Primeritat. Construcció de primers

Artur Travesa

(versió 2021-04)

Introducció general

L'origen d'aquestes notes es remunta a diferents cursos d'Aritmètica o de Criptografia, a càrrec de l'autor, per a estudiants de Matemàtiques o d'Informàtica de la Universitat de Barcelona.

La idea bàsica és descriure algunes aplicacions importants de l'Aritmètica bàsica (diguem, de nivell de primer curs) a les transmissions xifrades d'informació.

En cap cas es tracta d'un curs de Criptografia, que caldria encabir en espais més amplis de coneixements, que haurien d'incloure, probablement, parts de teoria de la comunicació, de complexitat algorítmica o computacional, d'aprenentatge automàtic, o d'estudi de teories de compartició de secrets, entre d'altres.

El format triat per a la presentació és el d'un *notebook* de *Mathematica*, per la facilitat que té aquest programari per a poder desenvolupar els càlculs no trivials de manera prou senzilla i entenedora, d'una banda, i per a permetre fer una presentació escrita prou raonable des del punt de vista de material escrit, de l'altra. En particular, la possibilitat d'incloure els càlculs dins del text de manera natural en fan una bona eina comunicativa i, alhora, facilita molt el càlcul amb exemples no trivials.

A fi de veure tot el contingut del *notebook* convé executar-lo. Això es pot fer de cop o bé, més recomanable, cel·la a cel·la a mesura que s'avança en la lectura i comprensió dels diferents continguts.

Observació: Per al cas en què no es disposi del programari, hi ha la versió executada del *notebook* en format pdf.

Amb la finalitat doble d'una banda, de no fer textos molt llargs o amb molts continguts, i de l'altra de poder ampliar de manera senzilla els continguts que s'hi tractin, el material s'ha dividit en diferents *notebooks*, que desrivim a continuació, en l'apartat de referències.

Referències

[Cripto- 1]: **Criptografia bàsica (1)**.
Travesa, A.: CriptografiaBasica-1 ; accessible en forma notebook o en format pdf des de https://travesa.cat/notes/
Contingut: Una iniciació a la codificació de missatges. El criptosistema de Cèsar. El criptosistema de Vigenère.

[Cripto- 2]: **Criptografia bàsica (2)**.
Travesa, A.: CriptografiaBasica-2; accessible en forma notebook o en format pdf des de https://travesa.cat/notes/
Contingut: Els criptosistemes lineals. Els criptosistemes afins. Sufixació de missatges. Farciment de missatges.

[Eratostenes]: **Un garbell d'Eratòstenes.**
Travesa, A.: Eratostenes; accessible en forma notebook o en format pdf des de https://travesa.cat/notes/
Contingut: Un garbell d'Eratòstenes.

[Cripto- 3]: **Primeritat. Construcció de primers**.
Travesa, A.: ConstruccioDePrimers; accessible en forma notebook o en format pdf des de https://travesa.cat/notes/
Contingut: Test de primeritat de Solovay-Strassen. Test de primeritat de Miller-Rabin. Un certificat congruencial de primeritat. Construcció certificada de nombres primers de mida prefixada. Aplicació al càlcul de claus RSA. Aplicació (exercici) al càlcul de claus ElGamal.

[Cripto- 4]: **Factorització**.
Travesa, A.: Factoritzacio; accessible en forma notebook o en format pdf des de https://travesa.cat/notes/
Contingut: Un garbell d'Eratòstenes. Tests de primeritat de Solovay-Strassen i de Miller-Rabin. Un certificat congruencial de primeritat. Un algoritme bàsic de divisó per nombres primers petits. Un algoritme bàsic de divisó per nombres petits. El mètode de factorització de Fermat. El mètode de factorització p-1 de Pollard. El mètode de factorització rho de Pollard.

[RSA]: **Criptosistemes de tipus RSA**.
Travesa, A.: RSA; accessible en forma notebook o en format pdf des de https://travesa.cat/notes/
Contingut: Una descripció bàsica dels criptosistemes de tipus RSA: les claus; xifratge; desxifratge; observacions sobre la seguretat.

[ElGamal]: **El criptosistema ElGamal**.
Travesa, A.: ElGamal; accessible en forma notebook o en format pdf des de https://travesa.cat/notes/
Contingut: Logaritmes discrets. Una descripció bàsica del criptosistema ElGamal: el grup cíclic; les claus; xifratge i desxifratge.

[Tr-1]: Travesa, A.: *Aritmetica*. Edicions de la Universitat de Barelona, col·lecció UB, n. 25. Barcelona, 1998. ISBN:84-8338-031-5.

1. Tests de primeritat

Exercici 1 (Test de Solovay-Strassen)

La funció següent és una implementació del test de primeritat de Solovay-Strassen. Es demana esbrinar i explicar com funciona; en particular, a quins nombres es pot aplicar, en quants passos acaba el càlcul, què podem dir d'un nombre segons la sortida de l'algoritme, ...

