RENDEZ-VOUS

☐ LOGIQUE & CALCUL

Déléguer un calcul sans divulguer ses données

Vous confiez des calculs à un tiers, il les effectue et vous transmet les résultats, mais sans avoir pu connaître ni les données du calcul ni ses résultats : un nouveau miracle cryptographique !

Jean-Paul DELAHAYE

Si vous êtes prudent, vous chiffrez vos données avant de les confier à un service de sauvegarde extérieur, dans le « cloud » par exemple. Comment demander alors à ce service de prendre les données d'un fichier et d'en calculer la moyenne, ou d'en extraire des informations particulières ? Pour calculer et manipuler des données, il semble qu'il faille les connaître.

Eh bien, non ! C'est tout l'enjeu des systèmes de chiffrement homomorphe : vous permettez à un tiers de calculer avec vos données qu'il ne connaît pas, et il produit un résultat qui lui est illisible, mais que vous savez déchiffrer.

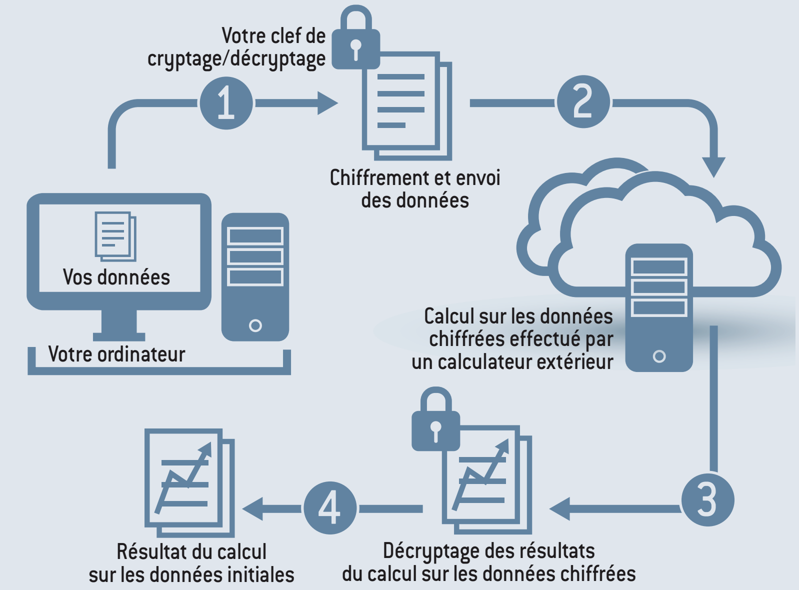
Une méthode très simple : la multiplication

Si les calculs à effectuer sont simples, il est facile d'imaginer une solution. Vous choisissez un nombre C que vous serez le seul à connaître. Vous multipliez toutes vos données par C, et ce sont ces données que vous transmettez à l'opérateur extérieur $\mathbb{O}$ chargé de les sauvegarder. Pour connaître par exemple la moyenne de vos données ou de certaines d'entre elles, vous demandez à $\mathbb{O}$ de calculer la moyenne sur les données dont il dispose et de vous transmettre le résultat. Puis vous divisez le résultat par C : vous avez votre moyenne, car la moyenne de $Cd_1, Cd_2, \ldots, Cd_n$ est C fois la moyenne de $d_1, d_2, \ldots, d_n$.

Calcul délocalisé

Avec les chiffrements dits homomorphes, un opérateur calcule sur des données chiffrées dont il n'a pas connaissance. Une fois le calcul achevé, il ne peut pas connaître le résultat trouvé, que seul le commanditaire du calcul déchiffre.

Vous chiffrez vos données avec une clef secrète et vous les envoyez à l'opérateur extérieur *(1)*. L'opérateur effectue des calculs avec les données cryptées et vous envoie les résultats (cryptés) sans avoir eu connaissance de la clef *(2, 3)*. Vous déchiffrez les résultats reçus grâce à la clef secrète *(4)* : vous avez ainsi connaissance des résultats d'un calcul que vous n'avez pas fait, résultats auxquels l'opérateur n'a pas pu avoir accès. Pour des calculs simples, des séries d'additions et de soustractions, la mise au point d'applications réelles telles que les systèmes de vote est possible.



