# Iniciació a la Criptografia

**Artur Travesa**

**(versió 2024-07)**

# Capítol 10. RSA

## 10.0. Introducció

La sigla RSA prové de les inicials dels cognoms de Ronald L. Rivest, Adi Shamir i Leonard Adleman que, el 1977, van fer la primera descripció practicable d'un criptosistema de clau pública, conegut des de ben aviat per RSA (cf. la referència **[RSA]**).

La seguretat del criptosistema proposat per Rivest, Shamir i Adleman es basa en la dificultat (conjectural) de factoritzar nombres enters arbitraris. En efecte, donats dos nombres primers $p$, $q$, el càlcul del seu producte, $N = p \cdot q$, es pot fer en temps polinòmic en la longitud en bits de $p$ i $q$; en canvi, no es coneix cap algoritme tal que, donat $N$, trobi els nombres $p$ i $q$ en temps polinòmic en la longitud en bits de $N$ (el millor algoritme que es coneix actualment és subexponencial).

**Observació**. Notem que parlem d'algoritmes *clàssics*, perquè actualment ja hi ha algoritmes quàntics polinòmics per a resoldre aquest problema.

A continuació proporcionem una descripció i, simultàniament, un exemple de criptosistema de tipus RSA.

## 10.1. Les claus

Com succeeix en qualsevol criptosistema de clau pública, cada usuari d'un criptosistema RSA disposa de dues claus: la clau pública, coneguda (o cognoscible) per tothom, i la clau privada, només cognoscible (i coneguda) per l'usuari propietari de la clau.

Tant les claus públiques com les claus privades d'un criptosistema RSA són parelles de nombres naturals.

### 10.1.0. La clau pública

Una clau pública per a un criptosistema de tipus RSA és una parella $(N, e)$ en la qual $N$, que s'anomena el mòdul de la clau, és el producte de dos o més nombres naturals primers senars diferents, $p_1, \ldots, p_r$ (doncs, $r \geq 2$). Per a descriure l'altre element, $e$, de la clau, que s'anomena l'exponent de la clau pública, considerem $M := mcm(p_1 - 1, \ldots, p_r - 1)$, el mínim comú múltiple dels nombres $p_1 - 1, \ldots, p_r - 1$. El nombre $e$ és un nombre natural menor que $M$ i sense factors comuns amb $M$; és a dir, $e$ és un nombre natural tal

que $1 \leq e \leq M$ i que $mcd(e, M) = 1$ (o sigui, invertible mòdul $M$).

#### 10.1.0.0. Observació

La clau pública només conté els nombres $N$ i $e$ i no ha de proporcionar cap informació addicional sobre els nombres primers $p_1, \ldots, p_r$, ni tampoc sobre el nombre $M$.

#### 10.1.0.1. Observació

En moltes descripcions del criptosistema RSA es canvia el mínim comú múltiple, $M$, dels nombres $p_i - 1$, pel seu producte, $(p_1 - 1) \cdots (p_r - 1) = \varphi(N)$, que coincideix amb el valor en $N$ de la funció phi d'Euler. Tot i que és possible usar aquest valor $\varphi(N)$ en lloc de $M$, per qüestions de seguretat convé usar $M$, tal com hem afirmat més amunt (cf. **[RSA,** secció IX, C**]**, o bé **[RSA-S,** secció 3.2, p. 7**]**).

## 10.1.1. La clau privada

La clau privada associada a la clau pública $(N, e)$ és la parella $(N, d)$, on $d$ és l'únic nombre natural menor que $M$ i tal que $ed \equiv 1 \pmod{M}$; s'anomena l'exponent de la clau privada.

Notem que, coneguts $M$ i $e$, el càlcul de $d$ es pot fer fàcilment amb l'algoritme d'Euclides. És per això que la clau pública no ha de donar cap informació que permeti calcular els nombres primers dels quals $N$ és producte.

## 10.1.2. Restriccions a les claus

#### 10.1.2.0. Mida dels factors

A fi que les claus d'un criptosistema RSA siguin criptogràficament útils, en el sentit que proporcionin una certa seguretat de privacitat, cal que el problema de la descomposició de $N$ com a producte de nombres primers sigui "computacionalment intractable". En l'actualitat (juliol de 2024), encara sembla prou segur utilitzar nombres $N$ de l'ordre de $1024$ bits que siguin producte de dos nombres primers d'aproximadament $512$ bits cadascun, i que satisfacin algunes altres restriccions que detallarem més avall. Els darrers estàndards, però, ja recomanen usar claus de $2048$ bits o, fins i tot, de $4096$ bits.

#### 10.1.2.1. Observació

Si el nombre $N$ és producte de més de dos nombres primers de mides similars, aquests són més petits que si és producte de només dos factors de mida similar, de manera que, *potser*, la factorització de $N$ és més senzilla.

