Notes de cours
Semaine 14

Cours Turing

1 La recherche de (très) grands nombres premiers

Bon nombre d'algorithmes de cryptographie à clé publique, que nous découvrirons dans les prochaines semaines, reposent sur un ingrédient essentiel : l'utilisation de grands nombres premiers (pour rappel, un nombre premier est un nombre qui n'admet que deux diviseurs distincts : 1 et lui-même). Par "grands nombres premiers", on entend ici de *vraiment grands* nombres premiers, par exemple des nombres à 100 chiffres. La tâche de trouver de tels nombres est un peu ardue, mais nous allons progresser pas à pas.

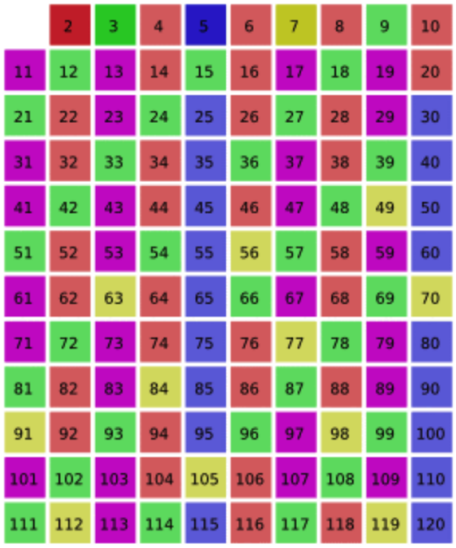
1.1 Tester tous les diviseurs

Une première méthode naïve pour tester si un grand nombre N est premier est d'essayer de le diviser par tous les nombres compris entre 1 et $N-1$. Mais si N est un nombre à 100 chiffres, il y a 10^{100} diviseurs potentiels à tester, à savoir plus que le nombre d'atomes dans l'univers ! Donc cette méthode reste complètement inapplicable dans ce cas.

On peut réduire sensiblement ce nombre d'opérations en observant la chose suivante : si un nombre N n'est pas premier et est donc le produit de deux nombres P et Q, c'est forcément le cas que soit P soit Q est plus petit ou égal à $\sqrt{N}$ (sinon, on obtiendrait que $P \cdot Q > \sqrt{N} \cdot \sqrt{N} = N$). Et donc, pour tester si un nombre N est premier, il suffit de tester tous ses diviseurs potentiels de 2 jusqu'à $\sqrt{N}$, et non de 2 jusqu'à N. Alors certes, $\sqrt{N}$ est beaucoup plus petit que N, mais lorsque $N = 10^{100}$, on obtient que $\sqrt{N} = 10^{100/2} = 10^{50}$, ce qui représente un nombre encore extrêmement grand !

1.2 Le crible d'Eratosthène (environ 200 avant J.-C.)

Une autre méthode pour trouver des nombres premiers est la suivante : l'idée est de dresser la liste de tous les nombres, puis de rayer systématiquement tous les nombres qui sont des multiples de 2, de 3, de 5, etc. (notez que 4 étant un multiple de 2, il n'y pas besoin de rayer les multiples de 4, qui sont donc tous aussi des multiples de 2) : le nombres qui restent sont les nombres premiers. Voici une illustration de cette méthode, où les nombres premiers sont ceux colorés en rose (auxquels il faut encore ajouter les premiers nombres de chaque autre couleur) :



Cette méthode est certes très intéressante pour établir la liste des nombres premiers existants, mais devient de plus en plus lente lorsqu'on cherche de plus en plus grands nombres : ça n'est donc pas ainsi qu'il faut procéder pour trouver des nombres premiers à 100 chiffres...