Avec ce procédé, vous pouvez faire exécuter par l'opérateur extérieur d'autres types de calculs. Pour connaître le nombre de données qui dépassent un certain nombre B, vous lui demanderez le nombre de données qui dépassent CB. Le tri de vos données par ordre croissant est aussi faisable, puisque si C est positif, le tri de $Cd_1, Cd_2, \dots, Cd_n$ exige les mêmes permutations que $d_1, d_2, \dots, d_n$. Toutes les moyennes pondérées, par exemple la moyenne $(d_1 + 2d_2 + 5d_3 + d_4)/9$, sont aussi possibles sans dévoiler C, car le résultat sur les données multipliées par C est égal à C fois le résultat attendu. Certaines opérations un peu plus compliquées sont envisageables : pour le calcul de $d_1d_2 + d_3d_4 + d_5d_6$, il suffira de diviser le résultat que $\mathbb{O}$ fournit par C^2.

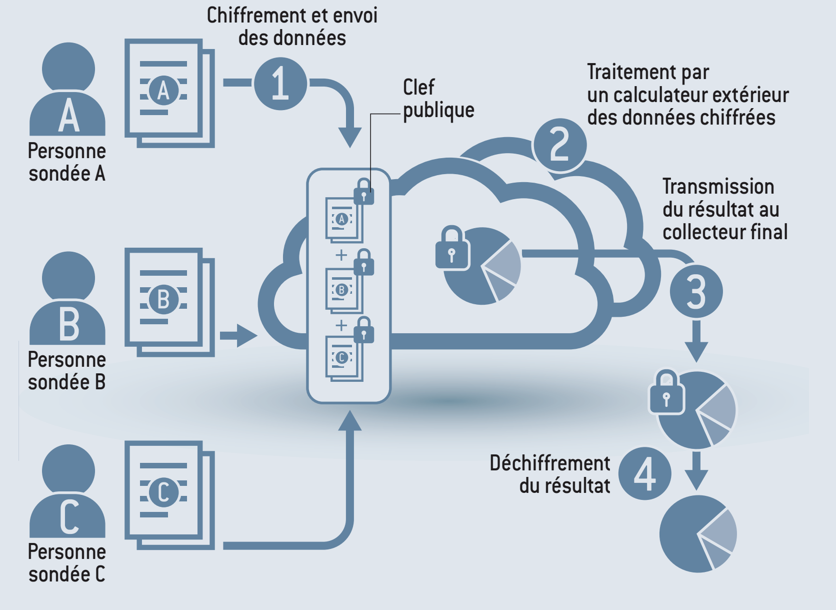
Cependant, vous ne pourrez pas demander $d_1 + d_2d_3$ et, finalement, ce n'est qu'un faible sous-ensemble de calculs que vous saurez faire exécuter à l'extérieur sans avoir à divulguer vos secrets. En particulier, pas question de faire opérer des algorithmes complexes sur vos données, comme l'identification, parmi les nombres confiés, de ceux qui sont des nombres premiers. Remarquons aussi que le chiffrement de vos données en les multipliant par une constante C laisse transparaître certaines informations que vous ne souhaitez peut-être pas que l'opérateur $\mathbb{O}$ connaisse : le numéro de la donnée la plus grande ou la plus petite, les valeurs égales, les données classées, etc.

Le problème sera résolu si vous disposez d'une méthode de chiffrement permettant à la fois de bien masquer toute l'information et autorisant l'exécution d'additions et de multiplications sans rien révéler : avec ces deux opérations, on reconstitue tout calcul.

Préserver la confidentialité

Imaginons que l'on souhaite établir une statistique sur un sujet délicat en utilisant Internet et en garantissant à chacune des personnes sondées que les données individuelles contenues dans les réponses ne seront pas divulguées. La procédure est la suivante :

Chaque personne chiffre ses réponses avec la clef publique du collecteur final et les envoie à un calculateur extérieur *(1)*. Ce calculateur traite ces données cryptées et envoie le résultat au collecteur final *(2, 3)*. Ce dernier déchiffre le résultat et obtient la statistique souhaitée *(4)*. Le calculateur extérieur ne peut pas accéder aux informations privées des sujets de l'enquête car il ne dispose pas de la clef privée de déchiffrement, et le collecteur final ne peut pas accéder aux données individuelles car il n'a pas eu accès aux informations chiffrées individuelles transmises.