In[1]:=
```
SolovayStrassen[nn_, ff_] := Module[{n = Abs[nn], f, g, x, y},
  If[Not[IntegerQ[n]], Print["Cal que n sigui un nombre enter."]; Return[Null], Null];
  If[ff < 1, Print["Cal fer alguna prova."]; Return[Null], Null];
  If[n == 2 || n == 3 || n == 5 || n == 7, Return[True], Null];
  If[n == 1 || EvenQ[n], Return[False], Null];
  f = 0;
  While[f < ff,
   g = Random[Integer, {2, n - 2}];
   x = PowerMod[g, (n - 1) / 2, n];
   If[x == 1 || x == n - 1,
    y = Mod[JacobiSymbol[g, n], n]; If[x ≠ y, Return[False], Null], Return[False]];
   f++];
  Return[Indeterminate]
 ]
```

S'aplica a un nombre enter nn i un nombre ff com a fita per al nombre de passos d'algoritme.

Proporciona
True, si nn és un dels nombres primers 2, 3, 5, o bé 7;
False, si nn és o bé 1 o bé és parell >2; o bé per a algun valor de g triat a l'atzar entre 2 i nn-2 és g^((nn-1)/2) != + - 1 (mod nn); per tant, nn és compost amb tota seguretat;
Indeterminate, si per a ff nombres g triats a l'atzar se satisfà que g^((nn-1)/2) = (g / nn) = +-1 (mod nn); és a dir, si nn es comporta com si fos primer per a ff valors g triats a l'atzar.

Si la sortida és Indeterminate, el fet és que si nn és compost, la probabilitat que el test no ho hagi detectat és menor que 1/ (2^ff) .
De fet, si el nombre nn és compost, el test ho detecta en pocs passos (i probabilitat alta).

Exercici 2 (Test de Miller-Rabin)

La funció següent és una implementació del test de primeritat de Miller-Rabin. Es demana esbrinar i explicar com funciona; en particular, a quins nombres es pot aplicar, en quants passos acaba el càlcul, què podem dir d'un nombre segons la sortida de l'algoritme, ...

In[2]:=

```
MillerRabin[nn_, ff_] := Module[{n = Abs[nn], v, m, f, g, x, k},
  If[Not[IntegerQ[n]], Print["Cal que n sigui un nombre enter."]; Return[Null], Null];
  If[ff < 1, Print["Cal fer alguna prova."]; Return[Null], Null];
  If[n == 2 || n == 3 || n == 5 || n == 7, Return[True], Null];
  If[n == 1 || EvenQ[n], Return[False], Null];
  v = 0; m = n - 1; While[EvenQ[m], v++; m = m / 2];
  f = 0;
  While[f < ff,
   g = Random[Integer, {2, n - 2}];
   x = PowerMod[g, m, n];
   If[x ≠ 1 && x ≠ n - 1,
    k = 1;
    x = PowerMod[x, 2, n];
    While[x ≠ n - 1 && k < v - 1, x = PowerMod[x, 2, n]; k++];
    If[k ≥ v - 1 && x ≠ n - 1, Return[False], Null], Null];
   f++];
  Return[Indeterminate]
 ]
```

S'aplica a un nombre enter nn i un nombre ff com a fita per al nombre de passos d'algoritme.

Proporciona
True, si nn és un dels nombres primers 2, 3, 5, o bé 7;
False, si nn és o bé 1 o bé és parell >2; o bé per mitjà d'algun valor de g triat a l'atzar entre 2 i nn-2 el test demostra que nn és compost.
Indeterminate, si per a ff nombres g triats a l'atzar el nombre nn es comporta com si fos primer.

Si la sortida és Indeterminate, el fet és que si nn és compost, la probabilitat que el test no ho hagi detectat és menor que 1/ (4^ff) .
De fet, si el nombre nn és compost, el test ho detecta en pocs passos (i probabilitat alta).

Exercici 3

Es demana aplicar els tests de primeritat dels dos exercicis anteriors: d'una banda, a una llista de 10000 nombres enters aleatoris i comparar-ne els resultats; i, de l'altra, a una llista de 10000 nombres naturals primers aleatoris i comparar-ne els resultats. Podem dir que algun dels mètodes és millor que l'altre? Per què? Quin tarda més? (Es pot provar amb una fita ff=2 en tots els casos...)

In[3]:=
```
n = 10 000
```
Out[3]=
```
10 000
```

In[4]:=
```
nr = RandomInteger[10^20, n];
```

In[5]:=
```
t = Table[{a = SolovayStrassen[nr[[i]], 2], b = MillerRabin[nr[[i]], 2], a == b}, {i, 1, n}];
```

In[6]:=
```
Select[t, #[[1]] ≠ #[[2]] &]
```
Out[6]=
```
{}
```

In[7]:=
```
Timing[Table[SolovayStrassen[nr[[i]], 2], {i, 1, n}];]
```
Out[7]=
```
{0.724, Null}
```

In[8]:=
```
Timing[Table[MillerRabin[nr[[i]], 2], {i, 1, n}];]
```
Out[8]=
```
{0.892, Null}
```

In[10]:=
```
Timing[Table[SolovayStrassen[nr[[i]], 4], {i, 1, n}];]
```
Out[10]=
```
{0.764, Null}
```

In[11]:=
```
pr = RandomPrime[10^20, n];
```

In[12]:=
```
tp = Table[{a = SolovayStrassen[pr[[i]], 2], b = MillerRabin[pr[[i]], 2], a == b}, {i, 1, n}];
```

