Primeritat i factorització

Artur Travesa

(versió 2024-07)

Capítol 3. Construcció certificada de nombres primers

3.0. Introducció

L'objectiu d'aquest capítol és aprendre a construir nombres primers de mida (en bits) prefixada, i de certificar que ho són, de primers.

Com a exemple, construirem una parella de nombres primers de 512 bits cadascun, de manera que el seu producte serà un nombre natural d'entre 1013 i 1024 bits. En particular, si satisfessin algunes condicions extres, podrien formar part d'una clau RSA de 1024 bits.

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.

3.0.0. Funcions que aprofitarem de capítols anteriors

En aquesta introducció, afegirem les funcions que hem programat en capítols anteriors i que seran útils per a aquest, encara que *SageMath* tingui funcions programades més eficients que les nostres.

3.0.0.0. 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'

```

3.0.0.1. 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']

```

3.1. Primers de 112 bits

Observació. A fi d'evitar feines feixugues i, alhora, poc eficients, suposarem que la funció is_prime( ) de *SageMath* dóna la resposta correcta per a nombres naturals menors que

2^{32}; és a dir, de 32 o menys xifres binàries (de fet, podríem suposar això mateix per a un interval molt més gran, però ens limitarem a aquest).

D'aquesta manera, si un nombre natural n és menor que 2^{32} i la comanda is_prime(n) retorna True donarem el nombre n com a primer certificat. (I, de fet, el nombre és prou petit perquè la nostra funció el pugui certificar sense problemes.)

3.1.0. Construcció d'una primera llista de nombres primers

Es tracta de construir una llista, que anomenarem lp112, de 10 nombres primers de 112 bits i certificar-los, però sense emmagatzemar, ni escriure, els certificats. (En farem servir 8, d'aquests nombres, però en tenim un parell més per si algun presentés problemes.)

Per a construir aquesta llista, definirem una funció Primer112( ) que calculi un nombre primer d'aquesta mida a partir d'una cerca 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 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 10 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(10)]
```

In [9]:
```
lp112
```

Out[9]:
```
[4836078527075274517620578857420277,
 2702358511910207490966490926725869,
 3812072656193959618310201348810357,
 5112470108458166326655290361928649,
 4307762224420481611345583776836697,
 4157980550779699401147180044184467,
 4437399728826100707896994190877723,
 4905770740078333347546692347112461,
 4408388088396110552787569923997423,
 4885247126083381437648993984927389]
```

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 [10]:
```
1 [Certifica(lp112[i],[],50) for i in range(10)]
```

Out[10]:
```
[[48360785270752745176205788574202777,
  True,
  45712285699407687933353638207935 79,
  [2, 12090196317688186294051447143550 69]],
 [27023585119102074909664909267258 69,
  True,
  44459145344937925546242922497225 2,
  [2, 3, 25037, 97379, 553961, 13436329, 1378833383]],
 [38120726561939596183102013488103 57,
  True,
  12566320201093095302460492330004 78,
  [2, 347, 27967, 61819, 15885608143115380666 19]],
 [51124701084581663266552903619286 49,
  True,
  36384167085388507252860774235097 68,
  [2, 3, 7, 227693, 44550293147942435636281859]],
 [43077622244204816113455837768366 97,
  True,
  38211553459768512739554109452529 14,
  [2, 3, 17949009268418673380606599070 1529]],
 [41579805507796994011471800441844 67,
  True,
  12359535915709185899950196058177 50,
  [2, 101, 107, 19237441245395111507111964 6719]],
 [44373997288261007078969941908777 23,
  True,
  19826975257400128054897333305870 42,
  [2, 97, 17377, 13162913321335705354978034 69]],
 [49057707400783333475466923471124 61,
  True,
  48133661451933047535385146938256 95,
  [2, 3, 5, 81762845667972222459111539118541]],
 [44083880883961105527875699239974 23,
  True,
  26336679867434083254854580373632 884,
  [2, 7, 15647, 66301, 5716561, 31926281, 1663096499]],
 [48852471260833814376489939849273 89,
  True,
  17763126788765913964799785732639 74,
  [2, 7, 17447311164583505134460692803312 1]]]
```

In [11]:
```
1 [Pocklington(lp112[i],[2],50) for i in range(10)]
```

Out[11]:
```
['U és massa gran.',
 'U és massa gran.',
 'U és massa gran.',
 'U és massa gran.',
 'U és massa gran.',
 'U és massa gran.',
 'U és massa gran.',
 'U és massa gran.',
 'U és massa gran.',
 'U és massa gran.']
```

3.2. Primers de 480 bits

(a) Notem que el producte de quatre nombres de 112 bits és un nombre de (aproximadament) 448 bits. Siguin n1 el producte dels quatre primers nombres de lp112, i n2, el producte dels quatre darrers.

(b) Fixem-nos en n1 (per a n2 es procedeix de la mateixa manera). Per a tot nombre natural k1, el nombre p:=2*k1*n1+1 és senar; i el seu nombre de bits depèn d'un interval on es pot triar k1, interval que cal calcular a fi d'obtenir un nombre q1 d'una quantitat de bits desitjada a priori (òbviament, més gran que una unitat més que el nombre de bits de n1). Es demana calcular l'interval on cal triar k1 a fi que els nombres q1 siguin de 480 bits.

(c) A partir d'un test de primeritat aplicat a valors successius de q1, obtinguts a partir de tries a l'atzar de valors de k1, obtindrem un valor que passi el test, de manera que (probablement) sigui un nombre primer. I com que el valor de k1 és prou petit per a poder-lo factoritzar, també coneixerem els divisors primers de q1-1=2*k1*n1 (els de n1, perquè l'hem construït com ho hem fet!). Això ens permetrà certificar el nombre q1 com a primer. Ara, però, podem usar la funció Pocklington, perquè n1 és més gran que l'arrel quadrada de q1; això evita factoritzar k1.

(d) Així, tenim una parella de nombres primers (amb certificats), q1 i q2, de 480 bits cadascun, i diferents, perquè els nombres q1-1 i q2-1 són divisibles per primers diferents.

(a)

In [12]:
```
n1=prod(lp112[0:4])
```

In [13]:
```
n2=prod(lp112[6:10])
```

(b)

In [14]:
```
x11=floor(2^478/n1)+1
x12=floor((2^479-1)/n1)
x21=floor(2^478/n2)+1
x22=floor((2^479-1)/n2)
```

(c)

In [15]:
```
kq1=ZZ.random_element(x11,x12)
kq2=ZZ.random_element(x21,x22)
```

In [16]:
```
q1=2*kq1*n1+1
q2=2*kq2*n2+1
```

In [17]:
```
print(q1,q2)
```