Això fa que, en la majoria de les implementacions de criptosistemes de tipus RSA, es consideri que els mòduls $N$ són producte de **dos** nombres primers senars diferents, $p$, $q$. En els exemples següents, tindrem en compte aquesta restricció.

Una possible construcció de nombres primers de mides prefixades es detalla en **[Tr-PrF**, cap. 3**]**.

#### 10.1.2.2. Restricció sobre els factors primers del mòdul

Una altra de les restriccions que cal tenir en compte per als nombres primers $p_i$ és que cal triar-los de manera aleatòria entre els nombres primers $p$ per als quals $p - 1$ és divisible per un nombre primer gran. *Per exemple*, per a un mòdul $N$ de $1024$ bits, es poden prendre nombres primers $p$ de $508$ a $516$ bits i de manera que $p - 1$ sigui divisible per algun nombre primer d'aproximadament $480$ bits.

#### 10.1.2.3. Restricció sobre l'exponent secret

Un cop triats els nombres primers $p$ i $q$, cal triar els nombres $d$ i $e$. La manera més adient de fer-ho (tot i que en alguns estàndards no es fa així!) és triar primerament $d$ i calcular $e$ a continuació. Per a evitar els atacs a la clau per força bruta, cal que $d$ no sigui previsible; en particular, no pot ésser gaire petit, perquè un criptoanalista podria provar tots els valors menors que una certa fita com a valors de $d$ fins a trobar-lo. Notem que, com que $p$ i $q$ són senars, el nombre $M$ és parell, de manera que $d$ i $e$ també són nombres senars.

#### 10.1.2.4. Observació sobre la tria dels exponents

Molt sovint, a la pràctica, es pren a l'atzar un nombre enter $d'$, de la mateixa mida aproximada que $M$, i se cerca el menor nombre enter $d > d'$ que sigui invertible mòdul $M$. O bé es pren com a valor de $d$ un nombre primer triat a l'atzar entre els nombres primers de la mateixa mida aproximada que $M$. I, a continuació, un cop triat $d$, es calcula $e$.

#### 10.1.2.5. Observació sobre el càlcul

Notem que el càlcul de la clau privada és senzill per al creador de la parella de claus, perquè coneix els valors $p_1, \ldots, p_r$. En canvi, no es coneix cap algoritme polinòmic per a poder calcular $d$ només a partir de $N$ i $e$; és a dir, de la clau pública. (Cf. però, més avall, la secció 10.5.)

#### 10.1.2.6. Simetria

Els nombres $d$ i $e$ de les claus del criptosistema RSA juguen un paper idèntic; tots dos són nombres naturals i actuen com a tals en els processos de xifratge i de desxifratge; de fet,

els nombres $d$ i $e$ són intercanviables. Això permet definir de manera molt senzilla algoritmes de signatura digital per als criptosistemes del tipus RSA.

### 10.1.3. Exemple de claus

Per a la resta d'aquesta descripció d'un criptosistema bàsic de tipus RSA, farem servir els valors següents de $p$, $q$, $N$, $e$ i $d$. Els nombres p0 i q0 són primers, tot i que no ho demostrarem. Comprovarem que podem calcular de manera senzilla N0:=p0·q0, i ema0:=mcm(p0-1,q0-1); i comrpvarem que per als valors e0 i d0 se satisfà que e_0d_0 ≡ 1 (mod ema0).

En particular, podrem escriure les claus pública i privada, i separar-ne els components, si convé.

In [1]:
```
1 p0 =1022339425069665774554704954139599512068441858482446680258497
```

In [2]:
```
1 q0=1145252227809332120648503566349584893170124250353616927857369
```

In [3]:
```
1 N0 = p0 * q0
```

In [4]:
```
1 e0=161352920771928245259105800590639398633907036856952523034577
```

In [5]:
```
1 d0=873609968216332119104885243296174908320584727631951062640237
```

In [6]:
```
1 ClauRSAPublica0=[N0,e0]
```

In [7]:
```
1 print(ClauRSAPublica0)
```

```
[11708365041383464933686602942455730985162648443781485491922389315722848100416841000536154729129480172461467154428548587283466853859471856428055372381517452641678917928981671254022173877166264998051552833100824075739103226848898756793867781981039606204676185641504614272955052833759647915179991686524110141761, 16135292077192824525910580059063939863390703685695252303457775545135311289135602375966530819235400026253694244894872507377475652597091159698183955143633066487355631062502424090397118919830508808096152196753458194543517481691880784596340479817001784873948706528044510367780820727472216676383823634399201850910 3]
```