La solution : des chiffrements pleinement homomorphes

Notons aussi qu'il n'est pas essentiel que l'opération effectuée par $\mathbb{O}$ soit identique à celle que vous voulez au final. En effet, supposons que le chiffrement de la donnée d soit $\mathscr{C}(d)$ et que le déchiffrement (que vous seul savez effectuer) soit l'opération $\mathscr{C}'$: $\mathscr{C}'(\mathscr{C}(d)) = d$. Pour réussir à faire exécuter toutes sortes de calculs par un opérateur extérieur, il suffirait qu'il existe une opération $*_1$ et une opération $*_2$ (pas nécessairement l'addition et la multiplication habituelles) vérifiant :

$\mathscr{C}(d_1 + d_2) = \mathscr{C}(d_1) *_1 \mathscr{C}(d_2)$ (propriété 1)

$\mathscr{C}(d_1 \times d_2) = \mathscr{C}(d_1) *_2 \mathscr{C}(d_2)$ (propriété 2).

En exécutant $*_1$ et $*_2$ à la place de + et ×, l'opérateur $\mathbb{O}$ produira à partir des données chiffrées un résultat qui, après le déchiffrement par $\mathscr{C}'$, sera exactement ce que vous voulez.

Par exemple, si vous souhaitez connaître $d_1 + d_2d_3$ sans le calculer vous-même (ce qui est impossible avec la méthode de multiplication par C), vous enverrez vos données cryptées $\mathscr{C}(d_1), \mathscr{C}(d_2), \mathscr{C}(d_3)$ à l'opérateur en lui demandant de calculer $\mathscr{C}(d_1) *_1 (\mathscr{C}(d_2) *_2 \mathscr{C}(d_3))$ et de vous renvoyer le résultat R. Grâce à l'opération $\mathscr{C}'$, vous obtiendrez alors la valeur de $d_1 + d_2d_3$, puisque :

$$\mathscr{C}'(R) = \mathscr{C}'(\mathscr{C}(d_1) *_1 (\mathscr{C}(d_2) *_2 \mathscr{C}(d_3))) = \mathscr{C}'(\mathscr{C}(d_1)) + \mathscr{C}'(\mathscr{C}(d_2) *_2 \mathscr{C}(d_3)) = d_1 + \mathscr{C}'(\mathscr{C}(d_2) \times \mathscr{C}'(\mathscr{C}(d_3))) = d_1 + d_2d_3.$$

La méthode fonctionnera pour toute expression, aussi compliquée soit-elle, constituée d'additions et de multiplications. Puisque dès qu'on sait mener des additions

Craig Gentry, le pionnier

© John D. & Catherine T. MacArthur Foundation

En proposant en 2009 le premier système de cryptage complètement homomorphe, Craig Gentry, aujourd'hui chercheur chez IBM, a révolutionné le domaine. Il a déclenché une multitude de travaux qui, petit à petit, ouvrent des voies pour rendre utilisable son idée, initialement théorique pour l'essentiel. Confier de façon massive des calculs à des opérateurs extérieurs qui ne peuvent pas accéder aux données n'est pas aujourd'hui envisageable. En revanche, peu à peu, il est devenu possible de résoudre des problèmes particuliers n'exigeant qu'une petite quantité de calculs homomorphes (comme pour un vote). Les applications de ces méthodes devraient rapidement s'étendre et aider à accroître la sécurité des réseaux et la protection des données… ce dont on a un grand besoin !

et des multiplications de cette façon, on peut exécuter tout algorithme, le problème des calculs secrets par un tiers se ramène à celui de trouver $\mathscr{C}$, $\mathscr{C}'$, $*_1$, $*_2$, vérifiant les propriétés 1 et 2.

Les méthodes de cryptage vérifiant les propriétés 1 et 2 sont dites « pleinement homomorphes ». Le terme vient du langage mathématique où l'on dénomme homomorphisme les fonctions qui transforment une opération en une autre, comme l'indique par exemple la propriété 1.

