Arithmétique TI-Nspire	Cryptographie, RSA

Vocabulaire

Cryptographie (du grec *kruptos*, caché et *graphein*, écrire) : technique consistant à transformer un message (par un procédé mathématique, par exemple) en un message codé, de telle sorte qu'un lecteur indésirable n'en comprenne pas le sens, et que le destinataire soit capable de le traduire.

Clés : procédés (mathématiques, par exemple) permettant de crypter un message (*clé de chiffrement*), ou de le retranscrire en clair (*clé de déchiffrement*).

Cryptanalyse : activité ayant pour but de décoder un message codé, sans en connaître la clé.

L'usage fait que l'on emploie indifféremment les mots de codage ou de chiffrement – mais contrairement au premier, le second manifeste la volonté explicite de cacher le message aux yeux indiscrets par un procédé cryptographique.

Objectifs

Nous retrouvons la sémillante Alice et ses aventures épistolaires (voir l'épisode précédent sur le chiffrement affine).
Alice, qui ne se lasse pas d'envoyer des messages à Bob, veut maintenant les crypter par la méthode **RSA**, du nom de ses inventeurs, **R**ivest, **S**hamir et **A**dleman (1977).
Le principal avantage de cette méthode réside en ce que la clé de chiffrement est publique, connue de tous, disponible dans un annuaire.
Avec le chiffrement affine, on se rappelle que Bob disposait d'une clé privée par correspondant, servant à la fois au chiffrement et au déchiffrement – donc strictement confidentielle. Cette clé devait au préalable avoir été partagée avec son interlocuteur. Les problèmes de gestion et d'échange de ces clés deviennent assez vite insupportables quand le nombre de correspondants augmente. D'où l'intérêt des clés publiques, qu'il suffit de lire dans un annuaire et qui sont utilisées par n'importe quel correspondant qui veut écrire à Bob. Il va de soi que la clé privée, qui sert au déchiffrement de ses messages, est gardée bien précieusement par Bob.

Principe de la cryptographie RSA

Bob doit d'abord fabriquer sa clé publique pour qu'Alice puisse lui écrire.

Il choisit deux grands nombres premiers **distincts** p et q et calcule leur produit $n = pq$. Si le nombre obtenu n est très grand (de l'ordre de quelques centaines de chiffres), sa décomposition en facteurs premiers résistera aux calculateurs les plus puissants... Le très grand entier n peut donc être divulgué sur la place publique par Bob : en l'état actuel des algorithmes et des ressources informatiques, personne ne peut retrouver ni p, ni q[1].

Bob calcule ensuite l'entier $m = (p-1)(q-1)$.[2]

Bob choisit ensuite un nombre c, premier avec m.

1) D'après le théorème de Bézout, comme c est premier avec m, il existe des entiers relatifs d et l tels que $cd + ml = 1$.
On peut donc écrire :

$cd = -ml + 1 = mk + 1$ en posant $k = -l$.

[1] Des précautions dans le choix de p et q doivent être prises. Dans des configurations particulières, par exemple si p et q sont très proches l'un de l'autre, la factorisation peut être possible malgré la taille de n.

[2] On sait que m n'est rien d'autre que le nombre d'entiers compris entre 1 et n et premiers n. L'entier m est donc l'indicateur d'Euler de n noté souvent $\varphi(n)$.

Remarquons que les entiers *d* et *k* peuvent être déterminés avec l'algorithme d'Euclide étendu, dont on sait qu'il est très performant, même si les entiers qui interviennent sont grands.

On peut, en quelque sorte, affirmer que *d* n'est rien d'autre que l'inverse de *c* modulo *m* puisqu'il vérifie :

$$cd \equiv 1 \pmod{m}.$$

L'entier *c*, diffusé publiquement, servira à chiffrer les messages, tandis que *d*, gardé précieusement secret par Bob, lui permettra de les déchiffrer.

Bilan

Les nombres *n* et *c* sont *publics* et serviront à Alice, ou à d'autres, à coder un message à destination de Bob selon la méthode RSA. Ils pourront figurer dans un annuaire.

Mais le nombre *d* est gardé *secret* par Bob : c'est ce nombre qui lui permettra, et à lui seulement, de décoder les messages qu'il recevra.

