045373992417119573377585668754218502505249886421409101556598458751617636935002400196731664752569352662768361294998188958562821776004195051901168084629389670742979803385593126460756854477256293322946242285260830196180743497179230426471334561940597454614838661876486444813981031330402555469344571986937390366677934939464676663492419345892518636038312314637375393204189028771422860599567939067734742662274863714520429246975790863689028996947118446564934891153731731841197015759127384040644805195877695960137566683116001787612944125211135363234824284484057074976344057550247333046415209485823419275252942295556432000381940905689616700224174027804337541324905765955937040980336501669962129524260616346134757 1]
```

Notem que, efectivament, n és de 4096 bits.

In [92]:
```
1 2^4095<n<2^4096
```

Out[92]:
```
True
```

Fi del capítol 10