Ces homomorphismes jouent un rôle très important en mathématiques et chacun connaît le logarithme qui vérifie $\log(a \times b) = \log(a) + \log(b)$ et qui transforme donc des multiplications en additions (ici, $*_2$ est l'addition habituelle). Avant l'avènement des ordinateurs, c'était le moyen le plus efficace pour mener à la main des calculs numériques, et c'est pourquoi on dressait des tables de logarithmes contenant des millions de données.

> Craig Gentry a fait une percée décisive, pour laquelle il s'est vu attribuer une bourse de 625 000 dollars.

En mathématiques, on connaît de nombreux homomorphismes, mais aucun des homomorphismes classiques ne peut servir à concevoir un cryptage $\mathscr{C}$ homomorphe car l'opération de décryptage $\mathscr{C}'$ est alors trop simple ; or une bonne méthode cryptographique est une méthode où il est difficile de trouver $\mathscr{C}'$. Le problème des systèmes de chiffrement pleinement homomorphes n'était donc pas évident au moment où, en 1978, l'idée en a été envisagée par les Américains Ron Rivest et Leonard Adleman et le Grec Michael Dertouzos. On s'est longtemps demandé si de telles méthodes existaient. Le problème a parfois été qualifié de graal de la cryptographie.

Une percée décisive a été réalisée en 2009 par Craig Gentry, de l'université Stanford et aujourd'hui chercheur au centre Thomas Watson d'IBM à New York. Craig Gentry a proposé une méthode qui résout théoriquement le problème et a pour cela reçu une bourse de 625 000 dollars de la fondation MacArthur. La méthode qu'il a initialement proposée n'est pas applicable, car elle exige des calculs trop volumineux au moment du cryptage (la transformation des données avant de les envoyer) et du décryptage. De plus, la taille des données que doit manipuler l'opérateur extérieur $\mathbb{O}$, et donc la quantité de calculs qu'il doit mener, est des millions de fois celle nécessaire quand les données ne sont pas cryptées.

Depuis 2009, une multitude de recherches ont été faites pour trouver des schémas plus économiques. En quelques années, les progrès ont rendu envisageables certaines applications, et l'espoir de méthodes encore plus économiques et faciles à mettre en œuvre suscite de nombreux travaux parmi les spécialistes.

Avant de donner des précisions sur les idées de la percée décisive de Craig Gentry, revenons sur les applications des cryptages pleinement homomorphes ou même partiellement homomorphes.

L'algorithme RSA (conçu en 1977 par Ronald Rivest, Adi Shamir et Leonard Adleman) est une méthode de chiffrement à clef publique : pour crypter un message, on utilise la clef publique de son correspondant, et lui seul peut déchiffrer le message grâce à sa clef secrète. C'est un bon algorithme, aujourd'hui très utilisé. Il a la propriété d'être homomorphe pour la multiplication : on en déduit une méthode de chiffrement homomorphe pour l'addition, comme celle évoquée au début de l'article, méthode ayant cette fois l'avantage que les données sont correctement masquées à l'opérateur extérieur.

Une application : les votes secrets

Avec un système additivement homomorphe de ce type, on peut opérer des votes secrets sur le réseau sans se rencontrer. Décrivons un tel procédé.

Il y a trois types d'intervenants : les votants (au plus $N - 1$), l'opérateur calculateur, et l'opérateur de dépouillement. L'opérateur de dépouillement choisit un couple clef publique/clef privée et communique la clef publique à chaque votant. Chaque votant choisit alors un nombre entier V de la forme Nk (k entier) s'il veut voter OUI, ou $Nk + 1$ (k entier) s'il veut voter NON ; puis il chiffre le nombre V choisi avec la clef publique et transmet le nombre crypté $\mathscr{C}(V)$ à l'opérateur de calcul. Ce dernier fait la somme S de tous les nombres qu'il reçoit et communique S à l'opérateur de dépouillement, lequel déchiffre le résultat grâce à la clef privée, qu'il est seul à connaître. Il trouve ainsi un nombre de la forme $Mk + p$, où p est le nombre de personnes ayant voté NON. Le résultat du vote est alors communiqué à tous.

