Il est facile d'emmêler un fil de pêche, alors qu'il est très difficile de le démêler. De même, certaines opérations sur les nombres sont faciles dans un sens et difficiles dans l'autre. On les appelle opérations à sens unique. Elles sont utilisées pour la cryptographie.

Christiane Rousseau
Université de Montréal

Lorsque vous envoyez votre numéro de carte de crédit par Internet à un site protégé, le logiciel crypte celui-ci par le *code RSA*, du nom de ses inventeurs : Ronald Rivest, Adi Shamir et Leonard Adleman.

Historiquement, les codes secrets ont tous été percés. Mais le code RSA, publié en 1978, résiste encore malgré tous les efforts de la communauté mathématique pour le briser. Il repose sur le fait que, si on prend deux très grands nombres premiers distincts p et q et qu'on les multiplie pour obtenir $n = pq$, alors il est hors de portée des ordinateurs actuels[2] de récupérer p et q. Ainsi, on peut se permettre de publier n, la *clé*, qui sert au cryptage, sans crainte que quiconque puisse récupérer p et q qui servent au décryptage. Ce procédé s'appelle la *cryptographie à clé publique* parce que la recette de cryptage est publique.

Aussi surprenant que cela puisse paraître, il est facile pour un ordinateur de fabriquer de grands nombres premiers. Plusieurs algorithmes générant de grands nombres premiers sont *probabilistes*. Nous allons en présenter un ci-dessous.

2. *Voir l'article « L'héritage de Pierre de Fermat pour la factorisation des grands nombres » de Jean-Marie De Koninck, p. 8 de ce numéro.*

Dans la cryptographie à clé publique, les messages sont des nombres m inférieurs à n. On demandera d'élever le nombre m: à une très grande puissance e, où e est un nombre qui est également public. Le message encrypté, a, est le reste de la division de m^e par n. Toujours surprenant, il est facile pour un ordinateur de calculer ce reste, même si m et e ont chacun 200 chiffres. Nous allons voir comment. Le décryptage se fait de manière symétrique en calculant le reste de la division de a^d par n. Mais, d est inconnu... et, pour le calculer, il faut connaître p et q, et donc, factoriser n. Donc, seul le concepteur qui a choisi p et q peut lire le message.

Combien de temps résistera le code RSA ? Si ce n'était que des progrès de la factorisation des entiers et de l'augmentation de la puissance des ordinateurs, il pourrait résister encore longtemps, quitte à allonger un peu la clé n. Mais c'est sans compter sur la présence possible d'un nouveau venu, l'*ordinateur quantique*. Cet ordinateur n'existe pas encore. Par contre, un algorithme rapide de factorisation sur un ordinateur quantique, *l'algorithme de Shor*, existe depuis 1997. Nous allons nous pencher sur l'idée de cet algorithme.

Comme 1 est le pgcd[1] de e et de $\phi(n)$, on peut écrire via l'algorithme d'Euclide

$$1 = 10 \times 1\,456 - 633 \times 23$$

Donc

$$1 = 10 \times 1\,456 - (1\,456 - 823) \times 23,$$

ou encore

$$1 = -13 \times 1\,456 + 823 \times 23.$$

Comme $823 \times 23 = 13 \times 1\,456 + 1$, le reste de la division de ed par $\phi(n)$ vaut bien 1.

1. *Voir* « Algorithmes dans l'histoire » *par André Ross, p. 2 de ce numéro.*

Un exemple de fonctionnement du code RSA

Aline veut pouvoir recevoir des messages secrets de Bernard. Elle construit donc un système à clé publique pour recevoir de tels messages. Elle prend les nombres premiers $p = 29$ et $q = 53$. La clé est $n = 1\,537$. Elle aura besoin du nombre

$$\phi(n) = (p - 1)(q - 1) = 1\,456.$$

Elle choisit e relativement premier avec $\phi(n)$, par exemple, $e = 23$. Soit $d = 823$. Il a été construit pour que le reste de la division de ed par $\phi(n)$ soit 1 (voir encadré ci-contre). Pour ce, il fallait connaitre $\phi(n)$, et donc, la factorisation de n, ce qui est difficile pour quelqu'un ne connaissant pas p et q. Aline publie n et e. Bernard veut lui envoyer le message

$$m = 1\,234.$$

Pour le crypter, il l'élève à la puissance $e = 23$ et le divise par n. Ceci lui donne le message crypté,

$$a = 1\,300,$$

qu'il envoie à Aline. Celle-ci utilise d connu d'elle seule, pour calculer le reste de la division de a^d par n. Ce reste est précisément le message $m = 1\,234$.