1.3 Le théorème des nombres premiers (Hadamard et de la Vallée Poussin, 1896)

Pour de plus grands nombres, une question qui vient tout de suite à l'esprit est la suivante : est-ce que des nombres premiers à 100 chiffres existent ? (car si tel n'est pas le cas, mieux vaut arrêter tout de suite...). A priori, vu qu'un nombre à 100 chiffres est de l'ordre de 10^{100}, il a donc aussi de l'ordre de 10^{100} diviseurs potentiels : il est donc justifié de se poser la question ! C'est cependant un fait qui est connu depuis Euclide : il existent une infinité de nombres premiers.

Mais vient maintenant la deuxième question : vu que plus un nombre grandit, plus il a de diviseurs potentiels, il doit donc être vrai que les nombres premiers se *raréfient* plus on va vers de grands nombres. Par exemple :

- parmi les 10 premiers nombres, on trouve 4 nombres premiers (donc 40%) ;

- parmi les 100 premiers nombres, on trouve 25 nombres premiers (donc 25%) ;

- parmi les 1000 premiers nombres, on trouve 168 nombres premiers (donc 17% environ) ;

- etc.

Si les nombres premiers se raréfiaient trop lorsque qu'on va vers de grandes valeurs, la recherche de grands nombres premiers serait problématique... Heureusement, le *théorème des nombres premiers* nous dit que cette raréfaction est plutôt lente, à savoir :

$\pi(N)$, le nombre de nombres premiers plus petits ou égaux à un nombre donné N,
est approximativement donné par $\pi(N) \simeq \frac{N}{\ln(N)}$

où $\ln(N)$ est le *logarithme népérien de* N. Ceci veut dire que la *proportion* de nombres premiers plus petits ou égaux à N est de l'ordre de $\frac{1}{\ln(N)}$.

La fonction $\ln(N)$ est une fonction qui grandit très lentement avec N (par exemple, $\ln(1'000) \simeq 6.9$, tandis que $\ln(1'000'000) \simeq 13.8$), donc la proportion $\frac{1}{\ln(N)}$ décroît très lentement avec N.

A nouveau, si N est un nombre à 100 chiffres, qu'implique ce résultat ? Dans ce cas, $N \simeq 10^{100}$ et on peut calculer que $\ln(N) \simeq 230$: même si N est gigantesque, $\ln(N)$ a une valeur assez faible : ceci veut dire en pratique que si on tire complètement au hasard un nombre à 100 chiffres, on a environ une chance sur 230 de tomber sur un nombre premier ; c'est une probabilité certes assez faible, mais suffisamment grande pour ne pas se décourager, car en effectuant 230 essais, on obtient une probabilité non-négligeable de tomber au moins une fois sur un nombre premier. Reste maintenant la question épineuse de trouver une manière efficace de *tester* si un nombre donné N est premier : il existe heureusement une façon efficace de procéder ! Mais elle demande un peu de travail, en particulier de revenir à l'arithmétique modulaire.

1.4 Le petit théorème de Fermat

En quoi l'arithmétique modulaire peut-elle nous être utile pour trouver des nombres premiers ? La première relation avec les nombres premiers est la suivante, aussi connue sous le nom de *petit théorème de Fermat*[1] :

Si N est un nombre premier, alors $A^{N-1} \pmod N = 1$ pour tout nombre entier $1 \leq A < N$.

Malheureusement, l'assertion ne va pas dans le bon sens ! En effet, si on trouve un nombre $1 \leq A < N$ tel que $A^{N-1} \pmod N = 1$, ceci n'implique pas que N est un nombre premier.

Par contre, on peut utiliser la *contraposée* de ce théorème, à savoir :

S'il existe un nombre entier $2 \leq A < N$ tel que $A^{N-1} \pmod N \neq 1$, alors N n'est *pas* un nombre premier (noter qu'il est impossible ici que $A = 1$, car $1^{N-1} \pmod N = 1$ pour tout N).