```
306050607497892733863668236509226294108158374331252420802862648417050852181299123332862740036082390992417498246600829191609245809344336335767809 5 28799542730978119085896619856120595280038756158093198813441821514582996803741833883024563262682342655978541868124466884176625516112211641728510 71
```

In [18]:
```
SolovayStrassenTest(q1,50)
```

Out[18]: False

In [19]:
```
SolovayStrassenTest(q2,50)
```

Out[19]: False

Com era d'esperar, els nombres triats a l'atzar no són primers. Caldrà repetir la cerca, i serà útil automatitzar-la.

Ho farem un màxim de 500 vegades per a cadascun. Si no el trobem, podem repetir la cerca, o bé canviar el nombre n1 (o n2).

In [20]:
```
fita=500
```

In [21]:
```
while fita>0 and SolovayStrassenTest(q1,25)==false:
    kq1=ZZ.random_element(x11,x12)
    q1=2*kq1*n1+1
    fita=fita-1
if fita==0:
    q1=0
else:
    q1
```

In [22]:
```
fita==0
```

Out[22]: False

In [23]:
```
q1
```

Out[23]:
```
204916578776191265458653830411614147267053138938455120180847837602350716687802454555872249767965646688543091228433203217290905066826152968221304 3
```

In [24]:
```
fita=500
```

In [25]:
```
while fita>0 and SolovayStrassenTest(q2,25)==false:
    kq2=ZZ.random_element(x21,x22)
    q2=2*kq2*n2+1
    fita=fita-1
if fita==0:
    q2=0
else:
    q2
```

In [26]:
```
1 fita==0
```

Out[26]:
```
False
```

In [27]:
```
1 q2
```

Out[27]:
```
18024339747248977924069514623883238570885924664981472132101456484772687973410413952377692724175767690488143644532689465495086803115336058581539 47
```

Cal certificar que els nombres q1 i q2 són primers. Com que coneixem els divisors primers de n1 i de n2, si volem fer-ho amb la funció Certifica, només cal calcular els de k1 i els de k2. I aquests són fàcils, perquè aquests nombres són prou petits per a poder-ne calcular la factorització. Donarem per certificats els factors primers d'aquests nombres (tot i que, formalment, caldria certificar-los).

Però no cal usar aquesta funció, i podem usar la funció Pocklington, perquè coneixem prou factorització (la de n1 i la de n2).

(Notem que les fites superiors dels intervals on es trien els valors de k1 i de k2 són menors que 2^{36}.)

In [28]:
```
1 x12<2^36, x22<2^36
```

Out[28]:
```
(True, True)
```

Farem, doncs, dues llistes de primers, fq1 i fq2, fent la reunió de les llistes de divisors primers dels factors corresponents.

Notem que els divisors primers de n1 i de n2 són els que hem usat per a construir aquests nombres com a producte; per tant, són les llistes lp112[0:4] i lp112[6:10]. Per comoditat, els donem un nom.

In [29]:
```
1 f0q1=lp112[0:4]
2 f0q2=lp112[6:10]
```

In [30]:
```
1 fkq1=[factor(kq1)[i][0] for i in range(len(factor(kq1)))]
2 fkq2=[factor(kq2)[i][0] for i in range(len(factor(kq2)))]
```

In [31]:
```
1 fq1=set([2]).union(fkq1).union(f0q1)
2 fq2=set([2]).union(fkq2).union(f0q2)
```

Ara ja podem certificar amb la funció Certifica.

In [32]:
```
1 Certifica(q1,fq1,50)
```

Out[32]:
```
[204916578776191265458653830411614147267053138938455120180847837602
350716687802454555872249767965646688543091228433203217290905066826 1
529682213043,
 True,
 829952303314104872742542812228995973103756344680136182660711102305
427957841962165369263788021274722506978950644339017463236994881417 4
91431765574,
 [2,
  13,
  113,
  487,
  5623,
  270235851191020749096649092672586 9,
  381207265619395961831020134881035 7,
  483607852707527451762057885742027 7,