L'exemple était avec de petits nombres. Mais, comment faire les calculs quand les nombres sont gros ? Il faut se rappeler que ce qui nous intéresse, ce ne sont pas les nombres m^e et a^d, mais seulement le reste de leur division par n. Alors, il ne faut pas laisser grossir les calculs. Il faut plutôt alterner entre prendre des puissances et diviser par n. (Voir détails dans l'encadré.)

Construire de grands nombres premiers

L'idée est la suivante. Si on veut construire un grand nombre premier de 100 chiffres, on génère au hasard un nombre de 100 chiffres et on « teste » s'il est premier. S'il ne l'est pas, on recommence avec un autre nombre généré au hasard, jusqu'à ce qu'on tombe sur un nombre premier.

Calculer le reste de la division de m^e par n

Écrivons la décomposition de $e = 23$ en somme de puissances de 2.

$$e = 1 + 2 + 4 + 16.$$

Alors

$$m^e = m \cdot m^2 \cdot m^4 \cdot m^{16}.$$

On va donc y aller pas à pas pour calculer ces puissances et, à chaque étape, on va diviser par n.

Le reste de la division de m^2 par n est 1 126. Remarquons que $m^4 = (m^2)^2$. Le reste de la division de m^4 par n est donc le même que le reste de la division de $1\,126^2$ par n :

$$1\,126^2 = 823n + 1\,388.$$

De même, le reste de la division de m^8 par n est le même que le reste de la division de $1\,388^2$ par n :

$$1\,388^2 = 1\,253n + 683.$$

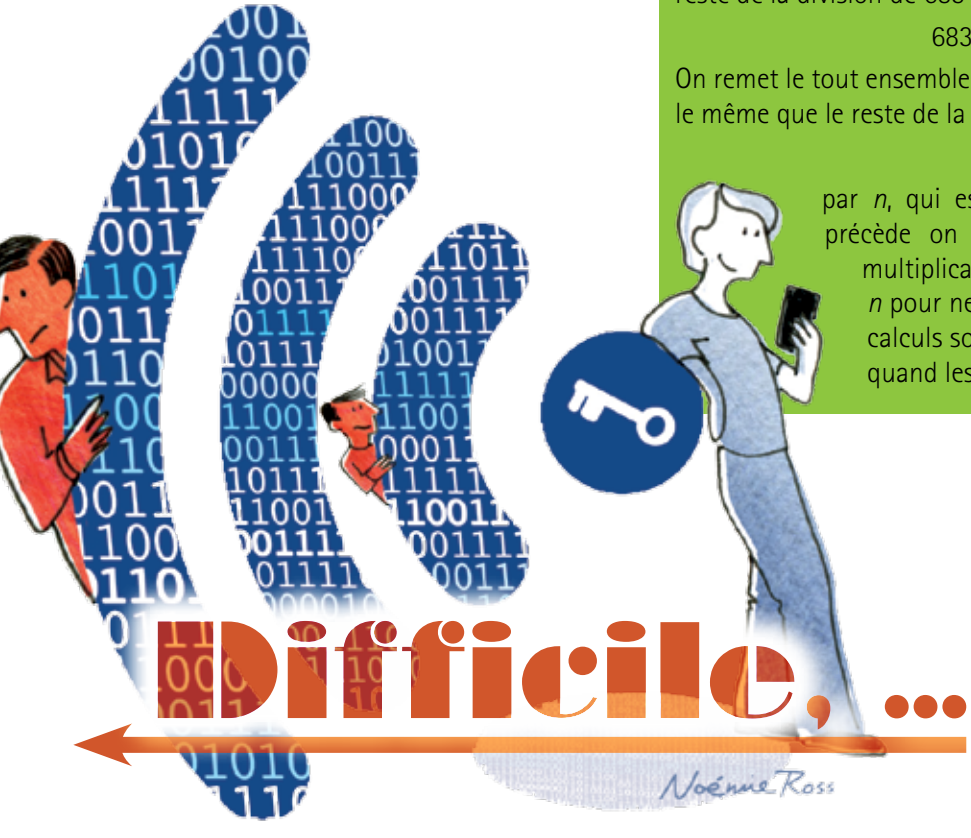
Finalement, le reste de la division de m^{16} par n est le même que le reste de la division de 683^2 par n :

$$683^2 = 303n + 778.$$

On remet le tout ensemble. Le reste de la division de m^{23} par n est le même que le reste de la division de

$$1\,234 \cdot 1\,126 \cdot 1\,388 \cdot 778$$

par n, qui est bien $a = 1\,300$. Dans la ligne qui précède on peut aussi alterner entre faire des multiplications et diviser les produits partiels par n pour ne pas laisser grossir les expressions. Ces calculs sont faciles pour des ordinateurs même quand les nombres sont très grands.