In [8]:
```
1 ClauRSAPrivada0=[N0,d0]
```

In [9]:
```
1 print(ClauRSAPrivada0)
```

```
[11708365041383464933686602942455730985162648443781485491922389315722848100416841000536154729129480172461467154428548587283466853859471856428055372381517452641678917928981671254022173877166264998051552833100824075739103226848898756793867781981039606204676185641504614272955052833759647915179991686524110141761, 87360996821633211910488524329617490832058472763195106264023712639307129497662547308067281720371188072092574553190248509965462743183373544171052520949542203730990526776399064043235279438145565342546167889102390222416203834714022177015360006228220501579144099760233703935250997915034184377946056633145176867]
```

In [10]:
```
1 ema0=lcm(p0-1,q0-1)
```

In [11]:
```
1 print(ema0)
```

```
11708365041383464933686602942455730985162648443781485491922389315722848100416841000536154729129480172461467154428548587283466853859471856428055372381517450474087265049983776050813653387981859759485443997037215959872171929830290945039389076116337116785906276324422405691407967120519607787919538671610392507700
```

In [12]:
```
1 (d0*e0)%ema0==1
```

Out[12]: True

# 10.2. Xifratge i desxifratge de missatges

## 10.2.0. Xifratge

### 10.2.0.0. Missatges que es poden xifrar

Les unitats de missatge que es poden xifrar amb el criptosistema RSA són els elements de $\mathbb{Z}/N\mathbb{Z}$. En particular, el conjunt d'unitats vàlides de missatge és diferent per a cada usuari, perquè cadascun treballa en un anell diferent, que depèn del seu propi valor de $N$.

**Observació**. Aquest fet fa necessari que hi hagi un algoritme estàndard, igual per a tots els usuaris del criptosistema, per a convertir els missatges plans que es volen xifrar en nombres mòdul $N$ i, recíprocament, reconvertir els nombres mòdul $N$ que provenen de desxifrar els missatges xifrats en missatges plans. Això es pot fer de manera semblant a com es tracta el problema de la codificació, sufixació i farciment de missatges per a la criptografia elemental (cf. **[Cr-04** (?)**]**) i aquí no ens hi entretindrem.

De fet, per a entendre bé com funciona el criptosistema RSA no cal xifrar i desxifrar missatges "reals", i és suficient fer-ho amb missatges "adequats a l'objectiu". En particular, ens estalviarem la feina de codificar, sufixar i farcir els missatges per al seu xifratge amb RSA, i suposarem que ja tenim els missatges, que triarem de manera aleatòria, preparats d'aquesta manera.

### 10.2.0.1. El xifratge del missatge

Suposem, per tant, que la unitat de missatge que es vol xifrar per a l'usuari U és un element $m$ de $\mathbb{Z}/N\mathbb{Z}$, on $(N, e)$ és la clau pública de U.

L'usuari V que vol xifrar el missatge $m$ per a U, calcula el representant $0 \leq c \leq N-1$ de la classe $c :\equiv m^e \pmod{N}$; el missatge xifrat associat a $m$ és el nombre natural $c$.

### 10.2.0.2. Observació sobre el càlcul

Notem que aquest càlcul es pot fer molt ràpidament amb un algoritme binari d'exponenciació mòdul $N$ (cf. **[Tr-Ar,** §1.6**]**), i que només cal conèixer les dades $m$, $e$ i $N$.

**Observació**. Notem que, en el nostre exemple pràctic, $c$ és un nombre de, com a màxim, $1024$ bits, perquè $N$ és de $1024$ bits. Per tant, amb un algoritme binari d'exponenciació, haurem de fer, com a màxim, $2048$ multiplicacions mòdul $N$.

### 10.2.1. Desxifratge

Per a desxifrar el missatge, l'usuari U utilitza la seva clau privada $(N, d)$ i calcula el representant $r$ de la classe $c^d \pmod N$ de l'interval $0 \leq r \leq N-1$; això reconstrueix el missatge $m$.

#### 10.2.1.0. Demostració

En efecte, es té que $c^d \pmod N \equiv (m^e)^d \pmod N \equiv m^{e \cdot d} \pmod N$. Però, per a cadascun dels nombres primers $p$ que divideixen $N$, és $m^{e \cdot d} \equiv m \pmod p$; en efecte, podem aplicar el petit teorema de Fermat, perquè $p-1$ divideix $M = mcm(p_1 - 1, \ldots, p_r - 1)$ i $M$ divideix $e \cdot d - 1$. Ara, pel teorema xinès del residu, és $m^{e \cdot d} \equiv m \pmod N$, com calia veure. □

#### 10.2.1.1. Observació

Cal notar que els càlculs que es realitzen per a xifrar el missatge són anàlegs als que es realitzen per a desxifrar-lo; només canvia l'exponent (i, òbviament, la base).

