Pràctiques Integrades
1er de Matemàtiques

Encriptació (RSA)

Curs 2003–04

Índex

VI Encriptació (RSA) 3

19 Codificació de missatges amb RSA 3

19.1 Codificació d'un missatge 4

20 Codificació de missatges amb RSA (II) 7

20.1 Cas pràctic 9

20.1.1 Missatges amb signatura 9

Part VI
Encriptació (RSA)

19 Codificació de missatges amb RSA

El primer que s'ha de fer per a poder treballar amb l'algoritme d'encriptació RSA és obtenir un parell de nombres primers prou grans i que es puguin mantenir en secret. Utilitzant les comandes `rand` i `nextprime` podem generar de forma aleatòria dos primers amb un nombre de xifres considerable. La comanda `randomize()` del principi fa que el generador de nombres aleatoris utilitzi com a llavor un nombre que depèn del rellotge del sistema i, d'aquesta forma, els nombres que surten canvien en cada sessió de Maple que arranquem (si no es fa així, la seqüència de nombres aleatoris sempre serà la mateixa).

```
> randomize();
> nombre1:= rand(10^80)();
> nombre2:= rand(10^80)();
> Primer1:= nextprime(nombre1);
> Primer2:= nextprime(nombre2);
```

(els nombres de 80 xifres caben prou bé en una sola línia i ja són prou grans com per fer força llargs els càlculs de descomposició del seu producte).

Seguidament s'han d'obtenir els dos nombres màgics que fan funcionar el sistema. El producte dels dos primers i el producte de cada un dels primers menys una unitat.

```
> n:= Primer1*Primer2;
> phin:= (Primer1-1)*(Primer2-1);
```

Podeu observar que, tot i que és fàcil determinar que `n` i `phin` no són primers, no és tan fàcil obtenir la seva descomposició.

```
> isprime(n);
> isprime(phin);
> ifactor(n);
> ifactor(phin);
```

Finalment s'ha de decidir quin serà el nombre `e` que s'ha de fer públic (juntament amb `n`) perquè ens puguin enviar missatges encriptats. Aquest nombre ha de ser coprimer amb `phin` i com que s'ha de fer públic es pot generar de qualsevol forma simple. Una opció consisteix en triar un nombre primer (així hi ha més opcions per a que no hi hagi factors comuns amb `phin`) d'algun tipus especial fàcil de generar. Per exemple podem fer

```
> e:= 2^16+1;
> isprime(e);
> igcd(e,phin);
```

Però també es pot deixar l'el·lecció a l'atzar encara que no sigui necessari en absolut.

Exercici 19.1

Trieu un nombre primer de fins a 6 xifres que sigui coprimer amb el nombre `phin` que teniu en aquests moments.

La clau privada per a desencriptar els missatges encriptats utilitzant la clau que acabeu de generar és el nombre `d` tal que `e*d` és 1 mòdul `phin`. Recordeu que la comanda `mod` pot fer la feina.

Observeu que si es coneix `phin` la clau `d` es pot obtenir fàcilment. Recordeu que, encara que conegueu `n`, el valor de `phin` no és immediat de calcular. En el paquet **numtheory** hi ha la comanda `phi` que pot determinar `phin` en funció de `n` i, per tant, la clau per a desencriptar es pot calcular a partir de les dades que són públiques (o no?).

Exemple 19.1

```
> with(numtheory);
> nombre:= 17*13;
> phi(nombre);
> phi(n);
```

Exercici 19.2

Determineu la clau `d` per a desencriptar que correspon a la clau pública `(e,n)` que acabeu de generar.

Exercici 19.3

Fabriqueu una funció `encript` que encripti un nombre i una `decript` que desencripti, a partir d'unes claus públiques i privades arbitràries (és a dir, entre els arguments hi ha d'haver els nombres `n`, `e` i `d`). Utilizeu `Power(a,e)` per a calcular potències amb combinació amb `mod`, si ho proveu de fer amb `a^e` ja veureu que els resultats no són gaire bons.

Podeu comprovar si les vostres funcions fan la feina calculant en alguns exemples

```
> encript(a,e,n);
> decript(%,d,n);
```

Nota: Recordeu que un nombre a s'encripta amb la clau pública (e, n) considerant a^e mòdul n i que si tenim un nombre b encriptat podem recuperar l'original fent b^d mòdul n.

19.1 Codificació d'un missatge

Si es vol utilitzar el sistema RSA per a enviar missatge autèntics, és a dir enviar una frase escrita com *"Hola, bon dia!"* a algú altre, cal codificar aquest missatge i convertir-lo en un nombre que pugui ser encriptat pel sistema. Aquesta part del procés és una altra de les parts que han de ser conegudes per tots els usuaris del mateix sistema. A continuació es proposa un sistema de codificació i descodificació de

missatges bastant simple basat en interpretar els codis ASCII dels caràcters que componen el missatge com els dígits d'un nombre escrit en base 256.