Ainsi, on a un outil pour dénicher des grands nombres qui ne sont *pas* premiers. "A quoi bon ?" direz-vous, car ce qui nous intéresse, ce sont les grands nombres premiers... Il se trouve qu'une extension du petit théorème de Fermat va nous aider. Celle-ci dit la chose suivante :

S'il existe un nombre $2 \leq A \leq N-1$ tel que $\mathrm{PGDC}(A, N) = 1$ et $A^{N-1} \pmod N \neq 1$, alors $B^{N-1} \pmod N \neq 1$ pour au moins la moitié des autres nombres B compris entre 2 et $N-1$.

Ce que dit ce résultat pour l'essentiel (si on fait abstraction d'un petit détail ; voir la 2e remarque ci-dessous), c'est : *si on choisit un nombre A uniformément au hasard entre 2 et $N-1$ et qu'on trouve que $A^{N-1} \pmod N = 1$, alors il y a au moins 50% de chances pour que N soit un nombre*

1. Attention à ne pas confondre ici avec le *grand* ou *dernier* théorème de Fermat, qui est resté un problème ouvert pendant de nombreuses années, plus précisément de 1637, date de son énoncé, à 1995, date de la parution de sa démonstration par Andrew Wiles.

premier. En effet, puisque si N n'était pas premier, au moins la moitié des nombres A entre 2 et $N-1$ donneraient $A^{N-1} \pmod N \neq 1$). Nous allons maintenant voir en détail quelle utilisation faire de ce résultat pour transformer l'essai, car il est clair qu'on ne peut pas se satisfaire d'un algorithme qui se trompe une fois sur deux !

1.5 Bis repetita...

L'idée est maintenant de répéter l'expérience précédente, en tirant plusieurs nombres $A_1, A_2, \ldots, A_k$ au hasard, chacun entre 2 et $N-1$, et indépendamment les uns des autres (d'où l'importance d'avoir un bon générateur de nombres aléatoires). Pour tous ces nombres, on teste si

$$A_1^{N-1} \pmod N = 1, \quad A_2^{N-1} \pmod N = 1, \quad \ldots, \quad A_k^{N-1} \pmod N = 1$$

- Si on n'obtient pas le résultat 1 pour un des nombres $A_1, A_2, \ldots, A_k$, alors on sait par le petit théorème de Fermat (plus précisément, par sa contraposée), que N n'est pas premier. Fin de la discussion : il faut essayer une nouvelle valeur de N.

- Si par contre c'est le cas qu'on obtient le résultat 1 pour chacun des nombres $A_1, A_2, \ldots, A_k$, alors qu'est-ce que ça nous dit ? Par ce qui précède, *si N n'est pas un nombre premier*, alors il y a au moins 50% de chances, pour chaque nombre A, que $A^{N-1} \pmod N \neq 1$. Donc dans ce cas, il faudrait être vraiment malchanceux pour obtenir le résultat 1 pour chacun des nombres $A_1, A_2, \ldots, A_k$. Plus précisément, la chance que ça arrive vaut au plus :

$$p = \underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdots \frac{1}{2}}_{k \text{ fois}} = \frac{1}{2^k}$$

Pour une valeur même assez petite de k, cette probabilité p est très proche de 0. Par exemple : si $k = 10$, alors $p \simeq 0,001$; si $k = 20$, alors $p \simeq 0,000001$, et si $k = 30$, alors $p \simeq 0,000000001$. En conclusion : si on n'obtient que des résultats 1 aux k tests ci-dessus, on peut déclarer avec confiance que N est un nombre premier (à noter que si N est un nombre premier, alors par le petit théorème de Fermat, on sait qu'on obtiendra toujours le résultat 1).

Remarques

- Il peut sembler assez troublant que pour vérifier une propriété *déterministe* d'un nombre entier, à savoir ici sa primalité, on ait recours au *hasard*. Par ce qui précède, vous voyez cependant que ce hasard peut être réduit "autant qu'on veut", en assez peu d'étapes. En pratique, c'est tout à fait acceptable ! Il existe d'ailleurs une pléthore d'algorithmes qui font appel au hasard pour résoudre des problèmes déterministes. Sans le hasard, nos ordinateurs modernes seraient beaucoup moins puissants !