  511247010845816632665529036192864 9]]
```

In [33]:
```
1 Certifica(q2,fq2,50)
```

Out[33]:
```
[180243397472489779240695146238832385708859246649814721321014564847
726879734104139523776927241757676904881436445326894654950868031153 3
605858153947,
 True,
 143823332298279005279742914860920792884823576849868853465217521420
649334847882750516177669043823149443603172488287950282203139112596 7
455202223640,
 [2,
  89,
  21599177,
  440838808839611055278756992399742 3,
  443739972882610070789699419087772 3,
  488524712608338143764899398492738 9,
  490577074007833347546692347112461]]
```

Però també podem certificar amb la funció Pocklington.

In [34]:
```
1 Pocklington(q1,f0q1,50)
```

Out[34]:
```
[204916578776191265458653830411614147267053138938455120180847837602
350716687802454555872249767965646688543091228433203217290905066826 1
529682213043,
 True,
 648999231984156350825346888606987001172369844834024599092563495960
004672335060519250551270812278733822635064146299532944380904324797 5
20480215992,
 [2,
  270235851191020749096649092672586 9,
  381207265619395961831020134881035 7,
  483607852707527451762057885742027 7,
  511247010845816632665529036192864 9]]
```

In [35]:
```
1 Pocklington(q2,f0q2,50)
```

Out[35]:
```
[18024339747248977924069514623883238570885924664981472132101456484772687973410413952377692724175767690488143644532689465495086803115336058581539 47,
 True,
 40827549410051472001296170771367761706153968023935149180102509935829603313771538135086266709831285227340124923197852327583938676881012526608166 7,
 [2,
  440838808839611055278756992399742 3,
  443739972882610070789699419087772 3,
  488524712608338143764899398492738 9,
  490577074007833347546692347112461]]
```

(d)

I notem que, efectivament, els nombres primers q1 i q2 són de 480 bits.

In [36]:
```
1 2^479<q1<2^480
```

Out[36]:
```
True
```

In [37]:
```
1 2^479<q2<2^480
```

Out[37]:
```
True
```

3.3. Primers de 512 bits

Es tracta de repetir el procés, però en comptes de n1 i n2, fer servir q1 i q2 per a construir p1 i p2 de 512 bits.

Caldrà veure on triem els nous valors de k, triar-los, reiterar les cerques a l'atzar, i finalment, certificar els primers obtinguts.

In [38]:
```
1 y11=floor(2^510/q1)+1
2 y12=floor((2^511-1)/q1)
3 y21=floor(2^510/q2)+1
4 y22=floor((2^511-1)/q2)
```

In [39]:
```
1 kp1=ZZ.random_element(y11,y12)
2 kp2=ZZ.random_element(y21,y22)
```

In [40]:
```
1 p1=2*kp1*q1+1
2 p2=2*kp2*q2+1
```

In [41]:
```
1 SolovayStrassenTest(p1,50),SolovayStrassenTest(p2,50)
```

Out[41]:
```
(False, False)
```

In [42]:
```
fita=1000
```

In [43]:
```
while fita>0 and SolovayStrassenTest(p1,25)==false:
    kp1=ZZ.random_element(y11,y12)
    p1=2*kp1*q1+1
    fita=fita-1
if fita==0:
    p1=0
else:
    p1
```

In [44]:
```
fita
```

Out[44]:
```
911
```

In [45]:
```
fita==0
```

Out[45]:
```
False
```

In [46]:
```
p1
```

Out[46]:
```
68704604424290156958885324474046478591462947244578935867249648458791137645904588232655784015394675117382507524736300823109544843867512275858647738302828 99
```

In [47]:
```
fita=1000
```

In [48]:
```
while fita>0 and SolovayStrassenTest(p2,25)==false:
    kp2=ZZ.random_element(y21,y22)
    p2=2*kp2*q2+1
    fita=fita-1
if fita==0:
    p2=0
else:
    p2
```

In [49]:
```
fita
```

Out[49]:
```
855
```

In [50]:
```
fita==0
```

Out[50]:
```
False
```

In [51]:
```
p2
```

Out[51]:
```
96117370768930929141853885990565871884089824201202163013315454838255851772587771750365664270963798071775322551968375468222789138593013699592009539094683 37