## 10.3. La funció RSA(mc,clau)

La funció RSAX0(mc,clau) pren com a entrades una llista d'unitats de missatge, mc, i una clau pública RSA, clau, i produeix una llista d'unitats de missatge xifrades amb aquesta clau pública.

### 10.3.0 La funció

In [13]:
```
def RSAX0(mc,clau):
    return [mod(mc[i],clau[0])^clau[1] for i in range(len(mc))]
```

### 10.3.1. Inventem-nos un missatge

Ja disposem d'una parella de claus pública i privada per al criptosistema RSA. Són en dues llistes anomenades ClauRSAPublica0 i ClauRSAPrivada0.

Ara, ens inventem un missatge aleatori (de fet, un missatge aleatori de cinc unitats de missatge).

In [14]:
```
m1=[ZZ.random_element(0,N0) for i in range(5)]
```

Vegem-lo (encara que veure'l no serveix per a res!).

In [15]:
```
1 print(m1)
```

```
[68397739446808441817793150095989517863258220580776459874043102968268258475456698811337116331090570980676698554073411876144184415165804510817995667392322473891088244312700767072910063212202602985357030673583232539468822563149352484661417661860789628904297531186244046746777202033773907389154888522228669725 5, 735647712966694468560123925778870363136368499682315769861829372559347655377122383642083610042335032488968705872622877429118398888804407401337526836945106504750966907245506458433046063222661932571848417427971999388980234413465036922365951871668058279255742908520932700300760628478288936323822321546023596630 6, 724687744509721043173720319131450028575055343390405618772740661380602014412097555773661905673420302339245040437947582678037784304385670390417807170971893865425085482601184973739724295271358672217455998598912059499014164299926471975828061474182668723991563308686924181174932344405240488884318893546769975426571269411059838052522844287968829124590323463215977901811281879789398002968095392871662561848356055313953202993105568158974827558317834410270881124208650603616371353587184006582870422964335200967052795651194673615384177750816481832847556353051280358776652956615351918182789256414501837795042708394061003537331 0, 110861607719959646356931754988715768746965461823929170720067652029713123543038904927665296835589784802231217653037548836595974755036595739762674636687824937084982346041217476641595159754327991580234138913713974643636555399615330373347539872843017103033980590437323144745222192892999141588217842990615216507591]
```

## 10.3.2. Xifrem el missatge

I el xifrem amb la clau pública.

In [16]:
```
1 x1=RSAX0(m1,ClauRSAPublica0)
```

Vegem-lo (encara que veure'l tampoc no serveix per a res!).

In [17]:
```
1 print(x1)
```

```
[259809532375174939018905118975204570749476000783218023121515381584
456430413824579774518072792039649204574709474802785520114644044346 9
091849695157743543750560756575916257133400968459051977946779905913 0
538147183886758051211632571234480212757964581231707369626568293235 3
059390820700855348485256718798421758434 34, 632044840427927634046768
090622616208781575969608573719765689992955646376287516744876898198 5
549156554345646716090070255678635612010158531360168474690598733572 3
187486394725891343196914479505768792259916774579605407882493707907 5
057850012115304968851213066711227573703840207149043453544460760452 9
9109699148887092, 498328726487713493161578009578811694301135997202 3
361376152237652343085087110724200349863796972182628145311736845474 7
659563557412045674849754754688581809737399716575102114997637459909 0
505311941446859984063947189229047593350687750917243270394272007756 2
180144207983054712568693097429722163724896950688401585967 3, 1646968
913364175737758491915176539703132993229326026693852331934382997841 1
943620175317722055478403161484123468288319245070899376853594282309
127280439153446549666392285205769086717360152512782289748822643884 6
821661889763109203841742246136880790873390106420197007715808120746 4
929701265800970968773192891727494, 298430851041025486252244246527 45
937560608619526083650360058931230151749379203666227587016469383483 5
666977902325301532484809237058906070337171504547080502778341734347 2
653709250577638837189279506838684253291867609376707015955643842407 7
207241177102522452202373798103986870814849451779985435622215918188 0
93733644]
```

### 10.3.3 Desxifrem el missatge xifrat

Notem que podem desxifrar-lo exactament igual amb la clau privada.