Partie publique annuaire par exemple	**Partie privée**
…	…
Bob…… *n*, *c*	*d*
…	…

Remarquons que, même lorsque *c* et *n* sont connus, rien ne permet alors de retrouver *d* (ou du moins en un temps humainement raisonnable…), puisque $m=(p-1)(q-1)$ est inconnu.

La seule façon de calculer *d* serait de connaître *p* et *q* mais la décomposition en facteurs premiers d'entiers suffisamment grands[3] est une tâche, qui, si elle est *théoriquement* possible, prend *pratiquement* un temps rédhibitoire (plusieurs milliers, voire millions, d'années).

Cryptographie RSA : la théorie

2) a) Dans le cas où *x* n'est pas divisible par *p*, *p* étant premier, le petit théorème de Fermat permet d'affirmer que :

$$x^{p-1} \equiv 1 \pmod{p}$$

Tenant compte du fait que $m=(p-1)(q-1)$, on peut alors affirmer que $x^{km}=\left(x^{p-1}\right)^{k(q-1)} \equiv 1 \pmod{p}$.

b) Si maintenant *x* est divisible par *p*, alors $x \equiv 0 \pmod{p}$. On en déduit que $x^{cd} \equiv 0 \pmod{p}$, ce qui prouve bien que $x^{cd} \equiv x \pmod{p}$.

3) On a donc à la fois : $x^{cd} \equiv x \pmod{p}$ et $x^{cd} \equiv x \pmod{q}$ donc $x^{cd}-x$ est à la fois multiple des nombres premiers *distincts* *p* et *q*, donc de *pq* car *p* et *q* sont premiers entre eux. En conséquence, on a bien $x^{cd} \equiv x \pmod{n}$.

Ces résultats permettent de comprendre le principe de la cryptographie RSA :

> un message *M*, qu'Alice veut envoyer à Bob, est converti en un entier naturel *x*, de telle sorte que $0 \leq x < n$;[4]
>
> pour chiffrer son message, Alice utilise les clés publiques de Bob, *n* et *c*, et calcule $y=x^c \pmod{n}$;
>
> Bob reçoit $y=x^c \pmod{n}$ et calcule *d* au moyen de sa clé secrète : $y^d=x^{cd} \equiv x \pmod{n}$;
>
> il peut alors récupérer le message initial.

[3] La méthode RSA utilise couramment des entiers *n* possédant entre 300 et 600 chiffres.

[4] Au besoin, on découpe le message en tranches…

Un exemple de détermination de *n*, *c* et *d* à l'aide de TI-Nspire

3) a) On n'a que l'embarras du choix. On peut, par exemple, utiliser la fonction **nextprime(n)** de la bibliothèque **Numtheory** : cette fonction renvoie le plus petit nombre premier supérieur ou égal à *n*. On peut, par exemple, prendre p = 1 237 et q = 2 531, de telle sorte que $n = p \times q = 3130\,847$, qui possède bien 7 chiffres.

numtheory\nextprime(1234)	1237
numtheory\nextprime(2523)	2531
1237·2531	3130847

On en déduit la valeur de *m* :

$$m = (p-1)(q-1) = 1\,236 \times 2\,530 = 3\,127\,080\,.$$

1236·2530	3127080

Pour choisir simplement un entier premier avec *m* [5], il suffit de choisir un entier premier n'apparaissant pas dans la décomposition en facteurs premiers de *m*, par exemple 107 : c'est cet entier qui nous servira de clé de chiffrement *c*.

factor(3127080)	$2^3 \cdot 3 \cdot 5 \cdot 11 \cdot 23 \cdot 103$
numtheory\nextprime(104)	107

b) La détermination des entiers *u* et *v* se fait à l'aide de la fonction **bezout** de la bibliothèque **Numtheory**, comme le montre l'écran suivant :

numtheory\bezout(107,3127080)

au+bv=d

u= -1256677 , v= 43 , d= 1

{-1256677,43,1}

Il est clair que $107 \times (-1\,256\,677) + 43 \times 3\,127\,080 = 1$. En conséquence, on peut prendre :

$$u = -1\,256\,677 \text{ et } v = 43\,.$$

L'égalité précédente modulo *m* conduit à :

$$107 \times (-1\,256\,677) \equiv 1 \pmod{m}\,.$$

On a donc aussi :

$$107 \times (m - 1\,256\,677) \equiv 1 \pmod{m} \text{ soit } 107 \times 1\,870\,403 \equiv 1 \pmod{m}\,.$$