Première question : combien faut-il faire de tests en moyenne ? Un nombre de 100 chiffres est un nombre de l'ordre de 10^{100}. Le théorème des nombres premiers nous dit que si N est grand, il y a environ $N/\ln N$ nombres premiers inférieurs ou égaux à N. Si on choisit au hasard un nombre inférieur ou égal à $N = 10^{100}$, la chance qu'il soit premier est donc d'environ

$$\frac{1}{\ln N} = \frac{1}{\ln 10^{100}} = \frac{1}{100 \ln 10} = \frac{1}{230}.$$

On peut faire mieux : la moitié des nombres sont pairs et, parmi les nombres impairs, on peut éliminer les multiples de 5. Donc, si on choisit au hasard un nombre inférieur ou égal à $N = 10^{100}$ qui se termine par 1, 3, 7 ou 9, on a une chance sur 92 qu'il soit premier (40% des nombres se terminent par 1, 3, 7, 9, et 4/10 de 230 donne 92). Ceci signifie qu'en moyenne, après 92 tests, on a trouvé un nombre premier. Faire 92 tests est **facile** pour un ordinateur.

Ceci nous amène à la deuxième question : que signifie « tester » qu'un nombre p est premier ? On a appris à le faire en cherchant si p a un facteur premier inférieur ou égal à $\sqrt{p}$, ce qui revient à factoriser partiellement p. Mais, factoriser est **difficile** pour un ordinateur…

On peut faire mieux. S'il est **facile** de tester si un nombre p est premier, c'est parce qu'on peut le faire *sans factoriser p*. L'idée est la suivante. Si p n'est pas premier, il laisse ses *empreintes* partout. Dans le test de primalité de Miller-Rabin, si p n'est pas premier au moins les trois-quarts des nombres entre 1 et p ont une empreinte de p, c'est-à-dire qu'ils sont des *témoins* du fait que p n'est pas premier. Le test pour décider si un nombre est un témoin est un peu technique (voir encadré page ci-contre). Mais, ce test est **facile** pour un ordinateur. On choisit au hasard des nombres $a_1, \ldots, a_m \leq p$. Si l'un des a_i est un témoin, on conclut que p n'est pas premier. Si aucun des a_i n'est un témoin, alors p a une très grande chance d'être premier. Par exemple, si $m = 100$, la chance que p ne soit pas premier, sachant que $a_1, \ldots, a_m$ ne sont pas des témoins est inférieure ou égale à 10^{-58}, soit très, très petite.

Peut-on se fier à un algorithme probabiliste ? Les informaticiens le font régulièrement dès qu'ils ont besoin de grands nombres premiers et on n'observe pas de catastrophe autour de nous. On voit d'ailleurs qu'on pourrait transformer cet algorithme probabiliste de primalité en algorithme déterministe : il suffirait de tester au moins le quart des nombres $a \in \{2, \ldots, p-1\}$. Si aucun n'est un témoin, alors on peut affirmer avec certitude que p est premier. Mais, cet algorithme serait sans intérêt car il requerrait de regarder si $p/4$ nombres sont des témoins, alors que l'algorithme usuel de factorisation demande de regarder si p a un facteur premier inférieur ou égal à $\sqrt{p}$, et

$$\sqrt{p} < \frac{p}{4} \text{ dès que } p > 16.$$

Casser le code RSA sur un ordinateur quantique

En 1997, Peter Shor a annoncé un algorithme qui casserait le code RSA sur un ordinateur quantique, qui n'existe pas encore… Que signifie cette affirmation ? L'algorithme de Shor est simplement un autre algorithme de factorisation. Il fonctionnerait sur un de nos ordinateurs, mais il serait moins bon que les meilleurs algorithmes de factorisation. La limite de nos ordinateurs est le parallélisme. On ne peut faire qu'un nombre limité d'opérations en parallèle.

Quelques avantages de la cryptographie à clé publique

La cryptographie à clé publique est très utile quand des milliers de personnes, par exemple des clients, peuvent vouloir envoyer leur numéro de carte de crédit à une même compagnie.

La méthode d'encryptage est publique et seule la compagnie peut décrypter les messages chiffrés.

Un autre avantage est que deux interlocuteurs n'ont pas besoin d'échanger de l'information secrète, par exemple une clé, pour communiquer de manière sécuritaire. Il suffit que chacun publie son système de cryptographie à clé publique.

Le système RSA permet de signer un message pour s'assurer qu'il provient bien de la bonne personne. Dans notre exemple, pour que Bernard signe son message il faut qu'il construise son propre système à clé publique, dont il publie la clé n' et la clé de cryptage e'. Il se servira de sa clé de décryptage d' pour apposer sa signature sur le message.

