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 $C(h(\mathrm{M}))$. 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 $h(\mathrm{M}_0)$, puis un nouveau mot de passe possible $C(h(\mathrm{M}_0)) = \mathrm{M}_1$, 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 $h(\mathrm{M}_n)$ 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 $[\mathrm{M}_0, h(\mathrm{M}_n)]$ dans la table.

On calcule un très grand nombre de couples de ce type. Si, lors d'un calcul, on tombe sur le même $h(\mathrm{M}_n)$ 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 $[\mathrm{M}_0, h(\mathrm{M}_n)]$ 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

$$\mathrm{M}_0 \xrightarrow{h} h(\mathrm{M}_0) \xrightarrow{C} \mathrm{M}_1 \xrightarrow{h} h(\mathrm{M}_1) \xrightarrow{C} \mathrm{M}_2 \xrightarrow{h} \ldots \xrightarrow{C} \mathrm{M}_n$$
$$\mathrm{N}_0 \xrightarrow{h} h(\mathrm{N}_0) \xrightarrow{C} \mathrm{N}_1 \xrightarrow{h} h(\mathrm{N}_1) \xrightarrow{C} \mathrm{N}_2 \xrightarrow{h} \ldots \xrightarrow{C} \mathrm{N}_p$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\mathrm{Q}_0 \xrightarrow{h} h(\mathrm{Q}_0) \xrightarrow{C} \mathrm{Q}_1 \xrightarrow{h} h(\mathrm{Q}_1) \xrightarrow{C} \mathrm{Q}_2 \xrightarrow{h} \ldots \xrightarrow{C} \mathrm{Q}_r$$

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.

le raisonnement que dans la phase de précalcul du tableau, on a pris en compte tous les mots de passe de la catégorie visée, par exemple les mots de passe de 12 caractères pris dans les 26 lettres de l'alphabet.

Pour examiner une empreinte e_0, on procède de la façon suivante. On calcule $h(C(e_0)) = e_1$, puis $h(C(e_1)) = e_2$, etc., jusqu'à obtenir une empreinte e_m qui commence par 20 zéros. On regarde alors dans la table à quel M_0 elle est associée. Il y en a nécessairement un, car le mot de passe qu'on étudie est dans l'une des séries qu'on a fait défiler. On part alors de ce M_0 et on calcule $h(M_0)$, $h(C(h(M_0))) = h_1$, etc., jusqu'à tomber sur e_0, ce qui se produit nécessairement. Imaginons que cela se produise pour h_k (c'est-à-dire $h_k = e_0$). Le mot de passe que l'on recherche est alors $C(h_{k-1})$.

Le temps de calcul nécessaire est celui qu'il faut pour rechercher e_m dans la table, auquel il faut ajouter le temps de calcul des h_1, h_2, ..., h_k, qui est environ un million de fois plus court que le temps de calcul ayant été nécessaire pour calculer la table, c'est-à-dire très raisonnable.

Ainsi, avoir fait un précalcul (très long) et en avoir enregistré partiellement les résultats permet de retrouver en un temps raisonnable tout mot de passe dont on connaît l'empreinte.

En résumé, l'idée consiste :
(1) à classer les mots de passe en paquets, dans chacun desquels se retrouvent les mots de passe qui aboutissent à la même empreinte commençant par 20 zéros ;
(2) face à un mot de passe dont on connaît l'empreinte, à retrouver d'abord le paquet auquel il appartient grâce à ce qu'on a mémorisé ;
(3) à déterminer dans ce paquet l'endroit où le mot de passe se trouve en opérant un calcul assez court.

> Une première idée consiste à chiffrer les mots de passe de la liste, c'est-à-dire à utiliser un code secret qui les transforme grâce à une clé de chiffrage en suites de signes apparemment aléatoires pour quiconque ignore la clé de déchiffrage. Cette méthode fonctionne, mais présente deux inconvénients. Il faut déchiffrer le mot de passe associé à un utilisateur à chaque fois qu'on veut le comparer à celui qu'il indique, ce qui n'est pas très commode. Plus grave, pour mener ce déchiffrement nécessaire à la comparaison, il faut garder, dans la mémoire de l'ordinateur du site, la clé de déchiffrement. Or cette clé pourrait être repérée par le pirate, ce qui ramène au problème précédent !

DU HACHAGE… PRÉCÉDÉ PAR UN SALAGE !