3127080−1256677	1870403
mod(107·1870403,3127080)	1

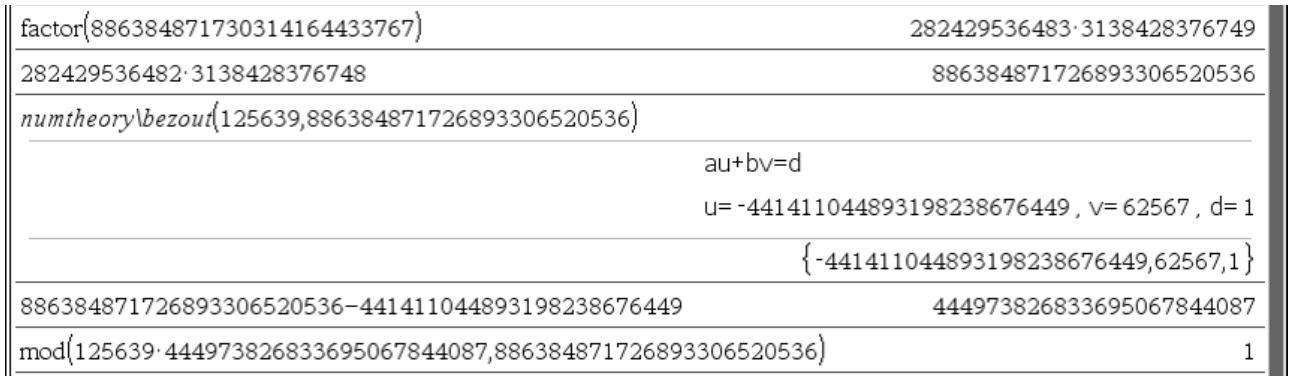
On peut donc choisir comme clé de déchiffrement *d* = 1 870 403.

c) La partie publique figurant dans l'annuaire est donc constituée de *n* = 3 130 847 et *c* = 107, tandis que la partie privée est *d* = 1 870 403.

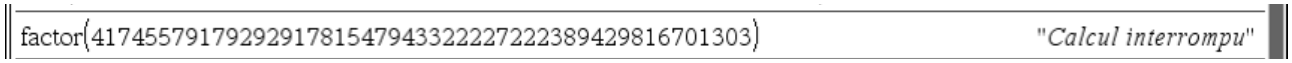
4) On tente donc de décomposer en facteurs premiers chacun de ces entiers.

[5] Sinon, notamment dans le cas où la décomposition en facteurs premiers de *m* ne peut pas être obtenue simplement, on prend un entier au hasard et on teste s'il est premier avec *m*, en répétant l'expérience jusqu'à ce qu'il le soit. L'algorithme d'Euclide étant performant, le procédé conduit à un résultat rapidement.

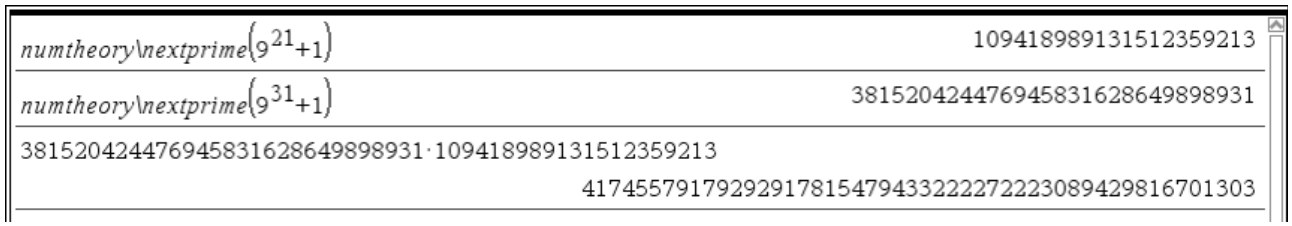
Pour le premier, TI-Nspire renvoie rapidement une réponse. À partir de là, on récupère facilement la valeur de *d* (= 444 973 826 833 695 067 844 087), comme le montre l'écran suivant :



Le deuxième résiste à la fonction **factor**, qu'on est obligé d'interrompre :