Primer de tot guardem en una variable la cadena de text que es vol codificar (el missatge)

```
> miss:= "Hola, bon dia!";
```

La comanda `convert` té una opció per a convertir un text en la llista dels codis ASCII de cada un dels caràcters que el composen. És suficient fer

```
> llascii:= convert(miss,bytes);
```

La mateixa comanda també fa el procés invers. És a dir, si tenim una llista de nombres entre 1 i 256 ens torna els caràcters que tenen aquests nombres com a codi ASCII

```
> convert(llascii,bytes);
```

Interpretar aquesta llista de codis ASCII com a dígits d'un nombre escrit en base 256 vol dir que el nombre que representa el missatge serà la suma

```
> misscod:=sum(llascii[i+1]*256^i,i=0..nops(llascii)-1);
```

I a aquest nombre és al que li podem aplicar els algoritmes d'encriptació.

El procés per a recuperar un missatge codificat també es pot fer sense gaire dificultat. Si tenim un nombre com `misscod`, el primer que podem fer és buscar la llista dels seus dígits si el volguéssim representar en base 256. La comanda `convert` també té les opcions necessàries per a obtenir aquest resultat amb

```
> llascii2:= convert(misscod, base,256);
```

Noteu ara que si feu escriure els caràcters que tenen els codis ASCII de la llista `llascii2` obteniu un altre cop el missatge original.

```
> printf(convert(llascii2,bytes));
```

Ara ja es pot fer un parell de procediments que realitzin tota la feina d'un sol cop.

Exercici 19.4

Feu un procediment `codif`, que tingui com argument una cadena de caràcters com `miss` i que doni com a resultat el nombre resultant de la forma de codificar que acabeu de veure, i un procediment `decod`, que realitzi la feina inversa, és a dir prengui un nombre i retorni la cadena de caràcters que representa d'acord amb la codificació anterior (utilitzeu `printf` per a representar el resultat, dóna els millors resultats de visualització).

Exercici 19.5

Els nombres que es poden encriptar bé amb el sistema RSA són tots els que són menors que `n`. Quants caràcters pot arribar a tenir un missatge per a poder codificar-lo amb els `n` que hem generat al principi?

20 Codificació de missatges amb RSA (II)

En la secció anterior heu pogut veure com funciona la base matemàtica per a poder encriptar missatges amb el sistema RSA. La finalitat d'aquesta pràctica és aplicar aquestes matemàtiques en un cas real (a més també veureu algunes funcions noves de Maple que poden resultar molt útils).

El primer que farem serà definir tots les mateixes funcions per a generar les claus públiques i privades i per a encriptar i desencriptar missatges. Per a generar un joc d'aquestes claus pública–privada es pot utilitzar la funció `generarcodi` següent:

```
> generarcodi:=proc(xifres)
> local p1,p2,m,n,phin,e,d;
> randomize();
> m:=floor(xifres/2+1);
> p1:=1;
> while p1<10^(m-1) do p1:=nextprime(rand(10^m)()); end do;
> p2:=p1;
> while (p2=p1) or (p1*p2<10^xifres) do p2:=nextprime(rand(10^m)()); end do;
> n:=p1*p2;
> phin:=(p1-1)*(p2-1);
> e:=phin;
> while igcd(e,phin)<>1 do e:=rand(10^6)() mod n; en do;
> d:=e^(-1) mod phin;
> [n,phin,e,d];
> end proc;
```

Aquesta funció té un únic argument `xifres` i produeix com a resultat una llista (delimitada amb `[ ]`) amb un nombre `n` que té, com a mínim, tantes xifres com el valor de l'argument `xifres` i és el producte de dos primers diferents `p1` i `p2` amb `xifres/2` xifres cada un; el producte `phin` de `(p1-1)` per `(p2-1)`; un exponent públic `e` que és un nombre arbitrari menor que 10^6 i coprimer amb el `phin`; i un exponent privat `d` que és l'invers de `e` mòdul `phin`. Observeu que s'utilitza la comanda `floor` (que calcula la part entera d'un nombre) per a poder ajustar la variable `m` i que la línia `while p1<10^(m-1) do p1:=...` faci que en `p1` s'hi trobi un nombre primer de més de `xifres/2` xifres; que la línia `while (p2=p1) or (p1*p2<10^xifres) do...` genera un segon primer `p2` que multiplicat per `p1` produeix un nombre de més de `xifres` xifres; i que `while igcd(e,phin)<>1 do...` genera un nombre de 6 xifres (o menys) que és coprimer amb el valor `phin`.