In [18]:
```
1 y1=RSAX0(x1,ClauRSAPrivada0)
```

In [19]:
```
1 print(y1)
```

```
[68397739446808441817793150095989517863258220580776459874043102968268258475456698811337116331090570980676698554073411876144184415165804510817995667392322247389108824431270076707291006321220260298535703067358323225394688225631493524846614176618607896289042975311862440467467772020337739073891548885222286697255, 73564771296669446856012392577887036313636849968231576986182937255934765537712238364208361004233503248896870587262287742911839888880440740133752683694510650475096690724550645843304606322266193257184841742797199938898023441346503692236595187166805827925574290852093270030076062847828893632382232154602359663067246877445097210431737203191314500285750553433904056187727406613806020144120975557736619056734203023392450404379475826780377843043856703904178071709718938654250854826011849737397242952713586722174559985989120594990141642999264719758280614741826687239915633086869241811749323444052404888843188935467699754265712694110598380525228442879688291245903234632159779018112818797893980029680953928716625618483560553139532029931055681589748275583178344102708811242086506036163713535871840065828704229643352009670527956511946736153841777508164818328475563530512803587766529566153519181827892564145018377950427083940610035373310, 11086160771995964635693175498871576874696546182392917072006765202971312354303890492766529683558978480223121765303754883659597475503659573976267463668782493708498234604121747664159515975432799158023413891371397464363655539961533037334753987284301710303398059043732314474522219289299914158821784299061521650759]
```

In [20]:
```
1 y1==m1
```

Out[20]: True

Efectivament, sembla que la funció xifra (i desxifra) correctament.

# 10.4. Consideracions bàsiques sobre la seguretat

A l'hora de dur a la pràctica una implementació d'un criptosistema RSA, cal tenir en compte algunes consideracions.

## 10.4.0. Mida de les claus

En primer lloc, cal tenir en compte la mida que poden tenir les claus. No és el mateix una mida de claus de $128$ o $256$ bits, per exemple, que una mida de claus de $1024$ o $2048$ o $4096$ bits. I no és recomanable que en el mateix criptosistema les mides de les claus siguin molt diferents les unes de les altres. A priori, una clau molt llarga pot proporcionar més seguretat que una clau no tan llarga. El fet que alguns usuaris disposessin de claus llargues, podria fer pensar que el criptosistema és molt segur, de manera que qualsevol altre usuari podria triar una clau molt més curta (per exemple, per a facilitar el procés de xifratge/desxifratge, si disposa de menys capacitat de càlcul); però això produiria un punt feble en el sistema, i els atacs a aquest usuari serien més factibles. Al contrari, si la majoria de les claus fossin d'una determinada mida i un usuari triés una clau molt més llarga, això podria fer pensar que els missatges d'aquest usuari són "més interessants" que els altres (per què, si no, cerca més seguretat?) i podria ésser el blanc dels atacs al criptosistema; ell

o bé els usuaris amb qui es comuniqués.

### 10.4.1. Codificació dels missatges

No només això, sinó que si es limita la llargada de les claus, llavors es pot usar un mètode comú per a codificar els missatges com a elements dels diferents anells $\mathbb{Z}/N\mathbb{Z}$, quan $N$ recorre les diferents claus.

Suposem que la llargada de les claus es fixa entre $n+1$ i $n+k$ bits. Això significa que tots els nombres $N$ de les claus públiques pertanyen a l'interval $2^n \leq N < 2^{n+k}$; llavors, es pot usar una codificació dels missatges plans com a successions de $n$ bits, tothom igual, i una codificació dels missatges xifrats com a successions de $n+k+1$ bits, també tothom igual.

En efecte, tots els nombres naturals de l'interval $0 \leq m < 2^n$ són menors que $N$, per a tot $N$, de manera que cadascun determina un element únic de $\mathbb{Z}/N\mathbb{Z}$.

**Observació**. Notem que, ara, no tots els missatges plans (=nombres) són diferents! Només ho són els nombres de $n$ bits que són menors que $N$, perquè, com a missatges, $m$ i $m+kN$ són el mateix, per a qualsevol nombre enter $k$ que faci que els dos nombres siguin de $n$ bits.

Un cop xifrat el missatge, tindrem un element $x$ de $\mathbb{Z}/N\mathbb{Z}$, que podrem interpretar de manera única com un nombre enter de l'interval $0 \leq x < N$, de manera que proporciona un element de l'interval $0 \leq x < 2^{n+k}$, perquè $N < 2^{n+k}$. Així, els missatges plans poden ésser nombres qualssevol de l'interval $0 \leq m < 2^n$, el mateix per a tothom, i els missatges xifrats seran nombres de l'interval $0 \leq x < 2^{n+k}$, també el mateix per a tothom, però menors que $N$ si el mòdul de la clau és $N$.

### 10.4.2. Longitud dels missatges

Una altra consideració que cal tenir en compte a l'hora de fixar la longitud de les claus, i que és vàlida per a tots els criptosistemes en general, és la longitud efectiva dels missatges plans que s'han d'enviar.