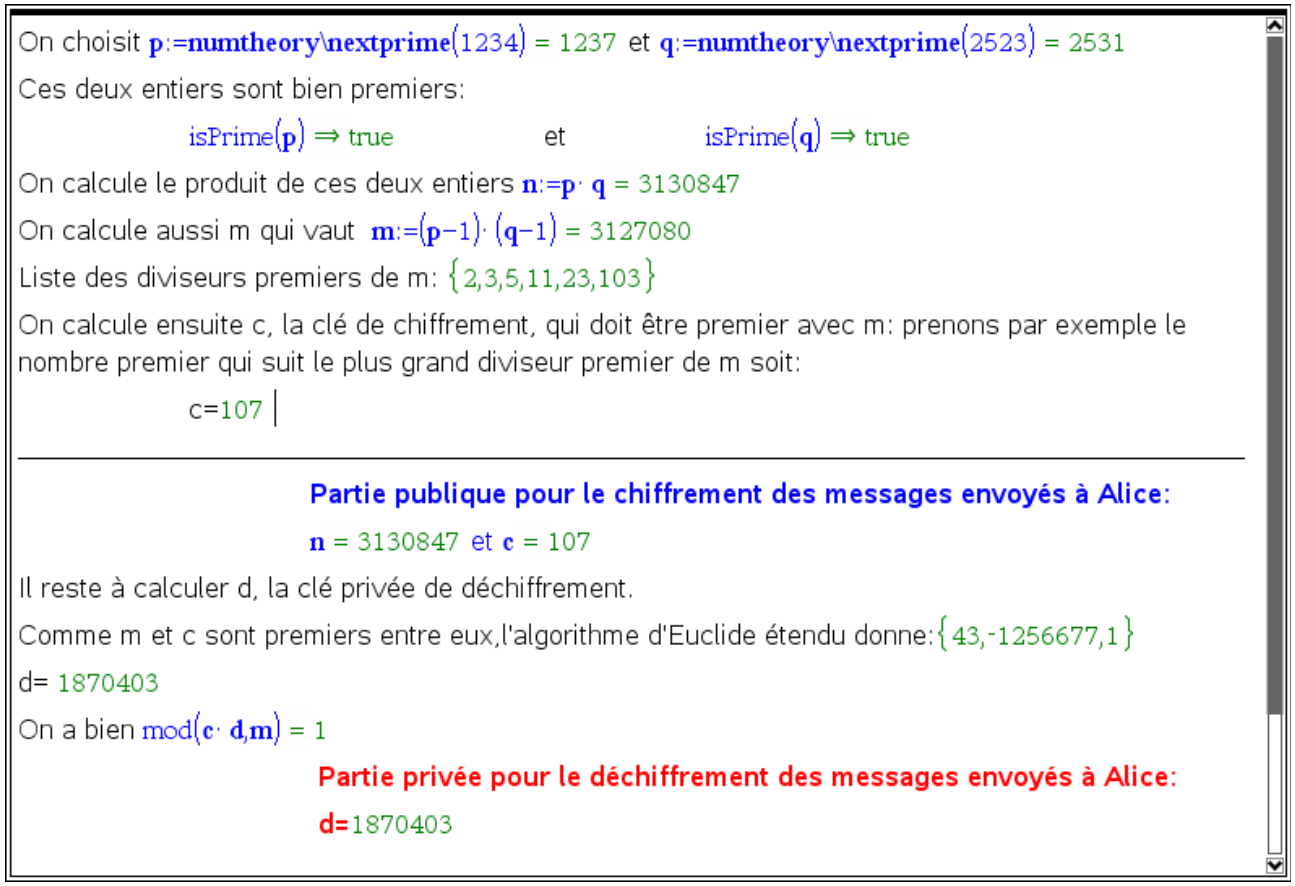


Pour information, ce nombre a été fabriqué à l'envers à partir de deux grands nombres premiers :



Tant que l'on n'utilise pas de technologie plus performante que TI-Nspire, on peut être sûr que les messages envoyés par Maud ne seront jamais déchiffrés…

Signalons que l'on peut utiliser l'**Éditeur Mathématique** pour automatiser, dans un environnement convivial, le choix de *n*, *c* et *d* comme le montre l'écran suivant :



Seules sont à choisir les valeurs du début, complètement au hasard, 1 234 et 2 523, le reste est obtenu automatiquement.

Un chiffrement à l'aide du tableur

Alice souhaite envoyer à Bob le message[6] :

« Il pleut sur la mer… ».

Dans l'annuaire RSA, elle a trouvé les informations suivantes[7] :

Bob : n = 1 125 337 et c = 1 009.