```

In [52]:
```
fkp1=[factor(kp1)[i][0] for i in range(len(factor(kp1)))]
fkp2=[factor(kp2)[i][0] for i in range(len(factor(kp2)))]
```

In [53]:
```
fp1=sorted(set([2,q1]).union(fkp1))
fp2=sorted(set([2,q2]).union(fkp2))
```

In [54]:
```
Certifica(p1,fp1,50)
```

Out[54]:
```
[687046044242901569588853244740464785914629472445789358672496484587
911376459045882326557840153946751173825075247363008231095448438675 1
227585864773830282899,
 True,
 371818739263817428924834531948417442421300586241507825945726768700
510852359560923601850713505515096841905852166006376340903653583557 5
333359108504884964624,
 [2,
  23,
  1709,
  42649,
  204916578776191265458653830411614147267053138938455120180847837 60
235071668780245455587224976796564668854309122843320321729090506682 6
1529682213043]]
```

In [55]:
```
Certifica(p2,fp2,50)
```

Out[55]:
```
[961173707689309291418538859905658718840898242012021630133154548382
558517725877717503656642709637980717753225519683754682227891385930 1
369959200953909468337,
 True,
 512117124318594935741499011144642657784787111549158278787318718815
978559431778708289989000903554271729325418149294923814556294268211 4
007614684245581024835,
 [2,
  3,
  257,
  269,
  1607,
  180243397472489779240695146238832385708859246649814721321014564847726879734104139523776927241757676904881436445326894654950868031153 3605858153947]]
```

Notem que, ara, la certificació encara és més senzilla amb la funció Pocklington, perquè per a p1 només necessitem el primer q1 i per a p2 només necessitem el primer q2.

In [56]:
```
Pocklington(p1,[q1],50)
```

Out[56]:
```
[687046044242901569588853244740464785914629472445789358672496484587
911376459045882326557840153946751173825075247363008231095448438675 1
227585864773830282899,
 True,
 374628660239904354458484719932094903655207856828721559742511585016
409750363882312109741410092741064306127702353221021427902428794834 5
735463499983577471721,
 [2,
  204916578776191265458653830411614147267053138938455120180847837 60
235071668780245455587224976796564668854309122843320321729090506682 6
1529682213043]]
```

In [57]:
```
1 Pocklington(p2,[q2],50)
```

Out[57]:
```
[9611737076893092914185388599056587188408982420120216301331545483825585177258777175036566427096379807177532255196837546822278913859301369959200953909468337,
 True,
 7051677869146333019795441759566079564754045346631756701576891195851379712199689517052764087780879385730650569669447366774807670323633502215705522097735350,
 [2,
  18024339747248977924069514623883238570885924664981472132101456484772687973410413952377692724175767690488143644532689465495086803115336058581539 47]]
```

3.4. Els valors obtinguts

In [58]:
```
1 p1
```

Out[58]:
```
6870460442429015695888532447404647859146294724457893586724964845879113764590458823265578401539467511738250752473630082310954484386751227585864773830282899
```

In [59]:
```
1 p2
```

Out[59]:
```
9611737076893092914185388599056587188408982420120216301331545483825585177258777175036566427096379807177532255196837546822278913859301369959200953909468337
```

En el moment de crear aquest fitxer, han aparegut els nombres primers

p1=74571552008823797195594432183249374523843100238599934516554961399020033

i

p2=11993318319225874756624727275295332613405843841164464060792461783148133

Però n'hem perdut el certificat, perquè no l'hem emmagatzemat. Per tant, difícilment podrem certificar-los!

Fi del capítol 3