- Un détail a été omis dans la présentation qui précède : il existe quelques nombres entiers N (pas si nombreux, mais quand-même), qui possèdent la propriété étrange de ne pas être premiers tout en vérifiant $A^{N-1} \pmod N = 1$ pour *toutes* les valeurs de A entre 2 et $N-1$. Ces nombres sont appelés les *nombres de Carmichael*. Le plus petit d'entre eux est $N = 561 = 17 \cdot 33$. Pour gérer ce problème, un autre algorithme est nécessaire : l'*algorithme de Miller-Rabin*.

Et pour finir, reste une question cruciale : est-ce qu'avec tout ce travail, nous avons gagné quoi que ce soit par rapport à l'algorithme qui teste tous les diviseurs de N allant de 2 à $\sqrt{N}$? ? ? Faisons un peu les comptes pour voir...

Comme mentionné au début de ce chapitre, l'algorithme classique demande d'effectuer de l'ordre de 10^{50} divisions pour vérifier qu'un nombre à 100 chiffres est premier. De plus, comme déjà vu ci-dessus, en tirant au hasard un nombre à 100 chiffres, on a environ une chance sur 230 de tirer un nombre premier. Au total, on trouvera donc avec cette méthode un nombre premier après

$$230 \cdot 10^{50} \text{ opérations}$$

en moyenne ; autrement dit, mieux vaut être patient !

Comparons maintenant avec la méthode vue plus haut : certes, on n'échappe pas au fait qu'il faut tirer de l'ordre de 230 nombres N au hasard pour finalement tomber sur un nombre premier. Mais combien d'opérations coûte à chaque fois le test proposé ci-dessus ? Pour fixer les idées, disons qu'on effectue le test avec $k = 30$ nombres A tirés au hasard (ce qui rappelons-le mène à une probabilité d'erreur inférieure ou égale à 0,000000001) : pour chaque nombre, ceci consiste à calculer

$$A^{N-1} \pmod N \tag{1}$$

Heureusement, comme nous allons le voir ci-dessous, pour un nombre N à 100 chiffres, ce calcul ne demande pas plus de deux millions d'opérations. Donc au total, le nombre d'opérations à effectuer pour trouver un nombre premier à 100 chiffres avec cette méthode est de l'ordre de

$$230 \cdot 30 \cdot 2 \text{ millions} \simeq 10 \text{ milliards}$$

Or 10 milliards d'opérations, avec un ordinateur moderne, ça se fait très vite (en tout cas beaucoup plus vite que $230 \cdot 10^{50}$ opérations !).

Reste à comprendre pourquoi l'opération (1) ne demande pas plus de deux millions d'opérations. Pour cela, il nous faut (clairement...) revenir un moment à l'arithmétique modulaire.

1.6 Exponentiation rapide ("square-and-multiply")

Nous avons vu il y a deux semaines que "prendre des modulos" permet de simplifier des additions et multiplications, mais c'est encore plus vrai lorsqu'on effectue une opération d'*exponentiation* (qui commute également avec l'opération modulo) : pour calculer $A^B \pmod N$, on peut bien sûr d'abord calculer A^B, puis prendre la reste de la division de ce nombre par N pour trouver le résultat, mais il est clairement plus facile d'utiliser le fait que

$$A^B \pmod N = (A \pmod N)^B \pmod N$$

Voyons ça sur un exemple avec $N = 11$, $A = 64$ et $B = 6$: pour calculer $64^6 \pmod{11}$, calculons d'abord $64 \pmod{11} = 9$, puis

$$64^6 \pmod{11} = 9^6 \pmod{11} = 531'441 \pmod{11} = 9$$

? ? ? A vous entendre, vous ne semblez pas forcément convaincus... Certes, il est plus facile de calculer 9^6 que 64^6, mais ça reste quand-même un calcul ardu ! En fait, on peut faire mieux que ça. Regardez plutôt :

$$9^6 = 9^{4+2} = 9^4 \cdot 9^2 = (9^2)^2 \cdot 9^2 \quad \text{donc} \quad 9^6 \,(\text{mod } 11) = (9^2 \,(\text{mod } 11))^2 \,(\text{mod } 11) \cdot (9^2 \,(\text{mod } 11))$$