Alice souhaite, pour commencer, coder son message dans le tableur.

Nous coderons d'abord les lettres du message en nombres entiers[8] de 11 à 36, de façon analogue à ce qu'on a vu pour le chiffrement affine.

5) La feuille de calcul complètement terminée se présente comme indiqué sur la page suivante.

a) Une fois saisies les valeurs, la touche [var] permet de mémoriser les valeurs dans les variables n et c.

b) On saisit 1 dans la cellule C1, puis **=C1+1** dans la cellule C2 et on recopie vers le bas jusqu'à la cellule C100.

c) Ne pas oublier de saisir les guillemets pour que les lettres soient considérées non pas comme des variables mais comme des caractères.

d) La deuxième forme proposée est la plus efficace mais elle est inopérante sur un tableur traditionnel.

6) a) et **b)** Cette instruction permet le regroupement des nombres de la colonne E par paquets de 3.

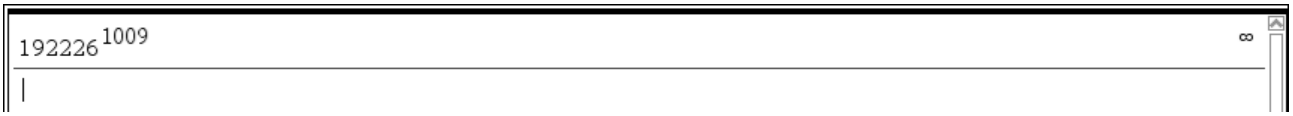
Dans la cellule F1, on retrouvera 192226, c'est-à-dire la concaténation des nombres des cellules E1, E2 et E3. Dans la cellule F2, pour laquelle C2 = 2 n'est pas congru à 1 modulo 3, apparaît un trait, traduisant le fait que cette cellule est vide (**void**). Il en est de même pour la cellule F3.

On peut alors commencer à appliquer dans la cellule H1 le procédé cryptographique RSA.

On sait que n =1 125 337 et c = 1 009.

Le procédé est particulièrement simple : il suffit de calculer le contenu de F1, soit 192 226, à la puissance c modulo n.

7) a) Le problème qui se pose, c'est que TI-Nspire évalue d'abord 192 226 à la puissance 1 009, ce qui dépasse ses capacités et interdit la poursuite du calcul.



b) Il n'en est pas de même avec la fonction **pwrmod** de la bibliothèque **Numtheory**, où le calcul est effectué de proche en proche.

On peut saisir dans la cellule H1

=when(not isvoid(F1),numtheory\pwrmod(f1,c,n),void).

Après la recopie vers le bas, le calcul n'est effectué que lorsque le contenu de la cellule correspondante de la colonne F n'est pas vide.

c) Dans la zone grisée de la colonne I, on saisit **=delvoid(h[])** pour ne récupérer que la liste des nombres correspondant au message codé par le procédé RSA. On mémorise le résultat dans une variable **mescod**.

[6] C'est le premier vers d'une belle chanson d'Allain Leprest, une belle plume normande de la chanson française.

[7] Nous revenons volontairement à des entiers de taille raisonnable pour des raisons pratiques.

[8] Ainsi les lettres sont toujours représentées par un nombre possédant *exactement* deux chiffres. Cela simplifiera notre travail.

	A	B	C	D	E	F	G	H	I mescod	J
◆					=ord(d[])−86				=delvoid(h[])	
1	n=	1125337	1	i	19	192226		820961	820961	
2	c=	1009	2	l	22	_		_	627270	
3			3	p	26	_		_	534096	
4			4	l	22	221531		627270	683715	
5			5	e	15	_		_	847441	
6			6	u	31	_		_		
7			7	t	30	302931		534096		
8			8	s	29	_		_		
9			9	u	31	_		_		
10			10	r	28	282211		683715		
11			11	l	22	_		_		
12			12	a	11	_		_		
13			13	m	23	231528		847441		
14			14	e	15	_		_		
15			15	r	28	_		_		
16			16			_		_		
17			17			_		_		
18			18			_		_		

H1 =when(not isvoid(f1),numtheory\pwrmod(f1,c,n),_,_)

Où Bob décode le message d'Alice

Bob reçoit le message *y* codé par Alice, qui est contenu dans la liste **mescod**. Il se propose de le déchiffrer dans une nouvelle feuille de calcul.