In[13]:=
```
Select[tp, #[[1]] ≠ #[[2]] &]
```
Out[13]=
```
{}
```

In[14]:=
```
Timing[Table[SolovayStrassen[pr[[i]], 2], {i, 1, n}];]
```
Out[14]=
```
{2.156, Null}
```

In[15]:=
```
Timing[Table[MillerRabin[pr[[i]], 2], {i, 1, n}];]
```
Out[15]=
```
{2.184, Null}
```

In[16]:=
```
Timing[Table[SolovayStrassen[pr[[i]], 4], {i, 1, n}];]
```
Out[16]=
```
{3.84, Null}
```

Per a nombres naturals triats a l'atzar, ha trigat més el test de Miller-Rabin, fins i tot en el cas d'aplicar el de Solovay-Strassen amb una fita doble a fi de millorar la probabilitat d'aquest fins a la probabilitat de Miller-Rabin.
En canvi, si passem els tests només a nombres primers, en tots els casos funciona millor el test de Miller-Rabin.
Això suggereix que, per a fer servir el test com a garbell per a determinar nombres primers d'entre nombres naturals arbitraris, sembla millor usar el test de Solovay-Strassen; en canvi, si es tracta de "confirmar" que certs nombres "són" primers, sembla millor usar el test de Miller-Rabin.

Notem que en tots els casos ens referim a aquestes implementacions dels tests; per a altres implementacions, els resultats poden ser diferents.

2. Certificats de primeritat

Exercici 4 (Un certificat congruencial de primeritat)

La funció següent és una implementació d'un certificat de primeritat. Es demana esbrinar i explicar com funciona; en particular, a quins nombres es pot aplicar, en quants passos s'acaba, què podem dir d'un nombre segons la sortida de l'algoritme, ...

In[17]:=
```
Certificat[pp_, f_, n_] := Module[{p = Abs[pp], fppmu, l, m, g, mm, s},
  If[IntegerQ[p], Null, Print["Cal que p sigui un nombre enter."]; Return[Null]];
  If[n < 1, Print["Cal fer alguna prova."]; Return[Null], Null];
  If[p == 2 || p == 3, Return[{True, p - 1, {p - 1}}], Null];
  If[p == 1 || EvenQ[p], Return[{False, 2}], Null];
  If [f == {}, fppmu = Transpose[FactorInteger[p - 1]][[1]], fppmu = Sort[f]]; l = Length[fppmu];
  m = 0; While[m < n, g = Random[Integer, {2, p - 2}]; If[(s = PowerMod[g, (p - 1) / 2, p]) == p - 1,
    i = 2; While[i ≤ l && PowerMod[g, (p - 1) / fppmu[[i]], p] ≠ 1, i++];
    If[i == l + 1, Return[{True, g, fppmu}], Null], If[s ≠ 1, Return[{False, g}], Null]]; m++];
  Return[Indeterminate]
 ]
```

S'aplica un nombre màxim n de vegades a un nombre pp i la llista f dels divisors primers de pp-1. Si la llista és buida, intenta calcular-la.

Proporciona llistes de la forma
{True, g, lta}, si pp és primer, on g és una arrel primitiva mòdul p, i lta és la llista dels divisors primers de pp-1;
{False, g}, si pp és compost i on g detecta que pp és compost; per exemple, proporciona g=2 si pp és un nombre parell diferent de 2; en general, proporciona un valor de g d'ordre no divisor de de pp-1, si pp és compost;
Indeterminate, si no pot certificar que pp és primer en el nombre de passos demanat.

Notem que l'algoritme pot no acabar si no proporcionem la llista dels divisors primers de pp-1, perquè intenta calcular-la.
D'altra banda, si la llista és no buida, suposa que és la llista dels divisors primers de p-1; si no ho és, el resultat pot ésser erroni. (No es fa cap intent de comprovar la llista.)

Exercici 5

La funció següent és una implementació d'una funció per a la comprovació del certificat de primeritat que proporciona l'exercici anterior. Es demana esbrinar i explicar com funciona.

In[18]:=
```
Comprova[pp_, cert_] := Module[{p = pp, g = cert[[2]], lta = cert[[3]], x, q, d},
  If[p == 2, Return[cert == {True, 1, {1}}], Null];
  If[p == 3, Return[cert == {True, 2, {2}}], Null];
  x = Product[lta[[i]], {i, 1, Length[lta]}];
  If[Mod[p - 1, x], Return[False], Null];
  q = (p - 1) / x;
  d = GCD[q, x];
  While[d > 1, q = q / d; d = GCD[q, x]];
  If[q ≠ 1, Return[False], Null];
  If[PowerMod[g, p - 1, p] ≠ 1, Return[False], Null];
  If[MemberQ[Table[PowerMod[g, (p - 1) / lta[[i]], p ], {i, 1, Length[lta]}], 1],
   Return[False], Return[True]]
 ]
```

La funció s'aplica a un nombre pp i una llista cert.

Proporciona
True, si cert és un certificat que pp és primer;
False, en cas contrari (o bé pp no és primer, o bé ho és però cert no n'és un certificat).

Exercici 6

Es demana modificar les funcions SolovayStrassen i MillerRabin dels exercicis anteriors a fi que proporcionin un certificat que el nombre és compost, si ho és i el test ho detecta.