L'opérateur calculateur, qui ne connaît pas la clef de déchiffrement, ne peut pas connaître les votes individuels ; l'opérateur de dépouillement, qui n'a pas eu connaissance des nombres transmis individuellement, ne peut pas lui non plus connaître les votes individuels. Les votants peuvent contrôler le travail de l'opérateur de calcul, s'ils acceptent de se communiquer les $\mathscr{C}(V)$, ce qui ne compromet pas le secret du vote. Avec un tel système, même si les votants sont séparés par des milliers de kilomètres et ne communiquent que par messages, aucun vote ne circule de manière lisible, et personne ne sait comment les autres ont voté. On peut encore perfectionner de tels systèmes, comme l'explique un article de Véronique Cortier et Steve Kremer *(voir la bibliographie)*.

Ce type d'applications de la cryptographie partiellement homomorphe est aujourd'hui arrivé à maturité et le système de vote *Helios* est un logiciel libre développé par les chercheurs de l'université Harvard et de l'université catholique de Louvain. On l'a utilisé pour l'élection du recteur de l'université catholique de Louvain et plusieurs fois lors d'élections étudiantes. L'Association internationale des chercheurs en cryptographie (IACR) l'utilise pour choisir les membres de son bureau. Il est fondé sur le système de chiffrement *El Gamal,* qui est additivement homomorphe.

Une arithmétique « secrète »

Voici maintenant la description d'une version du chiffrement pleinement homomorphe de Craig Gentry. Cette version est due à Marten van Dijk, Craig Gentry, Shai Halevi et Vinod Vaikuntanathan. Les idées sont simples et n'impliquent que des calculs élémentaires effectués avec des entiers.

La méthode utilise le fait que, pour créer un système de chiffrement pleinement homomorphe, il suffit de définir un système qui chiffre les bits, des 0 et des 1, et de savoir faire exécuter par l'opérateur extérieur un nombre quelconque d'opérations logiques du type XOR (OU exclusif) et ET entre de tels bits chiffrés.

Vers des machines à voter sûres ?

La question de savoir s'il faut développer l'utilisation de machines à voter est particulièrement délicate, et elle exige que des cryptologues compétents et des techniciens très qualifiés s'en occupent... ce qui n'est pas toujours le cas. Des appels à la prudence ont été émis par de nombreuses personnalités et mouvements. En France, un rapport du Sénat datant de 2014 préconisait le maintien du moratoire sur les machines à voter décidé en 2007. La CNIL a aussi exprimé des réserves sur le vote électronique.

Les incidents et problèmes semblent trop nombreux pour qu'on envisage aujourd'hui de remplacer au niveau national le bulletin de vote, l'isoloir et l'urne fermée à clef par des ordinateurs s'échangeant des messages et que les citoyens devraient manipuler pour effectuer leurs devoirs électoraux.

Le site *Ordinateurs de vote* (www.ordinateurs-de-vote.org/) et le récent rapport *Ethics and electronic voting* de Chantal Enguehard (https://hal.archives-ouvertes.fr/hal-01016256/document) détaillent les raisons et les faits montrant qu'il faut être prudent et patient.

Cependant, rien n'interdit de chercher à concevoir des méthodes de vote aussi rigoureuses que possible, permettant des contrôles *a posteriori* tout en garantissant l'anonymat des votes. Le calcul homomorphe présenté dans le présent article semble une voie sérieuse : il donne des garanties convenables concernant la protection du secret des votes et autorise le contrôle et le recalcul des résultats, tout en permettant à chacun d'être certain que son bulletin est pris en compte.

Le système *Helios* de vote par Internet (https://vote.heliosvoting.org/), fondé sur le calcul homomorphe et développé par l'université Harvard et l'université catholique de Louvain, est disponible gratuitement et librement. L'association internationale des chercheurs en cryptographie (IACR) l'utilise pour choisir les membres de son bureau... ce qui est un argument sérieux pour croire en ses qualités.

Il faut se méfier de toutes les solutions précipitées en la matière, et il ne faut renoncer à aucune des garanties données par les procédures classiques de vote. Il ne fait cependant pas de doute qu'il y aurait des avantages à pouvoir voter de chez soi, et que si c'était possible de manière sûre, le coût de l'organisation d'élections en serait, à terme, considérablement réduit.

