

blème particulier à chaque demande. Mais la première méthode permet une prise en compte plus fine de la surcharge du service à l'instant de la demande ou d'informations connues de S concernant D .

Le délai nécessaire pour que le demandeur résolve le problème posé est le plus souvent fondé sur un calcul à faire par le demandeur. Le temps qu'il met à résoudre le problème dépend donc de sa puissance de calcul et les machines puissantes sont avantagées. Aussi a-t-on envisagé des preuves de travail exigeant beaucoup de mémoires (plutôt que beaucoup de calculs) ou nécessitant la recherche d'informations sur le réseau. Dans ce dernier cas, les machines puissantes ne sont pas avantagées, puisque ce qui nécessite du temps est lié à la vitesse de circulation des informations sur le réseau, qui est la même pour tout le monde.

L'idée des preuves de travail est due à Cynthia Dwork et Moni Naor qui, dès 1993, suggérèrent cette méthode pour lutter contre les spams. Le système *Hashcash* conçu par Adam Back en 1997 (indépendamment de C. Dwork et M. Naor) et servant aujourd'hui de base dans la plupart des mises en œuvre pratiques des preuves de travail, utilise des « fonctions à sens

unique » pour définir les problèmes soumis et en contrôler finement la difficulté (voir l'encadré 2 pour des exemples).

Une fonction f à sens unique se calcule facilement, mais s'inverse difficilement. Autrement dit, il est rapide de passer de x à $f(x) = y$, mais difficile de trouver, à partir d'un y donné, un x tel que $y = f(x)$. On exigera souvent aussi que les valeurs de $f(x)$ quand x varie se présentent comme si elles étaient tirées au hasard.

Fonctions à sens unique, avec ou sans paramètre

Si l'on dispose d'une telle fonction f , on en déduit facilement une preuve de travail : le service S choisit un y et exige du demandeur D qu'il trouve un x tel que $f(x) = y$. Le demandeur ne peut guère faire mieux qu'essayer des valeurs de x , jusqu'à trouver la bonne, ce qui est impossible à faire rapidement. Il s'agit ici d'une impossibilité pratique : trouver le x est possible en essayant de manière ordonnée toutes les x envisageables, mais cela nécessite du temps.

Plus subtil, car cela ne demande pas l'intervention de S pour la formulation du problème (donc utilisable pour les preuves de travail sans échange), on utilise une

fonction à sens unique dépendant d'un paramètre p , $f_p(x)$, et dont le résultat, y , peut être interprété comme un nombre réel positif. Une telle interprétation est toujours possible : on écrit y en binaire, par exemple 01001110, et on assimile cette suite de 0 et de 1 au nombre réel dont l'écriture en base 2 est 0,01001110. Le service S attend que D lui propose un x tel que $f_p(x) < y$, où p est par exemple le message que D veut envoyer et y un nombre réel, appelé seuil, fixé à l'avance par S . La valeur de y détermine très précisément la difficulté de la preuve de travail, car plus y est petit, plus il est difficile de trouver un x convenable.

Les preuves de travail fournissent aussi un moyen simple d'organiser des courses entre machines sur un réseau. Or ces courses sont au cœur des protocoles de fonctionnement des cryptomonnaies.

Ces monnaies, dont le *bitcoin* est l'exemple le plus connu, fonctionnent sans autorité centrale, la gestion des comptes étant opérée par les machines des volontaires qui se contrôlent mutuellement. Ces contrôles empêchent toute manipulation du cahier de compte de la monnaie (la « blockchain »), lequel indique qui détient des devises (voir *Pour la Science*, décembre 2013).

1. L'idée des preuves de travail

Pour limiter le pouvoir de nuisance que chacun possède du fait de la puissance de son ordinateur, on peut soumettre des énigmes obligeant les machines à de longs calculs et dont seul l'aboutissement leur donne le droit de mener une action, comme envoyer un message électronique ou accéder à un service en ligne. Une machine soumise à ces énigmes ne pourra pas envoyer des milliers de messages non sollicités (spams), ou participer à une tentative de mise en panne d'un service informatique en le submergeant de requêtes (attaque par déni de service).

Le temps de calcul imposé est un prix à payer pour que l'ordinateur ac-

quière le droit de mener une action. Il doit travailler et le prouver par un résultat. La solution produite est la preuve qu'il a fourni ce travail. C'est l'idée des « preuves de travail ».

Les problèmes qu'on demande de résoudre pour ces défis doivent être asymétriques. Il faut que le calcul de la solution prenne du temps, mais que la vérification soit rapide ; sinon, celui qui se protégerait par les preuves de travail serait autant ennuyé que ceux dont il tenterait de se protéger.

On connaît une multitude de problèmes asymétriques. En voici deux exemples :

– L'entier b étant fixé, et p étant un grand nombre premier, trouver un



nombre a tel que $a^2 = b \text{ mod } p$. Ce problème de la « racine carrée modulo un grand nombre premier » a été proposé comme preuve de travail dès 1993 par Cynthia Dwork et Moni Naor.

– Trouver les deux nombres premiers p et q dont le produit est le

nombre n qu'on vous a donné sans vous avoir dévoilé p ni q . Il est facile de formuler de telles énigmes – on multiplie deux grands nombres premiers entre eux, ce qui produit n – mais aucun algorithme de factorisation rapide n'est connu, et il faut donc du temps pour retrouver p et q à celui qui ne connaît que n . Il ne faut pas prendre p et q trop grands, car le problème de la factorisation de n deviendrait impossible à résoudre en un temps raisonnable.

Un troisième exemple – le plus important, car le plus utilisé – est expliqué en détail dans les encadrés 2 et 3 et se fonde sur les fonctions « à sens unique ».