Les funcions `encriptarbloc` i `desencriptarbloc` següents són les que corresponen a la feina de la pràctica anterior. La primera té com arguments el missatge que es vol encriptar `missatge` i l'exponent `e` i el mòdul `n` públics per a donar com a resultat un nombre que representa el missatge encriptat. La funció de desencriptar té com arguments el nombre `numencript` que representa un missatge encriptat, l'exponent privat `d` i el mòdul públic `n`, i el resultat és el missatge que volem veure.

```
> encriptarbloc:=proc(missatge,e,n)
> local i,llistanum,num,numencript;
> llistanum:=convert(missatge,bytes);
> num:=sum(llistanum[i+1]*256^i,i=0..nops(llistanum)-1);
> numencript:=Power(num,e) mod n;
> numencript;
> end proc;
```

```
> desencriptarbloc:=proc(numencript,d,n)
> local i,num,missatge,llistanum;
> num:=Power(numencript,d) mod n;
> llistanum:=convert(num,base,256);
> missatge:=convert(llistanum,bytes);
> missatge;
> end proc;
```

Recordeu que aquestes funcions no permeten encriptar i desencriptar missatges de qualsevol longitud. Si el missatge que es vol enviar és massa llarg (té més de $\log_{256}(n) = \dfrac{\ln(n)}{\ln(256)}$ caràcters), el nombre que el representa serà més gran que el mòdul `n` i en el moment de desencriptar no el podrem recuperar. Si es volen enviar missatges de qualsevol longitud s'ha de dissenyar un mecanisme que parteixi el missatge en parts prou petites per a poder encriptar-les i desencriptar-les de manera fiable.

Exercici 20.1

Definiu un procediment `particionar` que tingui com arguments una llista qualsevol `llista` i un nombre `m`, i doni com a resultat una nova llista formada per les llistes obtingudes considerant el diferents blocs consecutius de `m` elements de `llista` i per una última llista amb els elements que puguin quedar si el nombre d'elements de `llista` no és múltiple de `m`.

Per exemple, si `llista:=[1,2,3,4,5]` i `m:= 2`, el resultat de `particionar(llista,m)` hauria de ser la llista `[[1,2],[3,4],[5]]`, i si `m:= 3` el resultat seria `[[1,2,3],[4,5]]`.

Un cop obtinguda aquesta funció que parteix un missatge en parts d'una longitud correcta, podeu veure com es pot utilitzar `particionar` i la comanda de Maple `map` per a modificar les funcions `encriptarbloc` i `desencriptarbloc` per a obtenir les noves comandes `encriptar` i `desencriptar` que poden encriptar i desencriptar missatges de qualsevol longitud seguint el mateix mecanisme que estem utilitzant fins ara.

```
> encriptar:=proc(missatge,e,n)
> local long,llistach,nums,llistablocs;
> global particionar;
> llistach:=convert(missatge,bytes);
> long:=floor(log(n)/log(256))-1;
> llistablocs:=particionar(llistach,long);
> nums:=map(x->sum(x[i+1]*256^i,i=0..nops(x)-1),llistablocs);
> map(x->Power(x,e) mod n,nums);
> end proc;
> desencriptar:=proc(llistanumsencript,d,n)
> local llistanums,llistablocs,llistach;
> llistanums:=map(x->Power(x,d) mod n,llistanumsencript);
> llistablocs:=map(x->convert(x,base,256),llistanums);
> llistach:=map(op,llistablocs);
> convert(llistach,bytes);
> end proc;
```

Si mireu el que diu l'ajuda de Maple sobre la comanda `map` veureu que `map(funcio,llista)` aplica la funció `funcio` a cada un dels elements de la llista `llista`. Per exemple, si considereu `map(sin,[a,b,c]);` obtindreu com a resultat `[sin(a), sin(b), sin(c)]`. Per un altre costat, fixeu-vos també en que dins

d'una comanda `map` es pot utilitzar una funció definida amb la notació `x -> expressio` (que és una funció sense cap nom assignat).

20.1 Cas pràctic

Amb les funcions de codificació i encriptació que teniu definides, ja es poden enviar i rebre missatges amb la seguretat de que, a part del destinatari i el remitent, ningú els pot llegir. Per a poder aplicar el mètode en un cas real disposeu d'una carpeta de la xarxa on hi podeu llegir i escriure tots vosaltres, dins aquesta carpeta podeu crear una subcarpeta amb el vostre nom (o qualsevol codi que us identifiqui) i posar dins aquesta subcarpeta un fitxer (creat amb qualsevol editor, per exemple NOTEPAD, i al que li podeu posar per nom `clau.txt`) que contingui la vostra *clau pública* `(e,n)` (generada amb la comanda de Maple corresponent). Quan algú us vulgui enviar un missatge només ha de recollir la vostra clau pública i utilitzar la dada per a encriptar el missatge, deixant el resultat (una llista de nombres més o menys llarga que per comoditat hauria de ser de la forma `[n1, n2, n3, ...]`) en la vostra carpeta en un altre fitxer de text que tingui un nom que identifiqui al remitent (o no!).