Pour effectuer ce calcul, on calcule d'abord $9^2 \,(\text{mod } 11) = 81 \,(\text{mod } 11) = 4$, puis $4^2 \,(\text{mod } 11) = 16 \,(\text{mod } 11) = 5$, et finalement la multiplication $(5 \cdot 4) \,(\text{mod } 11) = 20 \,(\text{mod } 11) = 9$, qui nous donne le résultat voulu : $64^6 \,(\text{mod } 11) = 9$, sans avoir eu à effectuer de multiplications plus compliquées que des livrets appris à l'école primaire.

Remarque : Attention ! Pour calculer $A^B \,(\text{mod } N)$, on peut remplacer A par $A \,(\text{mod } N)$, mais on ne peut *pas* remplacer B par $B \,(\text{mod } N)$. Par exemple :

$$64^{17} \,(\text{mod } 11) \neq 64^6 \,(\text{mod } 11)$$

En général, voyons maintenant comment cette méthode fonctionne : pour calculer A^B tout d'abord (en oubliant mod N pour l'instant), on utilise la décomposition binaire de B, c'est-à-dire qu'on écrit B comme une somme de puissances de 2. Par exemple, pour $B = 43$, écrivons $B = 43 = 32 + 8 + 2 + 1$, et donc

$$A^B = A^{32+8+2+1} = A^{32} \cdot A^8 \cdot A^2 \cdot A$$

Ainsi, en calculant successivement A, A^2, $A^4 = (A^2)^2$, $A^8 = (A^4)^2$, $A^{16} = (A^8)^2$ et $A^{32} = (A^{16})^2$, on n'effectue que des carrés, et pour obtenir le résultat A^B, il suffit de multiplier les quatre termes A, A^2, A^8 et A^{32}. Cette méthode, qui s'appelle en anglais "square-and-multiply", permet d'éviter le calcul direct de A^{43}, qui est fastidieux.

Viennent maintenant s'ajouter les modulos : comme déjà mentionné ci-dessus, l'avantage d'effectuer des opérations modulo permet de se cantonner aux nombres compris entre 0 et $N - 1$. Ainsi, si A^B est potentiellement un (très) grand nombre, $A^B \,(\text{mod } N)$ sera par contre toujours un nombre compris entre 0 et $N - 1$. Donc on peut refaire tout ce qu'on vient de faire en prenant chaque fois systématiquement le résultat modulo N. Ainsi, on sait qu'on n'effectuera que des carrés ou des multiplications avec des nombres compris entre 0 et $N - 1$.

Comptons maintenant le nombre d'opérations effectuées, en supposant que A, B, N sont tous des nombres à 100 chiffres :

- effectuer le carré d'un nombre à 100 chiffres ou la multiplication de deux nombres à 100 chiffres coûte $100 \cdot 100 = 10'000$ opérations environ (penser à la multiplication en colonnes) ;

- si B est un nombre à 100 chiffres, sa décomposition binaire sera également composée d'une centaine de termes (à peu près) ;

- et donc pour calculer la série A, A^2, A^4, A^8... jusqu'à $A^{2^{100}}$ (le tout modulo N), il faudra effectuer environ 100 carrés, et encore 100 autres multiplications (au pire) des termes dont on a besoin pour obtenir le résultat final A^B. Ceci représente au total

$$2 \cdot 100 \cdot 10'000 = 2 \text{ millions d'opérations}$$

comme annoncé plus haut ! Nous voilà donc arrivés à notre but, à savoir : effectuer un nombre raisonnable d'opérations pour trouver un (très) grand nombre premier.