3 LES TABLES ARC-EN-CIEL

Vous êtes un pirate informatique et vous cherchez le moyen d'exploiter une ou plusieurs données que vous vous êtes procurées. Ces données sont du type [identifiant d'utilisateur, *h*(mot de passe)], où *h* est une fonction de hachage connue et fixée *(voir l'encadré 2)*. Le mot de passe appartient à l'espace obtenu avec 12 lettres minuscules, ce qui correspond à 56 bits d'information et à $26^{12} = 9{,}54 \times 10^{16}$ mots de passe possibles.

Méthode 1. Vous faites défiler tous les mots de passe M possibles. Vous calculez l'empreinte *h*(M) de chacun en regardant si c'est la bonne. Vous n'avez pas besoin d'un important volume de mémoire, car, à chaque nouvel essai, vous effacez les résultats précédents, sauf bien sûr le point de repère qui vous permet de savoir où vous en êtes de votre énumération. En revanche, il vous faut beaucoup de temps pour faire défiler tous les mots de passe possibles. Si votre ordinateur peut mener un milliard d'essais par seconde, il vous faut $26^{12}/(10^9 \times 3\,600 \times 24) = 1\,104$ jours, soit environ 3 ans. Ce n'est pas impossible à envisager et, en utilisant un réseau d'ordinateurs d'un millier de machines, une journée suffira. Cependant, il n'est pas envisageable de reprendre un tel calcul à chaque fois que vous découvrirez une nouvelle donnée de la même catégorie.

Méthode 2. Vous vous dites : « Je vais calculer les empreintes de tous les mots de passe, ce qui me prendra du temps, mais je vais mémoriser ces empreintes dans une grande table. Je n'aurai ensuite qu'à regarder cette table pour savoir quel mot de passe a été utilisé, et cela me servira pour tout nouveau mot de passe de la même catégorie dont je réussirai à connaître l'empreinte. »

Il faudra $(9{,}54 \times 10^{16}) \times (12 + 32)$ octets de mémoire, car il faut 12 octets pour le mot de passe et 32 pour l'empreinte si elle comporte 256 bits comme pour la fonction SHA256. Cela fait $4{,}2 \times 10^{18}$ octets, soit 4,2 millions de disques durs d'une capacité de 1 téraoctet !

La mémoire requise est trop grande. Cette méthode n'est pas plus envisageable que la première. La méthode 1 exige trop de calcul, la méthode 2 trop de mémoire. Ça coince dans les deux cas : le calcul est trop long pour chaque nouveau problème, ou le précalcul avec mémorisation des résultats trop volumineux pour conserver les calculs faits.

Pourrait-on trouver une méthode intermédiaire qui exigerait moins de calculs que la méthode 1 pour chaque nouvelle donnée, mais peut-être un peu plus de mémoire, et qui exigerait moins de mémoire que la méthode 2, mais peut-être un peu plus de calculs ? Oui, c'est possible. L'idée a été proposée par l'Américain Martin Hellman en 1980, puis perfectionnée en 2003 par Philippe Oechslin, de l'École polytechnique fédérale de Lausanne, et plus récemment encore par Gildas Avoine, de l'Insa de Rennes.

Voici comment procéder : on se dote d'abord d'une fonction *C* qui, à partir de l'empreinte *h*(M) du mot de passe M produit un nouveau mot de passe *C*(*h*(M)). Les empreintes sont des nombres écrits en binaire, et les mots de passe des nombres écrits dans une base de numération ayant autant de chiffres qu'il y a de symboles autorisés (disons *K*). Cette fonction *C*, dite de conversion, sera par exemple une conversion de la base 2 à la base *K*. Le précalcul crée des tables de données dénommées *tables arc-en-ciel*, sans doute parce qu'elles sont comme un ensemble de lignes parallèles. Pour une donnée de cette table, on part d'un mot de passe quelconque M_0, on calcule son empreinte $h(M_0)$, puis un nouveau mot de passe possible $C(h(M_0)) = M_1$, et on recommence à partir de M_1, et ainsi de suite. Sans rien mémoriser d'autre que M_0, on calcule ainsi la suite M_1, M_2, ... jusqu'à ce que $h(M_n)$ commence par 20 zéros, ce qui ne se produit qu'une fois sur 1 000 000 environ (car ce que produit une fonction de hachage est assimilable à un tirage aléatoire uniforme et que 2^{20} vaut à peu près 1 000 000). On mémorise alors le couple $[M_0, h(M_n)]$ dans la table.

On calcule un très grand nombre de couples de ce type. Si, lors d'un calcul, on tombe sur le même $h(M_n)$ final, on n'ajoute rien, sauf si le calcul a été plus long (en nombre d'étapes), auquel cas on garde le nouveau couple et on efface l'ancien.

Chaque couple $[M_0, h(M_n)]$ représente la série de mots de passe M_0, M_1, ..., M_k et leurs empreintes, mais sans les mémoriser. Si l'on veut ne laisser aucun vide dans le tableau calculé, le temps de calcul sera plus important que celui nécessaire pour faire dérouler tous les mots de passe de l'espace qu'on explore. C'est envisageable avec un réseau d'ordinateurs, mais on peut ne mener le calcul de la table que partiellement : la base constituée ne résoudra alors qu'une partie des mots de passe dont on connaîtra l'empreinte. Dans le cas d'un calcul complet, l'espace mémoire pour stocker (indirectement) tous les couples calculés est, en ordre de grandeur, un million de fois plus petit que celui de la méthode 2 envisagée plus haut. Il est donc de moins de 4 disques durs de 1 téraoctet. Cette fois, c'est facile.

Voyons maintenant comment cette donnée enregistrée dans les disques permet de retrouver chaque mot de passe de l'espace envisagé en quelques secondes. Nous supposerons pour

$$M_0 \xrightarrow{h} h(M_0) \xrightarrow{C} M_1 \xrightarrow{h} h(M_1) \xrightarrow{C} M_2 \xrightarrow{h} \dots \xrightarrow{C} M_n$$

$$N_0 \xrightarrow{h} h(N_0) \xrightarrow{C} N_1 \xrightarrow{h} h(N_1) \xrightarrow{C} N_2 \xrightarrow{h} \dots \xrightarrow{C} N_p$$

$$\dots\dots\dots\dots\dots\dots$$

$$Q_0 \xrightarrow{h} h(Q_0) \xrightarrow{C} Q_1 \xrightarrow{h} h(Q_1) \xrightarrow{C} Q_2 \xrightarrow{h} \dots \xrightarrow{C} Q_r$$

En connaissant le début de chaque ligne et la fin (ce sont les seules choses qu'on mémorise lors du précalcul), le pirate peut retrouver tout mot de passe dont il connaît l'empreinte : on part de l'empreinte connue qui est quelque part dans le tableau, et on applique *C* et *h* plusieurs fois, ce qui amène en bout de ligne. Cela permet de repérer la ligne où se trouve le mot de passe.
Connaissant la ligne où se trouve le mot de passe, on repart du début de la ligne (qui est connu) et, en quelques applications de *h* et *C*, on retombe sur l'empreinte qu'on connaît, ce qui indique le mot de passe recherché.
Beaucoup de calculs auront été nécessaires pour établir la première et la dernière colonne de la table. Mais ensuite, en ne mémorisant que deux colonnes, et moyennant un calcul (en gros le parcours d'une ligne), on retrouve tout mot de passe à partir de son empreinte.