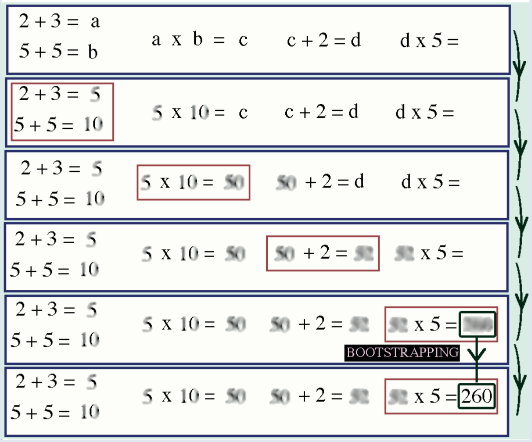
MACHINE À VOTER électronique utilisée à Aulnay-sous-Bois en 2007.

DR

Calcul flou et *bootstrapping*

Le système de calcul homomorphe initialement conçu par Craig Gentry se fonde sur deux idées que l'on retrouve dans tous les systèmes de calcul homomorphe inventés et perfectionnés depuis.

La première idée est que les données doivent être chiffrées d'une manière floue, ce flou ne pouvant être effacé que par celui qui connaît la clef de déchiffrement. À mesure que les calculs homomorphes s'effectuent, le flou s'accroît – comme l'illustration ci-dessous le symbolise. Après un certain nombre d'opérations, le flou devient si important que même avec la clef de déchiffrement, il est devenu impossible d'accéder aux résultats.



La seconde idée est qu'en s'y prenant bien (en demandant au système homomorphe d'exécuter l'opération de déchiffrement sur ses données, donc sans en prendre connaissance, et sans prendre connaissance des résultats), on réussit à effacer le flou qui risquait de devenir irrécupérable. C'est le *bootstrapping*, opération miraculeuse analogue à celle du baron de Münchhausen qui s'élevait dans les airs en tirant sur ses bottes. Soigneusement conçu et réalisé, ce nettoyage périodique des données est réellement possible et permet alors de mener des calculs homomorphes sans limitations de complexité : dès que le flou devient trop grand, on le réduit par *bootstrapping*.

La clef secrète du système, que l'utilisateur doit garder pour lui, comme le *C* de l'exemple du début de l'article, est ici un long nombre entier impair *p*, mettons de 500 chiffres binaires. Dans toute la description de la méthode, la taille des entiers manipulés est importante ; les opérations ne sont pas faisables à la main, mais ce n'est aujourd'hui pas un problème, puisque n'importe quel téléphone peut manipuler des nombres de plusieurs millions de chiffres.

Un multiple « approximatif » de *p* est un nombre de la forme $q\,p+e$, où *e* est un nombre d'une vingtaine de chiffres binaires introduisant une sorte de bruit. Retrouver *p* à partir de la donnée d'un ou plusieurs multiples approximatifs de *p* nécessite des calculs si longs qu'ils sont impossibles à réaliser pratiquement en un temps raisonnable. C'est sur cette impossibilité pratique que se fonde le chiffrement pleinement homomorphe que nous décrivons. Si *q* comporte beaucoup de chiffres (par exemple plusieurs millions de chiffres binaires), un attaquant ignorant *p* ne peut pas faire la différence entre $q\,p+e$ et un nombre quelconque de même taille.

De ce fait, chiffrer un bit *b* ($b = 0$ ou 1) par $\mathscr{C}(b) = pq + b + 2r$ où *r* a une vingtaine de chiffres et *q* plusieurs millions est une bonne méthode : celui qui connaît *p* n'a aucun mal à retrouver *q* en faisant la division par *p*. Le reste de la division est $b + 2r$, qui est pair si $b = 0$ et impair si $b = 1$. Ainsi, celui qui connaît *p* saura sans mal déchiffrer $\mathscr{C}(b)$. Et celui qui ne connaît pas *p* ne voit dans $\mathscr{C}(b)$ qu'un nombre quelconque dont il ne tire rien.