Suposem que els missatges plans que cal enviar són molt curts, posem de menys de $16$ bits (per exemple, els PIN d'un caixer automàtic són nombres naturals de quatre xifres decimals de manera que podem representar-los amb només $14$ bits). No té sentit usar un criptosistema amb claus molt llargues, perquè la quantitat de missatges plans possibles és de $2^{16} = 65536$, una quantitat prou petita com perquè qualsevol pugui xifrar-los tots amb la clau pública del destinatari (coneguda per tothom) i comparar els resultats amb el missatge xifrat que li és destinat. Això determina el missatge pla sense necessitat de trencar el criptosistema ni la clau del destinatari, però fa impossible la confiança en la confidencialitat de les comunicacions. (Això és l'atac anomenat "del diccionari", o també "per força bruta", perquè es fa un "diccionari" dels missatges xifrats que es corresponen als missatges plans.)

En aquests casos, s'imposa la necessitat de farcir els missatges plans abans de xifrar-los, a fi que els conjunts de missatges plans que poden ésser xifrats siguin molt grans i es puguin evitar, d'aquesta manera, els atacs per força bruta.

## 10.5. Una vulnerabilitat del criptosistema RSA

Observem que, com hem comentat més amunt, coneguts $p$ i $q$, es pot calcular $M$ en temps polinòmic (en la longitud de $N$) i, amb l'algoritme d'Euclides, també es pot calcular el nombre $d$, invers de $e$ mòdul $M$, en temps polinòmic.

Recíprocament,

**Teorema:** *existeix un algoritme probabilístic que, coneguts $N$, $e$ i $d$, permet (probablement) factoritzar $N$ en temps polinòmic*.

### 10.5.0. Demostració

Notem que $M$ és parell i que, per tant, $e$ i $d$ són senars, de manera que $ed-1$ és parell. Podem escriure, doncs, el nombre $ed-1$ en la forma $ed-1=2^v s$, amb $v \geq 1$ i $s$ senar, sense cap dificultat, perquè coneixem $e$ i $d$ i sabem dividir per $2$ tants cops com faci falta. Això produeix els nombres $v$ i $s$.

Seguidament, prenem a l'atzar un nombre $m$, $2 \leq m \leq N-2$, i calculem $x_0 := m^s \pmod N$ (de nou, podem fer-ho sense dificultat). Ara, amb l'algoritme d'Euclides, calculem el nombre $t := mcd(x_0-1, N)$. Pot ser que $t$ sigui $1$, $N$, o bé un factor propi de $N$.

Si $t$ és un factor propi de $N$, ja hem factoritzat $N$ i acabem. Si $t=N$, cal canviar el valor de $m$. Finalment, si $t=1$, calculem $x_1 := x_0^2 \pmod N$ i, de nou amb l'algoritme d'Euclides, calculem $mcd(x_1-1, N)$.

I apliquem el mateix algoritme de decisió que més amunt: si és un factor propi de $N$, ja hem factoritzat $N$; si és $N$, canviem el valor de $m$; i si és $1$, podem repetir el procés: calculem $x_2 := x_1^2 \pmod N$ i $mcd(x_2-1, N)$.

Successivament, podrem repetir el procés fins a arribar, com a màxim, al càlcul de $x_v$, perquè es té que $x_v = m^{2^v s} = m^{ed-1} \equiv 1 \pmod N$, de manera que $mcd(x_v-1, N) = N$, i no hauríem aconseguit factoritzar amb aquest valor de $m$.

El fet que $N$ és compost, producte de $r \geq 2$ nombres primers, fa que la probabilitat que un $m$ triat a l'atzar en les condicions anteriors permeti factoritzar $N$ sigui com a mínim $1 - \frac{1}{2^{r-1}} \geq \frac{1}{2}$ (i, notem que aquesta probabilitat és més gran com més gran és la quantitat $r$ de nombres primers que divideixen $N$). Per tant, és d'esperar que només calgui provar uns quants valors de $m$ triats a l'atzar (sovint n'hi ha prou amb tres o quatre). □

### 10.5.1. Observació

Encara que el valor de $m$ es prengui a l'atzar, i diferent de $1$ i de $-1$ mòdul $N$, el valor de $x_0$ pot ser $1$ mòdul $N$, de manera que $mcd(x_0-1, N) = N$ i tampoc no factoritzaríem.

**Exercici**. Notem que no triem cap dels valors $m = 0, 1, N-1$, perquè aquests no serveixen per a factoritzar $N$ per aquest procediment. Per què?

## 10.5.2. Exemple

Suposem, doncs, que coneixem la clau pública $(N, e)$ i la clau privada $(N, d)$. Es tracta de factoritzar $N$.

Els valors següents de $n$, $e$ i $d$ són els de la clau pública [N0,e0] i de la clau privada [N0,d0] de més amunt.