In[19]:=

```
SolovayStrassenCert[nn_, ff_] := Module[{n = Abs[nn], f, g, x, y},
  If[Not[IntegerQ[n]], Print["Cal que n sigui un nombre enter."]; Return[Null], Null];
  If[ff < 1, Print["Cal fer alguna prova."]; Return[Null], Null];
  If[n == 2 || n == 3 || n == 5 || n == 7, Return[True], Null];
  If[n == 1, Return[{False, 1}], Null];
  If[EvenQ[n], Return[{False, 2}], Null];
  f = 0;
  While[f < ff,
   g = Random[Integer, {2, n - 2}];
   x = PowerMod[g, (n - 1) / 2, n];
   If[x == 1 || x == n - 1, y = Mod[JacobiSymbol[g, n], n];
    If[x ≠ y, Return[{False, g}], Null], Return[{False, g}]];
   f++];
  Return[Indeterminate]
 ]
```

In[20]:=
```
MillerRabinCert[nn_, ff_] := Module[{n = Abs[nn], v, m, f, g, x, k},
  If[Not[IntegerQ[n]], Print["Cal que n sigui un nombre enter."]; Return[Null], Null];
  If[ff < 1, Print["Cal fer alguna prova."]; Return[Null], Null];
  If[n == 2 || n == 3 || n == 5 || n == 7, Return[True], Null];
  If[n == 1, Return[{False, 1}], Null];
  If[EvenQ[n], Return[{False, 2}], Null];
  v = 0; m = n - 1; While[EvenQ[m], v++; m = m / 2];
  f = 0;
  While[f < ff,
   g = Random[Integer, {2, n - 2}];
   x = PowerMod[g, m, n];
   If[x ≠ 1 && x ≠ n - 1,
    k = 1;
    x = PowerMod[x, 2, n];
    While[x ≠ n - 1 && k < v - 1, x = PowerMod[x, 2, n]; k++];
    If[k ≥ v - 1 && x ≠ n - 1, Return[{False, g}], Null], Null];
   f++];
  Return[Indeterminate]
 ]
```

Exercici 7

Es proposa la implementació d'una funció per a comprovar els certificats que proporcionin les funcions de l'exercici anterior. No en donarem solució.

In[21]:=
```

```

3. Construcció certificada de nombres primers de mida prefixada

Es tracta de construir nombres primers (i certificar que ho són) de la mida que vulguem; per exemple, una parella de nombres primers, d'entre 508 i 512 bits cadascun, a fi que el seu producte sigui un nombre natural d'entre 1016 i 1024 bits.

Exercici 8

Suposem que la funció PrimeQ de *Mathematica* certifica els nombres primers de 15 xifres binàries o menys.
Es demana construir una taula amb 10 nombres primers de 112 bits i certificar-los. (No cal escriure els certificats.)

In[22]:=
```
lp112 = Table[Module[{n, q, fita = 500}, n = 0;
   While[SolovayStrassen[q = 2 * Random[Integer, {2^110, 2^111 - 1}] + 1, 25] == False && n < fita,
    n++]; If[n < fita && Certificat[q, {}, 50][[1]] == True, q, 0]], {i, 1, 10}]
```

Out[22]=
```
{3 039 533 704 730 774 443 973 740 128 862 559, 4 328 562 366 497 619 986 661 709 352 984 983,
 2 637 952 242 395 485 168 075 533 603 647 507, 3 508 116 282 987 403 395 128 596 643 648 971,
 3 608 991 486 892 389 638 320 507 600 161 781, 4 995 843 885 904 312 923 397 626 087 183 733, 2 629 999 761 528 192 653 367 795 116 605 843,
 5 131 313 732 358 696 780 046 487 442 190 211, 2 917 806 757 850 496 310 150 701 367 680 803, 3 830 865 752 607 052 844 371 360 918 761 457}
```

Observació

Notem que el producte de quatre d'aquests nombres primers és un nombre d'aproximadament 448 bits, posem n1. Llavors, per a nombres naturals k en un cert interval, que s'ha de calcular, tots els nombres p=2 k n1 +1 són senars i del nombre de bits que es desitja.
Ara, donat un d'aquests nombres p, per a saber si és primer, es pot utilitzar un test de primeritat, de manera que si p és compost, no passarà el test.
Finalment, com que el nombre k és prou petit, es pot calcular fàcilment la seva descomposició en factors primers, de manera que podem conèixer fàcilment la llista de divisors primers de p-1 i, en conseqüència, serà senzill certificar els nombres p que siguin primers.

Exercici 9

Es demana construir, amb alguns dels nombres primers calculats a l'exercici anterior, una parella de nombres primers, p, q, de 512 bits cadascun, i de manera que p-1 i q-1 siguin divisibles per algun nombre primer de 480 bits (diferent per a p-1 i q-1). Es demana, també, certificar tots els nombres primers que es facin servir.
(Notem que cal usar l'observació anterior un mínim de dues vegades per a cadascun dels nombres primers p que es volen construir; una, per a construir un nombre primer de més de 480 bits que cal que divideixi p-1; i una altra per a la construcció de p.)
Es poden usar (o no) les indicacions posteriors.)

Començarem per calcular nombres primers de 480 bits.

Calculem dos productes de quatre dels nombres anteriors (més endavant necessitarem saber quins fem servir per a cada producte).