Ce système de chiffrement est homomorphe pour le XOR et le ET et est donc pleinement homomorphe. Le vérifier est une simple question d'arithmétique. En voici les détails, que vous pouvez passer si vous trouvez cela fastidieux.

Soient b_1 et b_2 deux bits. Chiffrons-les : $\mathscr{C}(b_1) = c_1 = q_1p + 2r_1 + b_1$ et $\mathscr{C}(b_2) = c_2 = q_2p + 2r_2 + b_2$ avec des entiers r_1 et r_2 d'une vingtaine de chiffres, *p* d'environ 500 chiffres et q_1 et q_2 de plusieurs millions de chiffres. Celui qui connaît *p* peut calculer le reste de la division par *p* de :
$c_1 + c_2 = (q_1 + q_2)p + 2(r_1 + r_2) + (b_1 + b_2)$.

Ce reste est $2(r_1 + r_2) + (b_1 + b_2)$; puisque $2(r_1 + r_2)$ est pair, connaître le reste donne la parité de $(b_1 + b_2)$ c'est-à-dire le XOR entre b_1 et b_2. Additionner $\mathscr{C}(b_1)$ et $\mathscr{C}(b_2)$ donne donc, si l'on connaît *p*, le XOR entre b_1 et b_2 ($b_1 + b_2$ n'est impair que si l'un ou l'autre est impair, mais pas les deux, et est pair dans les deux autres cas). On peut donc faire calculer à un opérateur extérieur des XOR sans qu'il connaisse ni les données b_1 et b_2, ni le résultat qu'il calcule. C'est la propriété d'homomorphisme additif. Notons que le « bruit » sur $c_1 + c_2$ est passé de r_1 et r_2 à $2(r_1 + r_2)$, il a donc à peu près doublé.

Pour la multiplication, l'opérateur extérieur calcule : $c_1c_2 =$
$(q_1q_2p + 2q_1r_2 + q_1b_2 + 2q_2r_1 + q_2b_1)p$
$+ 2(2r_1r_2 + r_1b_2 + r_2b_1) + b_1b_2$.

La division par *p* donne le reste $2(2r_1r_2 + r_1b_2 + r_2b_1) + b_1b_2$ et donc la parité de b_1b_2, c'est-à-dire le ET entre b_1 et b_2 (b_1b_2 est impair si b_1 et b_2 le sont). C'est la propriété d'homomorphisme multiplicatif. Notons cependant que le bruit sur le résultat est maintenant de l'ordre de $4\,r_1\,r_2$.

Nettoyer périodiquement le « bruit »

Cette méthode est donc parfaite à un détail près. Plus on fera de calculs successifs de XOR et de ET, plus le bruit introduit par les 2*r* augmentera en taille et cela en particulier à cause des ET. Ce n'est pas grave si l'on n'effectue que quelques opérations, mais cela devient très ennuyeux si le bruit atteint la taille de *p*, car alors on ne peut plus retrouver $(b_1 + b_2)$ et b_1b_2.

En effet, si le bruit devient supérieur à p, le reste n'est plus comme il faut (car le quotient augmente de 1 ou plus) pour connaître la somme et le produit des deux bits codés. La méthode n'est bonne qu'à la condition d'opérer un nombre limité d'opérations successives. Pour un calcul dépendant de plusieurs dizaines de bits $b_1, b_2,..., b_k$ présents dans une expression complexe utilisant beaucoup de XOR et de ET, le résultat obtenu par l'opérateur extérieur risque d'être devenu illisible à celui qui lui a confié le calcul. La méthode sans le perfectionnement qui va venir ne fonctionne qu'avec des calculs assez limités. C'est là qu'intervient la deuxième idée majeure de Craig Gentry. Elle consiste à nettoyer périodiquement le bruit pour éviter qu'il ne devienne trop important.

Pour cela, on fait déchiffrer le résultat du calcul partiel avant qu'il ne soit trop brouillé. Ce nettoyage enlève tout le bruit. On l'effectue en demandant à l'opérateur extérieur d'appliquer la fonction de déchiffrement sur le résultat partiel, c'est-à-dire de diviser par p et examiner la parité du reste obtenu. Suprême astuce : puisqu'on ne fait pas confiance à l'opérateur extérieur, on ne lui communique pas la clef de déchiffrement (le nombre p), mais on lui demande de faire ce déchiffrement par un calcul homomorphe utilisant la méthode qu'on vient de décrire ! Ce n'est possible bien sûr que si ce calcul homomorphe du déchiffrement n'est pas trop complexe, c'est-à-dire n'exige pas, une fois traduit en XOR et en ET, une succession trop longue de calculs ; on s'en assurera.