Un ordinateur est composé de transistors qui fonctionnent à la manière d'interrupteurs en ayant deux positions, OUVERT et FERMÉ, que l'on peut coder comme des bits d'information 0 ou 1. Un nombre de 200 chiffres est inférieur à $10^{200} < 2^{665}$, et donc peut s'écrire avec 665 bits. Tester s'il a un diviseur premier inférieur à $10^{100} < 2^{333}$ revient à regarder toutes les suites de 333 bits égaux à 0 ou 1. Un travail titanesque, sauf si l'ordinateur a un grand parallélisme. Dans un ordinateur quantique, les bits d'information sont des *bits quantiques* : ils peuvent être dans un *état superposé* entre 0 et 1. L'image que l'on pourrait donner est celle d'un sou. Une fois lancé, il tombe sur PILE ou sur FACE. Mais, avant de le lancer, il a probabilité 1/2 de tomber sur PILE, et probabilité 1/2 de tomber sur FACE. On pourrait dire qu'il est dans un état superposé. Le fait que m bits quantiques soient dans un état superposé implique qu'ils peuvent faire simultanément les 2^m opérations différentes correspondant à tous les choix de valeurs 0 ou 1 pour chacun des bits. Par contre, de même que notre sou tombe nécessairement sur PILE ou sur FACE, dès qu'on veut observer un bit quantique, il prend nécessairement la valeur 0 ou la valeur 1, et donc, on ne peut observer le résultat de tous les calculs faits en parallèle. La subtilité de l'algorithme de Shor vient du fait qu'on pourra récupérer le résultat du calcul, lequel donne une factorisation de p si p n'est pas premier.

Le nombre 15 a été factorisé sur un ordinateur quantique en 2007, et en 2012 on a factorisé 143 = 11x13. Les progrès semblent lents depuis ce temps. Le défi technologique de l'ordinateur quantique est de maintenir de nombreux bits en état superposé. Ce rêve deviendra-t-il réalité ? Même si les progrès semblent lents, les experts y croient de plus en plus.

Le fonctionnement de l'algorithme de Shor

On veut factoriser un entier n. L'algorithme de Shor cherche un diviseur d de n qui soit différent de 1 et de n. Ensuite, on peut itérer en cherchant un diviseur de l'entier $S = n/d$. Voyons qu'il suffit de trouver un entier r tel que n divise $r^2 - 1 = (r-1)(r+1)$, mais n ne divise, ni $r-1$, ni $r+1$. Puisque n divise $r^2 - 1$, il existe un entier m tel que $r^2 - 1 = mn$. Soit p, un facteur premier de n. Alors, p divise $r-1$, ou encore p divise $r+1$. Dans le premier cas, d est le pgcd de $r-1$ et n, et dans le deuxième cas, celui de $r+1$ et n. Calculer le pgcd de deux nombres par l'algorithme d'Euclide[3] est facile pour un ordinateur. Comment construire un tel entier r est un peu technique et nous ne ferons pas les détails.

3. *Voir « Algorithmes au cours de l'histoire » par André Ross, p. 2 de ce numéro.*

Tester si un nombre est un témoin

On veut tester si p impair est premier. Alors, on peut écrire $p - 1$, qui est pair, comme $p-1 = 2^s d$, où d est impair. (Il suffit de diviser $p - 1$ par 2 successivement jusqu'à ce que le quotient soir impair.) Soit $a \in \{2, ..., p-1\}$.

a		
2	8	12
3	1	1
4	12	1
5	8	12
6	8	12
7	5	12
8	5	12
9	1	1
10	12	1
11	5	12
12	12	1

Critère pour être témoin : Si a est un témoin que p n'est pas premier, alors on a simultanément que le reste de la division de a^d par p est différent de 1 et que les restes de la division de $a^{2^r d}$ par p sont différents de $p - 1$, pour tous les $r \in \{0, ..., s-1\}$

Regardons un premier exemple. Prenons $p = 13$. Alors $p - 1 = 2^2 \cdot 3$. Ici, $d = 3$ et $s = 2$. Voyons que p n'a aucun témoin.

La première colonne du tableau à gauche donne a, la deuxième colonne donne le reste de la division de $a^d = a^3$ par p, et la troisième colonne, le reste de la division de $a^{2d} = a^6$ par p.

On voit bien que pour chaque a, soit on a un 1 dans le première colonne, ou bien un 1 ou un 12 dans la deuxième colonne, et donc, aucun a n'est un témoin de 13. On sait alors que 13 est premier.

Refaisons l'exercice avec $p = 15$. Alors, $p - 1 = 2 \cdot 7$. Comptons combien p a de témoins. Ici, la deuxième colonne représente le reste de la division de $a^d = a^7$ par p. Tous les a sauf 14 sont des témoins et chacun nous dit que 15 n'est pas premier !

a	
2	8
3	12
4	4
5	5
6	6
7	13
8	2
9	9
10	10
11	11
12	3
13	7
14	14

Mais, bien sûr, apprendre qu'un nombre a est un témoin que p n'est pas premier ne nous apprend rien de la factorisation de p.