In[23]:=
```
{n1 = Product[lp112[[i]], {i, 1, 4}], n2 = Product[lp112[[i]], {i, 7, 10}]}
```

Out[23]=
```
{121 756 330 837 974 762 077 206 536 499 417 186 472 724 673 989 747 177 703 511 602 077 529 017 068 140 315 878 872 486 030 194 372 141 435 444 570 579 978 ↘
   725 374 124 257 299 609,
 150 847 367 831 260 302 873 024 984 521 455 932 457 822 874 183 360 989 275 602 763 864 981 898 852 009 672 439 460 533 850 225 236 615 655 795 320 003 520 ↘
   971 804 640 398 246 683}
```

Calculem els intervals on hem de cercar els valors de k a fi de trobar nombres de 480 bits. Notem que els nombres que hem calculat són de 448 bits, aproximadament.

In[24]:=
```
{x11 = Floor[2^478 / n1] + 1, x12 = Floor[(2^479 - 1) / n1],
 x21 = Floor[2^478 / n2] + 1, x22 = Floor[(2^479 - 1) / n2]}
```

Out[24]=
```
{6 409 827 992, 12 819 655 983, 5 173 687 475, 10 347 374 949}
```

Calculem valors aleatoris de k.

In[25]:=
```
{kq1 = Random[Integer, {x11, x12}], kq2 = Random[Integer, {x21, x22}]}
```

Out[25]=
```
{6 929 644 346, 5 251 923 650}
```

Calculem les llistes dels divisors primers dels nombres k que, com que són petits, podem factoritzar.

In[26]:=
```
{f1kq1 = Transpose[FactorInteger[kq1]][[1]], f1kq2 = Transpose[FactorInteger[kq2]][[1]]}
```

Out[26]=
```
{{2, 17, 1249, 163 181}, {2, 5, 103, 479, 2129}}
```

Calculem la llista dels divisors primers de q-1, per als dos nombres que cerquem.

In[27]:=
```
{f0q1 = Table[lp112[[i]], {i, 1, 4}], f0q2 = Table[lp112[[i]], {i, 7, 10}]};
```

In[28]:=
```
{fq1 = Union[{2}, f1kq1, f0q1], fq2 = Union[{2}, f1kq2, f0q2]}
```

Out[28]=
```
{{2, 17, 1249, 163 181, 2 637 952 242 395 485 168 075 533 603 647 507,
   3 039 533 704 730 774 443 973 740 128 862 559, 3 508 116 282 987 403 395 128 596 643 648 971, 4 328 562 366 497 619 986 661 709 352 984 983},
 {2, 5, 103, 479, 2129, 2 629 999 761 528 192 653 367 795 116 605 843, 2 917 806 757 850 496 310 150 701 367 680 803,
   3 830 865 752 607 052 844 371 360 918 761 457, 5 131 313 732 358 696 780 046 487 442 190 211}}
```

Calculem els dos nombres q.

In[29]:=
```
{q1 = 2 * kq1 * n1 + 1, q2 = 2 * kq2 * n2 + 1}
```

Out[29]=
```
{1 687 456 139 162 154 504 238 018 982 254 857 877 071 888 440 655 489 583 885 192 875 363 901 613 554 352 073 329 365 487 348 200 832 381 836 081 585 031 895 ↓
   030 178 173 788 595 369 469 721 429,
 1 584 477 717 306 490 387 930 005 726 498 236 688 216 085 160 869 136 032 127 869 047 095 727 682 805 554 897 427 111 941 939 246 956 017 217 263 401 371 619 ↓
   750 183 548 174 594 345 963 505 901}
```

Mitjançant el test de Solovay-Strassen o el de Miller-Rabin, determinem si els nombres són o no primers; si no ho són (com probablement serà el cas), caldrà repetir la tria del valor de k, el càlcul dels seus factors primers, i el càlcul del nou valor de q, fins a trobar-ne un que sigui primer (d'acord amb el test).

In[30]:=
```
SolovayStrassen[q1, 50]
```

Out[30]=
```
False
```

No podem passar-vos la vida amb aquest càlcul; per tant, provarem un màxim de 500 valors aleatoris de k.

In[31]:=
```
fita = 500
```

Out[31]=
```
500
```

Ara, iterem el càlcul de k a fi que q sigui primer.

In[32]:=
```
While[fita > 0 && SolovayStrassen[q1, 25] == False,
  kq1 = Random[Integer, {x11, x12}]; q1 = 2 * kq1 * n1 + 1; fita = fita - 1];
If[fita == 0, 0, q1]
```

Out[33]=
```
2 133 012 908 707 561 481 576 819 493 767 938 499 340 231 920 472 845 794 822 119 024 065 874 694 837 324 394 056 878 570 784 474 519 594 223 651 008 465 981 ↘
  089 283 504 873 371 978 714 614 777
```

Mirem si el valor de q sembla primer. (Com que hauria de ser-ho, passarem el test de Miller-Rabin en comptes del de Solovay-Strassen.)

In[34]:=
```
MillerRabin[q1, 50]
```

Out[34]=
```
Indeterminate
```

Ara, ho repetim tot per a trobar el segon valor de q.

