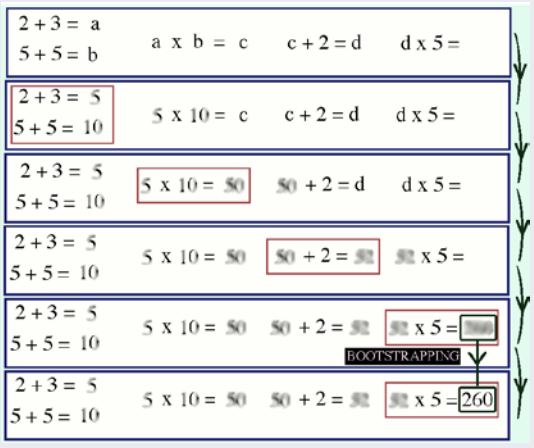
Calcul flou et *bootstrapping*

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 $qp + e$, 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 $qp + e$ et un nombre quelconque de même taille.

De ce fait, chiffrer un bit b (b = 0 ou 1) par $\mathcal{C}(b) = pq + b + 2r$ 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 $\mathcal{C}(b)$. Et celui qui ne connaît pas p ne voit dans $\mathcal{C}(b)$ 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 : $\mathcal{C}(b_1) = c_1 = q_1 p + 2r_1 + b_1$ et $\mathcal{C}(b_2) = c_2 = q_2 p + 2r_2 + b_2$ 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 $\mathcal{C}(b_1)$ et $\mathcal{C}(b_2)$ 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 : $c_1 c_2 = (q_1 q_2 p + 2q_1 r_2 + q_1 b_2 + 2q_2 r_1 + q_2 b_1)p + 2(2r_1 r_2 + r_1 b_2 + r_2 b_1) + b_1 b_2$.

La division par p donne le reste $2(2r_1 r_2 + r_1 b_2 + r_2 b_1) + b_1 b_2$ et donc la parité de $b_1 b_2$, c'est-à-dire le ET entre b_1 et b_2 ($b_1 b_2$ 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 $2r$ 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 $b_1 b_2$.