In [21]:
```
n=11708365041383464933686602942455730985162648443781485491922389
```

In [22]:
```
e=16135292077192824525910580059063939863390703685695252303457775
```

In [23]:
```
d =87360996821633211910488524329617490832058472763195106264023717
```

#### 10.5.2.0.

Calcul de v i s:

In [24]:
```
v=0
s= e*d-1
while is_even(s):
    s=s//2
    v=v+1
```

Notem que coneixem els valors de v i de s.

In [25]:
```
print([v,s])
```

```
[2, 352398799967941472283643658817922640316301429732750444433070185
578655718788710042940820669959576820558981793221030280328970711128 0
406828557349084795054593206656569245921598667239628194014206374195 8
955462404025641435216148978729543498471141089660557572541673042329 5
638510382825829222978678709366686732440775800373491959083240607185 5
860188404878645952161207717342413060632149730288018084203280205830 6
362340811388783684697770337614190543066715066988981221834681832024 9
246587695845017269922751396947535189415243665930844247600475058963 0
314111048130525948056497273969352653544303475536671610800943516323 9
887589146130075]
```

#### 10.5.2.1.

Fem el càlcul per a un primer valor de m (obtindrem una factorització en el primer pas).

Prenem a l'atzar un valor de m en l'interval adequat.

In [26]:
```
m = ZZ.random_element(2,n-1)
```

In [27]:
```
1 print(m)
```

```
218512447333772688634267416481378212526045710135261658870858739428642985414955065861660172172722206123625133325384896076123802614663984750731022436772620350808105348726385350368324694850762554900214036873919906656342299659824690023725658093515146969483658640110796936256132692404744458256709662661145600114 21
```

**Observació**. A fi de veure el resultat predit al títol a cada execució de la llibreta, i que això no depengui de la tria a l'atzar dels valors de $m$, reprodueixo valors obtinguts a l'atzar que han funcionat per a cadascuna de les possibilitats. (O sigui, canvio el valor de $m$.)

In [28]:
```
1 m=201265870375843499998724125260565378897979855324954068935182221
```

In [29]:
```
1 print(m)
```

```
201265870375843499998724125260565378897979855324954068935182223463791365164740975165401071504064602898124983048867923167014398780005970303330669514788929304760117628169742725384300818597048722649397585462822954616954488365480471317481223765326615199805351368077783379884777851510454759502806800084814080756 10
```

Calculem $x_0 := m^s \pmod{n}$ i $mcd(x_0 - 1, n)$:

In [30]:
```
1 x0=mod(m,n)^s
2 p1=gcd(x0-1,n)
```

In [31]:
```
1 print(p1)
```

```
114525222780933212064850356634958489317012425035361692785736953014714210967159281059878318387427968967909926998541501267489521762880810794593407284853059 11
```

Mirem si hem factoritzat n; si obtenim "true" és que no hem factoritzat i cal canviar de valor de $m$; si surt "false", és que hem trobat un dels factors de $n$.

In [32]:
```
1 p1==n
```

Out[32]:
```
False
```

Efectivament, hem factoritzat $n$ (hem trobat el factor q0).

In [33]:
```
1 p1==p0
```

Out[33]:
```
False
```

In [34]:
```
1 p1==q0
```

Out[34]:
```
True
```

**10.5.2.2.**

Fem el càlcul per a un altre valor de m (de nou obtindrem una factorització en el primer pas,

però ara l'altre factor).

In [35]:
```
m = ZZ.random_element(2,n-1)
print(m)
```

```
834483475315851006653762193099898275070886544350299714190946011571807697899597775742342103284062466623275787891242198895834532325297333185670714931663462829485290928097817917882615719872343584253088641049151752921718905503644339084364477748468889667438601179913501620457288562596738961185683100461024798484496
```

In [36]:
```
m=704690153247548514094793934335849778076426989413598734324648049
```

In [37]:
```
print(m)
```

```
704690153247548514094793934335849778076426989413598734324648049921558630428630945052831235697882665709208224967811950728254697396828777941595821035717574568625184608304670959319421315680545315746086982201493739874169467600816112409325809896663343007950644835399297710867676756917312725684559599744885583932841
```

In [38]:
```
x0=mod(m,n)^s
p1=gcd(x0-1,n)
print(p1)
```

```
102233942506966577455470495413959951206844185848244668025849740114987649814016166810708151861513908023021781222316653441081802241131915250708084086910347 01
```

In [39]:
```
p1==n
```

Out[39]:
```
False
```

In [40]:
```
p1==p0
```

Out[40]:
```
True
```

Hem obtingut l'altre factor.

**10.5.2.3.**

Fem el càlcul per a un altre valor de m (ara, cal fer un segon pas).