In[35]:=
```
SolovayStrassen[q2, 50]
```

Out[35]=
```
False
```

In[36]:=
```
fita = 500
```

Out[36]=
```
500
```

In[37]:=
```
While[fita > 0 && SolovayStrassen[q2, 25] == False,
  kq2 = Random[Integer, {x21, x22}]; q2 = 2 * kq2 * n2 + 1; fita = fita - 1];
If[fita == 0, 0, q2]
```

Out[38]=
```
1 604 354 219 081 260 227 719 787 336 795 996 237 145 658 154 386 420 560 431 133 852 969 399 062 675 988 003 426 403 889 710 289 954 666 733 422 679 290 432 ↘
  333 329 060 233 294 859 805 838 217
```

In[39]:=
```
MillerRabin[q2, 50]
```
Out[39]=
```
Indeterminate
```

Escrivim els valors de k.

In[40]:=
```
{kq1, kq2}
```
Out[40]=
```
{8 759 351 132, 5 317 806 476}
```

Calculem els factors primers dels nombres k.

In[41]:=
```
{fkq1 = Transpose[FactorInteger[kq1]][[1]], fkq2 = Transpose[FactorInteger[kq2]][[1]]}
```
Out[41]=
```
{{2, 7, 103, 433 889}, {2, 19 913, 66 763}}
```

Calculem les llistes de divisors primers de q-1, per als dos valors de q.

In[42]:=
```
{fq1 = Union[{2}, Transpose[FactorInteger[kq1]][[1]], f0q1],
 fq2 = Union[{2}, Transpose[FactorInteger[kq2]][[1]], f0q2]}
```
Out[42]=
```
{{2, 7, 103, 433 889, 2 637 952 242 395 485 168 075 533 603 647 507, 3 039 533 704 730 774 443 973 740 128 862 559,
  3 508 116 282 987 403 395 128 596 643 648 971, 4 328 562 366 497 619 986 661 709 352 984 983},
 {2, 19 913, 66 763, 2 629 999 761 528 192 653 367 795 116 605 843, 2 917 806 757 850 496 310 150 701 367 680 803,
  3 830 865 752 607 052 844 371 360 918 761 457, 5 131 313 732 358 696 780 046 487 442 190 211}}
```

Certifiquem els nombres primers q. Notem que coneixem les llistes dels divisors primers de q-1.

In[43]:=
```
{Certificat[q1, fq1, 500], Certificat[q2, fq2, 500]}
```
Out[43]=
```
{{True,
  1 369 511 902 471 892 891 582 176 559 268 397 397 938 780 051 949 507 840 464 673 560 287 549 490 950 914 256 055 616 420 785 338 939 694 182 679 485 540 ∴
   532 410 733 291 410 622 418 531 535 817, {2, 7, 103, 433 889, 2 637 952 242 395 485 168 075 533 603 647 507,
   3 039 533 704 730 774 443 973 740 128 862 559, 3 508 116 282 987 403 395 128 596 643 648 971, 4 328 562 366 497 619 986 661 709 352 984 983}}, {True,
  701 396 986 490 190 455 633 562 614 098 680 479 015 892 891 331 150 315 990 853 899 234 701 338 681 696 614 336 742 116 934 828 783 795 423 661 651 070 842 ∴
   062 328 635 051 950 721 191 550 594, {2, 19 913, 66 763, 2 629 999 761 528 192 653 367 795 116 605 843,
   2 917 806 757 850 496 310 150 701 367 680 803, 3 830 865 752 607 052 844 371 360 918 761 457, 5 131 313 732 358 696 780 046 487 442 190 211}}}
```

Comprovem que, efectivament, els nombres q són de la mida adequada (de fet, no caldria...).

In[44]:=
```
2^479 < q1 && q1 < 2^480 - 1 && 2^479 < q2 && q2 < 2^480 - 1
```

Out[44]=
```
True
```

Amb aquests nombres q, construïm els p de la mateixa manera que abans hem fet els q. Comencem per calcular els intervals adequats per als nous valors de k.

In[45]:=
```
{y11 = Floor[2^510 / q1] + 1, y12 = Floor[(2^511 - 1) / q1],
 y21 = Floor[2^510 / q2] + 1, y22 = Floor[(2^511 - 1) / q2]}
```

Out[45]=
```
{1 571 463 525, 3 142 927 048, 2 089 284 239, 4 178 568 476}
```

Calculem els valors de k.

In[46]:=
```
{kp1 = Random[Integer, {y11, y12}], kp2 = Random[Integer, {y21, y22}]}
```

Out[46]=
```
{1 820 826 600, 2 715 673 716}
```

Calculem la llista dels divisors primers de k.

In[47]:=
```
{fkp1 = Transpose[FactorInteger[kp1]][[1]], fkp2 = Transpose[FactorInteger[kp2]][[1]]}
```

Out[47]=
```
{{2, 3, 5, 3 034 711}, {2, 3, 7, 233, 571}}
```

Calculem la llista dels divisors primers dels valors de p-1.