Cela semble assez fou et cela ressemble à l'opération consistant à s'élever dans les airs en tirant sur ses bottes… c'est d'ailleurs pourquoi le nom donné à cette phase du calcul est *bootstrapping*. Pour que la phase de *bootstrapping* soit possible, il faut qu'un minimum d'opérations soient faisables sans que le bruit qui s'introduit soit devenu trop grand, et cela oblige à prendre des nombres entiers très longs pour p et les q_i, mais cela marche, et on dispose donc d'une méthode de calcul pleinement homomorphe comme on en rêvait. L'opération de *bootstrapping*, en nettoyant autant de fois que nécessaire les calculs partiels, rend possible des séries d'opérations aussi longues qu'on le souhaite de XOR et de ET ; et on peut donc faire exécuter tout calcul par un opérateur extérieur qui ignorera ce qu'il calcule.

Puisque la taille des bits chiffrés $\mathcal{C}(b)$ est nécessairement grande, les calculs effectués par l'opérateur extérieur sont très lourds, énormes comparés à ce qu'il faudrait faire si on ne souhaitait rien cacher, mais le but est atteint : on sait déléguer des calculs sans dévoiler ni les données ni le résultat.

Encore quelques ordres de grandeur à gagner

Les travaux très nombreux sur ce sujet menés depuis 2009 ont visé à réduire les surcoûts en taille et en lourdeur de calcul des systèmes homomorphes. On progresse et des réalisations logicielles utilisant les idées mathématiques de ces travaux tentent de rendre les systèmes utilisables en pratique. La série d'opérations nécessaires au chiffrement de données par l'algorithme AES (*Advanced Encryption Standard*), très largement utilisé dans le monde entier, sert de repère pour mesurer les progrès des différents programmes de chiffrement homomorphe. On est ainsi passé, pour l'exécution de ce calcul test par un système homomorphe, de plusieurs heures (sur une machine courante) à quelques secondes.

C'est remarquable, mais pour que réellement on puisse envisager de confier massivement les calculs qu'on veut faire sur des données confidentielles à des opérateurs extérieurs qui les exécuteraient sans rien y voir, il faudra encore attendre un long moment, que le surcoût ait diminué de plusieurs ordres de grandeur.

Certains experts tels que l'Américain Bruce Schneier restent réservés tant les surcoûts sont encore élevés. Aujourd'hui, la formidable idée de la cryptographie homomorphe est utile, par exemple pour la conception de systèmes de vote. Cependant, malgré une série d'idées intéressantes qui ont prouvé qu'elle était possible dans sa forme la plus générale, sa mise en œuvre pour tous, permettant plus de sécurité et une meilleure confidentialité sur les réseaux, devra encore attendre un peu. ■

■ L'AUTEUR

J.-P. DELAHAYE est professeur émérite à l'Université de Lille et chercheur au Centre de recherche en informatique, signal et automatique de Lille (CRISTAL).

■ BIBLIOGRAPHIE

X. Yi *et al.*, **Homomorphic Encryption and Applications**, Springer, 2014.

V. Cortier et S. Kremer, **Vote par Internet**, *Interstices*, 2013 (https://interstices.info/jcms/int_68258/vote-par-internet).

M. Van Dijk *et al.*, **Fully homomorphic encryption over the integers**, *Advances in cryptology - EUROCRYPT 2010*, Springer, pp. 24-43, 2010.

C. Gentry, **Fully homomorphic encryption using ideal lattices**, *Proceedings of the 41st ACM Symposium on Theory of Computing*, pp. 169-178, 2009.

R. Rivest *et al.*, **On data banks and privacy homomorphisms**, *Foundations of Secure Computation*, Academic Press, pp. 169-180, 1978.

Retrouvez la rubrique Logique & calcul sur www.pourlascience.fr