In [41]:
```
m = ZZ.random_element(2,n-1)
print(m)
```

```
999569011234758128160392320961911098534020895501513933277885900869158135351004394098945128232356684050530381337369086279341360048700662175274340996099480500091949386700216495230987467539499248884332943315985968888264293148027288289004484214392302604626687277633381215834604081995408214282567352797792279619 64
```

In [42]:
```
m=111762776895405824585925620414799801960804692557814079906321044
```

In [43]:
```
print(m)
```

```
11176277689540582458592562041479980196080469255781407990632104447772868366949702533834527323970264526442733129980789357172482683405025319580840546194626998069515428810829960766554826660359425453361860721157974202395372109377828943503054758063242851708912518535154757556064078214049778831626190177301254357574 9
```

In [44]:
```
x0=mod(m,n)^s
p1=gcd(x0-1,n)
print(p1)
```

```
1
```

Ara no hem factoritzat, perquè hem trobat que el mcd és 1; cal elevar al quadrat.

In [45]:
```
x1=mod(x0,n)^2
p2=gcd(x1-1,n)
print(p2)
```

```
11452522278093321206485035663495848931701242503536169278573695301471421096715928105987831838742796896790992699854150126748952176288081079459340728485305911
```

Però...

In [46]:
```
p2==n
```

Out[46]:
```
False
```

In [47]:
```
p2==p0
```

Out[47]:
```
False
```

In [48]:
```
p2==q0
```

Out[48]:
```
True
```

Efectivament, també hem factoritzat n (hem obtingut l'altre factor).

**10.5.2.4.**

Fem el càlcul per a un altre valor de m (ara, no obtenim cap factorització).

In [49]:
```
m = ZZ.random_element(2,n-1)
print(m)
```

```
4955828482377268918444958176987227957036194445949271012109262693396311559111919603625661333395351338206917049783690752718696558376521415323417718141450748564092095379480971759960687433438214210859452715450678178119404315093226222012508221831945804850094956240473928859395399276163348546515908794838765628661 5
```

In [50]:
```
m=6632483662396665093823467674595146935966003371812584222822267344
```

In [51]:
```
print(m)
```

```
663248366239666509382346767459514693596600337181258422282226734405865395427411830684190443721536803170314750615269756301273813740440776226739157148571065815443422212622016835706247776953093951267464395122910770607012759244811371548953387285057878720652772410384182378644464079848308237861002728447985293974096
```

In [52]:
```
x0=mod(m,n)^s
p1=gcd(x0-1,n)
print(p1)
```

```
0
```

In [53]:
```
p1==n
```

Out[53]:
```
True
```

No hem factoritzat i cal canviar de valor de m.

**10.5.2.5.**

Fem el càlcul per a un altre valor de m (ara també cal elevar al quadrat, i tampoc no factoritzem).

In [54]:
```
m = ZZ.random_element(2,n-1)
print(m)
```

```
390909894936342576650399450133032118148724390326432812607290281205659497383505272655863660646484748581866151609716315182513372036884681618122900662982967092619877495498749603121427380761279068706677471392964982782411950894991915868685150717660484620817339332536732241894879327391376409053897644919966511229980
```

In [55]:
```
m=905819683539719474269139156734185660465014451524041339546008255
```

In [56]:
```
print(m)
```

```
905819683539719474269139156734185660465014451524041339546008255792358308648908948825840888572682890375451337834257831989745267073304096414107048660157832287073265259087181810432165169873429537942144979130153909175487584033385879310205715462226056437484476931896027789526626997402082406254052554936030906832120
```

In [57]:
```
x0=mod(m,n)^s
p1=gcd(x0-1,n)
print(p1)
```

```
1
```

Encara no hem factoritzat, perquè hem trobat que el mcd és 1; cal elevar al quadrat.

In [58]:
```
1 x1=mod(x0,n)^2
2 p2=gcd(x1-1,n)
3 print(p2)
```

```
0
```

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

Out[59]:
```
True
```

Ara, però, hem trobat que el mcd és n i hem de canviar de valor de m.

### 10.5.3. Observació final

Notem que un cop trobat un dels dos factors del mòdul, l'altre es pot obtenir trivialment per divisió. Això fa el càlcul de M també immediat, i el coneixement de la clau pública [N0,e0] i de la factorització permet calcular una clau privada [N0,dd0] tal que dd0·e0-1 sigui múltiple de M. Aquest valor potser no és el d0 original (que potser s'hagi obtingut a partir del valor de la phi d'Euler del mòdul), però és "la" clau privada bona associada a la clau pública. Per tant, podrem desxifrar qualsevol missatge xifrat amb la clau pública [N0,e0].

Deixem per a un capítol posterior aprofundir en aquestes qüestions de seguretat.

# Fi del capítol 10