In[48]:=
```
{fp1 = Union[{2}, fkp1, {q1}], fp2 = Union[{2}, fkp2, {q2}]}
```

Out[48]=
```
{{2, 3, 5, 3 034 711,
  2 133 012 908 707 561 481 576 819 493 767 938 499 340 231 920 472 845 794 822 119 024 065 874 694 837 324 394 056 878 570 784 474 519 594 223 651 008 465 ↘
   981 089 283 504 873 371 978 714 614 777}, {2, 3, 7, 233, 571,
  1 604 354 219 081 260 227 719 787 336 795 996 237 145 658 154 386 420 560 431 133 852 969 399 062 675 988 003 426 403 889 710 289 954 666 733 422 679 290 ↘
   432 333 329 060 233 294 859 805 838 217}}
```

Calculem els valors dels p.

In[49]:=
```
{p1 = 2 * kp1 * q1 + 1, p2 = 2 * kp2 * q2 + 1}
```

Out[49]=
```
{7 767 693 284 636 199 133 580 965 755 302 393 293 525 553 461 932 084 401 820 513 174 770 369 593 253 365 859 055 292 829 308 708 144 598 767 260 210 663 367 ∖
   124 928 761 229 330 661 076 408 789 429 336 401,
 8 713 805 167 825 368 137 149 602 167 293 053 270 502 733 426 776 544 846 569 639 665 173 611 173 648 434 490 472 805 967 372 794 569 254 558 991 097 734 649 ∖
   235 796 559 309 079 357 705 239 418 510 408 745}
```

Són primers?

In[50]:=
```
SolovayStrassen[p1, 50]
```

Out[50]=
```
False
```

Iterem el càlcul.

In[51]:=
```
fita = 1000
```

Out[51]=
```
1000
```

In[52]:=
```
While[fita > 0 && MillerRabin[p1, 25] == False,
 kp1 = Random[Integer, {y11, y12}]; p1 = 2 * kp1 * q1 + 1; fita = fita - 1]
```

In[53]:=
```
fita
```

Out[53]=
```
956
```

Tenim el valor d'un dels nombres primers. (Excepte si fita val 0, cas en què hem de repetir el càlcul anterior i la iteració.)

In[54]:=
```
p1
```

Out[54]=
```
10 962 557 935 735 849 906 299 622 502 611 332 796 712 117 104 378 931 158 611 931 807 004 008 506 169 285 764 752 114 307 703 168 344 866 055 751 610 764 040 ∖
 091 737 437 932 249 839 855 452 433 412 212 353
```

Calculem la llista de divisors primers de p-1.

In[55]:=
```
fp1 = Union[{2}, Transpose[FactorInteger[kp1]][[1]], {q1}]
```

Out[55]=
```
{2, 3, 73, 183 343,
 2 133 012 908 707 561 481 576 819 493 767 938 499 340 231 920 472 845 794 822 119 024 065 874 694 837 324 394 056 878 570 784 474 519 594 223 651 008 465 981 ⁖
  089 283 504 873 371 978 714 614 777}
```

Escrivim un certificat per a p.

In[56]:=
```
Certificat[p1, fp1, 500]
```

Out[56]=
```
{True,
 7 679 814 616 889 248 389 098 293 025 897 779 888 396 595 402 162 330 532 838 867 015 071 524 144 556 691 129 817 196 813 068 690 690 208 448 668 411 241 406 ⁖
  925 027 090 372 893 924 499 920 848 799 251 812, {2, 3, 73, 183 343,
  2 133 012 908 707 561 481 576 819 493 767 938 499 340 231 920 472 845 794 822 119 024 065 874 694 837 324 394 056 878 570 784 474 519 594 223 651 008 465 ⁖
   981 089 283 504 873 371 978 714 614 777}}
```

Comprovem que p és de la mida adequada (no caldria...).

In[57]:=
```
2^511 < p1 && p1 < 2^512 - 1
```

Out[57]=
```
True
```

Ara, repetim el mateix per a trobar un segon valor de p.

In[58]:=
```
SolovayStrassen[p2, 50]
```

Out[58]=
```
False
```

In[59]:=
```
fita = 1000
```

Out[59]=
```
1000
```

In[60]:=
```
While[fita > 0 && MillerRabin[p2, 25] == False,
 kp2 = Random[Integer, {y21, y22}]; p2 = 2 * kp2 * q2 + 1; fita = fita - 1]
```

In[61]:=
```
fita
```

Out[61]=
```
822
```

In[62]:=
```
p2
```

Out[62]=
```
9 271 271 145 490 108 525 994 381 047 554 729 611 685 349 616 028 464 279 693 281 058 403 751 048 555 465 778 599 772 699 270 495 096 126 440 112 185 097 674 ↓
 428 792 507 198 966 510 550 915 140 672 828 323
```

In[63]:=
```
fp2 = Union[{2}, Transpose[FactorInteger[kp2]][[1]], {q2}]
```

Out[63]=
```
{2, 7, 31, 13 315 249,
 1 604 354 219 081 260 227 719 787 336 795 996 237 145 658 154 386 420 560 431 133 852 969 399 062 675 988 003 426 403 889 710 289 954 666 733 422 679 290 432 ↓
  333 329 060 233 294 859 805 838 217}
```

In[64]:=
```
Certificat[p2, fp2, 500]
```

Out[64]=
```
{True,
 2 652 964 096 065 810 595 535 472 843 506 857 911 911 265 985 922 387 896 180 160 395 682 270 362 065 868 670 464 643 577 256 640 805 682 085 282 906 994 238 ↓
  428 704 755 887 471 196 482 042 942 286 514 614, {2, 7, 31, 13 315 249,
  1 604 354 219 081 260 227 719 787 336 795 996 237 145 658 154 386 420 560 431 133 852 969 399 062 675 988 003 426 403 889 710 289 954 666 733 422 679 290 ↓
   432 333 329 060 233 294 859 805 838 217}}
```

Exercici 10

A partir de la parella de nombres primers p, q que hem calculat a l'exercici anterior, es demana construir una parella de claus (privada-pú blica) per a un criptosistema RSA de 1024 bits.