La bonne méthode est celle des fonctions de hachage qui produisent des empreintes. Une fonction de hachage est une fonction qui, à tout fichier F donné, associe une « empreinte » (on parle aussi de « condensé », ou de « hash ») $h(F)$, qui est un mot assez court, propre au fichier F, mais produit de telle façon qu'en pratique, il est impossible de retrouver F à partir de $h(F)$. On parle de fonction à sens unique : passer de F à $h(F)$ est facile, passer de $h(F)$ à F est impossible en pratique. De plus, les fonctions de hachage utilisées ont la propriété que même s'il se peut que deux fichiers F et F' aient la même empreinte $h(F)=h(F')$ (on parle de collision), en pratique personne ne sait, pour un F donné, trouver un F' ayant la même empreinte que F.

Avec une telle fonction de hachage, le stockage des mots de passe sur un serveur informatique est facile : au lieu de la liste des couples [identifiant, mot de passe], on ne gardera sur le serveur que la liste des couples [identifiant, h(mot de passe)].

Quand un utilisateur voudra se connecter, le serveur lira son mot de passe *mdp*, en calculera l'empreinte $h(mdp)$ et regardera si c'est bien ce que sa liste de couples mémorisés indique en face de l'identifiant d'utilisateur. Le hacker sera bien ennuyé, car même s'il a pu accéder à cette liste, il ne pourra pas connaître les mots de passe des utilisateurs (impossibilité en pratique de passer de $h(mdp)$ à *mdp*), et sera aussi dans l'impossibilité de produire un autre mot de passe *mdp'* qui aurait la même empreinte et bernerait le serveur (impossibilité pratique des collisions).

Cette prudence recommandée est réellement importante, car même de grands sites prenant la peine de bien se protéger contre les intrusions sont régulièrement piratés. En 2016, le groupe Yahoo! s'était ainsi fait voler les données de un milliard de comptes !

Pour compliquer encore l'exploitation d'une liste volée du type [identifiant, h(mot de passe)], on utilise parfois la méthode du « salage » qui consiste à ajouter au mot de passe une chaîne de caractères aléatoires, différente pour chaque mot de passe. De cette façon, même si deux utilisateurs utilisent le même mot de passe, les empruntes mémorisées seront différentes. Bien sûr, il faudra alors que la liste détenue par le serveur contienne pour chaque utilisateur trois éléments : [identifiant, h(mot de passe+salage), salage]. Quand le serveur veut vérifier le mot de passe qu'un utilisateur vient de donner, il lui ajoute le salage, calcule l'empreinte, et compare avec ce qu'il a dans sa base.

Même quand les mots de passe des utilisateurs sont faibles, cette méthode complique considérablement le travail du hacker. Sans salage, il peut calculer toutes les empreintes d'un dictionnaire et regarder celles qui sont présentes dans les données volées ; tous les mots de passe présents dans son dictionnaire seront repérés. Avec salage, le travail d'exploitation de son dictionnaire devient plus compliqué : le hacker est obligé, pour chaque utilisateur, de calculer les empreintes de tous les mots de passe de son dictionnaire auquel il ajoute le salage propre de l'utilisateur (qu'il connaît puisqu'il a volé la liste). S'il y a 1000 utilisateurs, cela multiplie par 1000 le calcul à faire pour exploiter son dictionnaire.

UN COMPROMIS ENTRE TROP DE CALCUL ET TROP DE MÉMOIRE

Pour bien se défendre contre un attaquant, mieux vaut connaître les méthodes qu'il pourrait mettre en œuvre pour repérer des mots de passe et plus généralement exploiter une table de couples [identifiant d'utilisateur, h(mot de passe)]. On se trouve pris entre deux difficultés : soit il faut beaucoup calculer, c'est-à-dire détenir une grande puissance de calcul, soit il faut disposer de beaucoup de mémoire pour mener un long précalcul, éventuellement pendant des semaines ou des mois, en enregistrant tout ce qu'on calcule et qu'on pourra ensuite exploiter facilement.

Bien souvent, aucune des deux méthodes ne fonctionne. Reste alors l'espoir de concevoir une méthode intermédiaire, qui exige moins de mémoire pour enregistrer le résultat d'un calcul fait à l'avance, et qui, au moment de son exploitation, exige une quantité de calcul raisonnable. L'encadré 3 présente la méthode des tables arc-en-ciel, qui réussit ce compromis.

La science des mots de passe, à l'ère d'Internet, des puissants calculateurs et des réseaux d'ordinateurs, est le théâtre d'une lutte sans merci entre ceux (chacun de nous) qui veulent les utiliser pour se protéger, et ceux qui cherchent à les connaître pour en tirer profit… à nos dépens. ■