On sait que Bob, qui est le seul à posséder la clé de déchiffrement *d*, doit juste calculer $y^d = \left(x^c\right)^d = x^{cd} \equiv x \pmod{n}$ pour récupérer le message initial *x* ; plus exactement, il doit calculer le reste dans la division de y^d par *n*.

Sachant qu'au départ, on a choisi *x* de telle sorte que $0 \leq x < n$, on retrouve bien le message initial, qu'il ne reste plus qu'à convertir en lettres.

8) a) On copie dans la cellule B1 la valeur 227 089 que l'on mémorise dans une variable *d*.
On copie ensuite la variable **mescod** dans la colonne C (**=mescod** dans la zone grisée de la colonne).

b) Il suffit de saisir en D1 l'instruction **=when(not isvoid(c1),numtheory\pwrmod(c1,d,n),void)** et de recopier vers le bas, sur 100 lignes par exemple. Le calcul ne sera effectué que quand un nombre est donné dans la cellule correspondante de la colonne C ; sinon un tiret (**void**) sera placé dans la cellule.

c) On peut, par exemple, récupérer les deux premiers chiffres dans E1 avec **=int(D1/10000)** ; mieux, on peut ne faire ce calcul qu'à la condition qu'il y ait un nombre dans la cellule D1, avec **=when(not isvoid(D1),int(D1/10000), void).**

De même, on recommence avec le reste de cette division **mod(D1,10000)** en prenant dans F1 la partie entière du quotient de ce reste par 100 soit **=int(((mod(d1,10000))/(100)))** ou mieux encore **=when(not isvoid(D1),int(((mod(d1,10000))/(100))),void)**.

Enfin les deux derniers chiffres dans G1 sont, par exemple, le reste dans la division de D1 par 100 soit **=mod(d1,100)** ou mieux **when(not isvoid(D1),mod(D1,100),void)**.

On recopie vers le bas ces instructions sur 100 lignes.

On retrouve bien le message envoyé par Alice à Bob (lecture de gauche à droite et de haut en bas).

	A	B	C	D	E	F	G	H	I	J	K
◆			=mescod								
1	d=	**227089**	820961	192226	19	22	26		i	l	p
2			627270	221531	22	15	31		l	e	u
3			534096	302931	30	29	31		t	s	u
4			683715	282211	28	22	11		r	l	a
5			847441	231528	23	15	28		m	e	r
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											

E1 $=\text{int}\left(\frac{d1}{10000}\right)$

9) Les chiffrements demandés sont réalisés sur les écrans suivants (comme convenu, on a ajouté la première lettre du prénom de l'expéditeur pour obtenir en tout un nombre de lettres égal à un multiple de 3).

	A	B	C	D	E	F	G	H	I	J
◆					=ord(d[])−86				=delvoid(h[])	mescod
1	n=	**1125337**	1	p	26	261524		348439	348439	
2	c=	**1009**	2	e	15	_		_	110702	
3			3	n	24	_		_	905800	
4			4	s	29	291528		110702	813320	
5			5	e	15	_		_	137981	
6			6	r	28	_		_	216709	
7			7	c	13	131529		905800		
8			8	e	15	_		_		
9			9	s	29	_		_		
10			10	t	30	301419		813320		
11			11	d	14	_		_		
12			12	i	19	_		_		
13			13	r	28	281524		137981		
14			14	e	15	_		_		
15			15	n	24	_		_		
16			16	o	25	252411		216709		
17			17	n	24	_		_		
18			18	a	11	_		_		

F1 $=\text{when}(\text{mod}(c1,3)=1, e1\cdot 10000+e2\cdot 100+e3, \_)$

	A	B	C	D	E	F	G	H	I	J	K
◆			=mescod								
1	d=	**227089**	348439	261524	26	15	24		p	e	n
2			110702	291528	29	15	28		s	e	r
3			905800	131529	13	15	29		c	e	s
4			813320	301419	30	14	19		t	d	i
5			137981	281524	28	15	24		r	e	n
6			216709	252411	25	24	11		o	n	a

Avec les valeurs calculées au départ, chacun obtiendra son chiffrement RSA particulier.

Écriture d'une fonction

Elle est indispensable pour une automatisation complète de la méthode RSA mais elle demeure délicate à écrire pour des élèves de lycée. Il reste qu'il n'est pas inintéressant, pour les passionnés d'algorithmique, de tenter d'en comprendre le principe.