Recordeu que la seguretat del sistema està garantida pel temps que es necessita per a descompondre `n` en factors primers i que si s'utilitza una clau `n` massa petita qualsevol que tingui accés al contingut dels missatges encriptats i a les claus públiques podrà calcular les claus privades i desencriptar els missatges.

Exercici 20.2

Definiu un procediment `trencarcodi` que tingui com argument una clau pública `(e,n)` i obtingui com a resultat la clau privada `d` corresponent. Un cop fet això haurieu de poder desencriptar un missatge encriptat amb una certa clau pública `(e,n)` utilitzant `desencriptar(missatge,trencarcodi(e,n),n)`. **Nota:** Si intenteu utilitzar `trencarcodi` amb un `n` de més de 30 xifres és molt probable que deixeu l'ordinador un xic penjadet.

20.1.1 Missatges amb signatura

Segurament ja deveu haver notat que amb aquest sistema d'encriptació hi ha la seguretat de que els missatges que s'envien només els pot llegir el destinatari autèntic però que no hi ha cap garantia sobre el remitent. És a dir, és molt fàcil enviar un missatge utilitzant una identitat falsa.

Una forma d'afegir una *signatura digital a un missatge* consisteix en afegir al text que es vol enviar el resultat d'encriptar amb la clau privada del remitent un text que identifiqui tant al missatge com a aquest remitent (sovint es repeteix el missatge que es vol enviar, o una part del missatge). D'aquesta manera, quan el destinatari desencripta el missatge troba tot el text del missatge pròpiament dit i, a més, una llista de nombres que correspon a la signatura. Observeu que aquesta signatura es desencripta amb la clau pública del remitent i, per tant, el destinatari pot comprovar si el remitent és l'autèntic.

Exemple 20.1

L'Anna, que té com a clau pública `e:= 234281, n:= 2013113304270640992168460 1` i clau privada `d:= 859255783441064766364492 1`, vol quedar amb la Jessica, que té com a clau pública `e1:= 183643, n1:=`

107015351631161709967664 71, per anar a dinar a les 14 hores. Encripta amb la seva clau privada la signatura `"Quedem per dinar a les 14. De l'Anna per la Jessica."` i obté

```
[8568033446836833340239, 96172083442151221291194 82, 3569295709510289113864968,
10191676509591032189424162, 7327887385199906740741770, 5144324498380469293535799]
```

Seguidament redacta el missatge

```
"Quedem per dinar a les 14. De l'Anna per la Jessica. [8568033446836833340239,
96172083442151221291194 82, 3569295709510289113864968, 10191676509591032189424162,
7327887385199906740741770, 5144324498380469293535799]"
```

l'encripta amb la clau pública de la Jessica i envia

```
[3563711015752004319402424, 10302110634942014127686446, 5179879773797379577822967,
20199947758034170341824 2, 9999366564979028193721266, 8764097362618589741135968,
7683139254169211311340606, 9524083387533957137565194, 9347924217402170216685381,
6458480987451128724947 09, 7097746577892742359072760, 4709368691474043888219700,
3027831765049390860565394, 2019554872802334481328513, 3735030092110383645909035,
4570576624440291186973375, 1065866632153528263254 3875, 7180206697277055734198727,
5960888968783855178496638, 8305018049790205930420916, 1560125811123715010751873,
9011004426070128228498872, 2183985496715709988142589, 1306070483254503631042304]
```

Quan la Jessica desencripti amb la seva clau privada el missatge, tornarà a recuperar

```
"Quedem per dinar a les 14. De l'Anna per la Jessica. [8568033446836833340239,
96172083442151221291194 82, 3569295709510289113864968, 10191676509591032189424162,
7327887385199906740741770, 5144324498380469293535799]"
```

i per a comprovar si el missatge és realment de l'Anna desencriptarà amb

```
desencriptar( [8568033446836833340239, 96172083442151221291194 82,
3569295709510289113864968, 10191676509591032189424162, 7327887385199906740741770,
5144324498380469293535799],234281,20131133042706409921684601 );
```

la part de la signatura utilitzant la clau pública de l'Anna i obtindrà un altre cop el mateix missatge de forma que pot estar segura de que el missatge prové de l'Anna.