In[65]:=
```
e = 2 * Random[Integer, {2^1022, 2^1023 - 1}] + 1
```

Out[65]=
```
128 251 736 477 792 679 208 027 437 477 019 774 659 072 437 792 163 989 883 971 159 873 551 881 977 954 093 072 098 143 089 403 912 202 716 877 848 113 692 586 ↓
 684 289 990 439 090 397 413 761 826 522 576 009 665 839 988 535 621 287 649 617 857 902 626 652 402 552 472 879 539 237 881 468 223 714 760 226 562 078 659 ↓
 834 185 571 143 024 142 315 632 875 985 612 925 691 594 035 616 974 078 196 620 634 995 330 505
```

In[66]:=
```
d = PowerMod[e, -1, (p1 - 1) (p2 - 1) / GCD[p1 - 1, p2 - 1]]
```

Out[66]=
```
49 276 947 500 712 420 263 126 543 999 217 009 958 411 255 544 955 381 146 219 345 119 277 424 767 298 976 824 406 482 011 992 299 310 403 612 144 429 971 139 ↓
 261 347 529 310 328 683 447 332 364 779 190 158 796 624 342 534 093 778 736 863 680 804 907 086 067 525 364 887 281 368 294 211 480 288 807 147 622 034 185 ↓
 421 324 323 743 300 579 030 405 622 991 359 241 453 491 149 333 009 246 403 896 282 002 981 497
```

Observació: Si e no és invertible, cal repetir la tria fins que ho sigui.

In[67]:=
```
{ClauRSAPublica = {p1 * p2, e}, ClauRSAPrivada = {p1 * p2, d}}
```

Out[67]=
```
{{101 636 847 070 351 392 689 338 145 864 067 461 752 990 817 556 043 790 267 167 780 230 202 455 093 409 057 278 194 835 806 268 217 409 438 236 650 867 373 ↓
    656 471 992 531 977 596 209 105 844 494 225 027 324 580 695 325 076 664 815 826 960 538 269 061 378 279 826 145 426 600 373 591 833 294 109 506 021 038 027 ↓
    291 973 990 349 829 192 932 201 913 320 665 853 493 965 490 658 738 890 299 341 732 425 188 874 019,
   128 251 736 477 792 679 208 027 437 477 019 774 659 072 437 792 163 989 883 971 159 873 551 881 977 954 093 072 098 143 089 403 912 202 716 877 848 113 692 ↓
    586 684 289 990 439 090 397 413 761 826 522 576 009 665 839 988 535 621 287 649 617 857 902 626 652 402 552 472 879 539 237 881 468 223 714 760 226 562 078 ↓
    659 834 185 571 143 024 142 315 632 875 985 612 925 691 594 035 616 974 078 196 620 634 995 330 505},
  {101 636 847 070 351 392 689 338 145 864 067 461 752 990 817 556 043 790 267 167 780 230 202 455 093 409 057 278 194 835 806 268 217 409 438 236 650 867 373 ↓
    656 471 992 531 977 596 209 105 844 494 225 027 324 580 695 325 076 664 815 826 960 538 269 061 378 279 826 145 426 600 373 591 833 294 109 506 021 038 027 ↓
    291 973 990 349 829 192 932 201 913 320 665 853 493 965 490 658 738 890 299 341 732 425 188 874 019,
   49 276 947 500 712 420 263 126 543 999 217 009 958 411 255 544 955 381 146 219 345 119 277 424 767 298 976 824 406 482 011 992 299 310 403 612 144 429 971 ↓
    139 261 347 529 310 328 683 447 332 364 779 190 158 796 624 342 534 093 778 736 863 680 804 907 086 067 525 364 887 281 368 294 211 480 288 807 147 622 034 ↓
    185 421 324 323 743 300 579 030 405 622 991 359 241 453 491 149 333 009 246 403 896 282 002 981 497}}
```

4. Claus ElGamal

Definició

Una parella de claus privada pública per a un criptosistema ElGamal consta de nombres naturals p, q, g, x, h, de manera que:
p és un nombre primer;
q és un nombre primer que divideix p-1;
g és un nombre natural invertible mòdul p i d'ordre exactament q en G(p);
x és un nombre natural;
h és la potència x-è sima de g mòdul p.
La clau privada és la llista {p, q, g, x } i la clau pública és la llista {p, q, g, h}.

Exercici

Es demana construir un nombre primer p de 1024 bits tal que p-1 sigui divisible per un nombre primer q de 1000 bits.

(Cal certificar tots els nombres primers que es facin servir i que siguin de més de 15 xifres binàries.)

Es demana utilitzar aquesta parella de nombres primers p, q per a construir una parella de claus privada-pú blica per al criptosistema ElGamal.