Pour le codage RSA, la fonction s'appelle **crsa** et prend comme paramètres d'entrée le message à coder *mes* sous forme de chaîne de caractères, ainsi que les entiers *n* et *c* ; elle renvoie un nombre entier qui correspond au message codé.

Autant que c'est possible, les instructions importantes de cette fonction sont commentées dans le code qui suit :

```
crsa                                                    15/19
Define crsa(mes,n,c)=
Func
Local l,i,cod,chif,k,nn,p,nb,rsa,d,p
{}→cod:{}→chif
© codage du message lettre par lettre, 11 pour a, 12 pour b, etc.
For i,1,dim(mes)
  mid(mes,i,1)→l:  augment(cod,{ord(l)-86})→cod
EndFor
© taille des paquets quand on regroupe les lettres… p paquets de 2 chiffres
dim(string(n))→nn © taille du nombre n choisi:int((nn-1)/2)→p © nombre de paquets
© on complète la liste cod avec des 0 pour qu'elle ait un nombre d'éléments multiples de p
While mod(dim(cod),p)≠0
  augment(cod,{0})→cod
EndWhile
© on extrait les paquets de p chiffres et on les code par rsa
For i,1,dim(cod),p
  sum(mid(cod,i,p)·seq(10^(2·k),k,p-1,0,-1))→nb
  numtheory\pwrmod(nb,c,n)→rsa
  augment(chif,{rsa})→chif
EndFor
Return sum(chif·seq(10^(k·nn),k,dim(chif)-1,0,-1))
EndFunc
```

Un exemple de codage, toujours avec les paramètres publics de Bob, mémorisés dans les variables *n* et *c* :

```
crsa("ilpleutsurlameretcasertarienquanoyerdeboutlegardiendephare",n,c)
 8209616272705340966837158474419272688929221610120378413601748733837437480431366746220739790
```

La fonction pour le décodage s'appelle **drsa**. Elle prend comme paramètres d'entrée le message codé sous forme d'un nombre entier *mescod*, la valeur de *n* et de la clé de décodage *d*. Elle renvoie le message décodé sous la forme d'une chaîne de caractères.

```
"drsa" enregistrement effectué
Define drsa(mescod,n,d)=
Func
Local p,nb,liste,mesclair,nn,i,j,rep,l1,l2,l3,letcod,n1
{}→liste:""→rep
© taille des paquets quand on regroupe les lettres... p paquets de 2 chiffres
dim(string(n))→nn © taille du nombre n choisi
2·int((nn−1)/2)→p © nombre de paquets
© extraction des nombres à partir de la droite
While mescod>0
  mod(mescod,10^nn)→nb: augment({nb},liste)→liste:(mescod−nb)/10^nn→mescod
EndWhile
For i,1,dim(liste)
  string(numtheory\pwrmod(liste[i],d,n))→mesclair
  For j,1,dim(mesclair),2
    expr(mid(mesclair,j,2))→n1
    If n1=0:Exit
    rep&char(n1+86)→rep
  EndFor
EndFor
Return rep
EndFunc
```

Le décodage du message précédent donne alors :

```
drsa(8209610627270053409606837150847441092726808929220161012003784103601740873383074374800 ▸
                                                    "ilpleutsurlameretcasertarienquanoyerdeboutlegardienduphare"
```

On peut automatiser le chiffrage et le déchiffrage des messages à l'aide de l'**Éditeur Mathématique** :

```
Entrer la valeur de n: nn:=1125337

Entrer la valeur de c: cc:=1009

Entrer ici le message à coder (pas d'accents, pas de majuscules, pas d'espaces et pas de ponctuation)
mm:="cestuntroudeverdureouchanteunerivieraccrochantfollementauxherbesdeshaillonsdargent"

Le message codé est alors:
90580003825060442906098637702047250693147065815704653000033609053527309150390183246053081101 ▸
```

Entrer la valeur de n: **nn**:=1125337

Entrer la valeur de d: **dd**:=227089

Le message à décoder est:

mc

= 9058000382506044290609863770204725069314706581570465300003360905352730915039018324605308 1

Déchiffrement :

drsa(**mc**,**nn**,**dd**) ▸ cestuntroudeverdureouchanteunerivieréaccrochantfollementauxherbesdeshaillonsdargent