L'ART ET LA SCIENCE DES MOTS DE PASSE

Choisir les mots de passe, les garder en les protégeant, réussir à les dévoiler: la science des mots de passe est un trésor de subtilités.

L'AUTEUR

JEAN-PAUL DELAHAYE
professeur émérite à l'université de Lille et chercheur au Centre de recherche en informatique, signal et automatique de Lille (Cristal)

Nous utilisons tous des mots de passe pour protéger nos comptes, notre messagerie électronique, nos accès à des sites particuliers, nos ordinateurs, téléphones, etc. Souvent, quand nous choisissons un mot de passe, le système le refuse en nous indiquant que notre choix est trop simple et cela nous agace. Parmi les conseils donnés, il y a celui de ne pas écrire les mots de passe sur un papier à côté de l'ordinateur. Il faut aussi renoncer à utiliser le même mot de passe sur plusieurs sites internet, en changer régulièrement et... ne pas l'oublier.

Pour bien comprendre tout cela, un peu de bon sens suffit-il? Que nenni! Savez-vous pourquoi 6 caractères ne sont pas assez pour un bon mot de passe et pourquoi il ne faut pas utiliser que des minuscules? Savez-vous qu'il existe des méthodes permettant de stocker indirectement des mots de passe sur un serveur informatique qui, même s'il est visité par un hacker, empêcheront celui-ci de tirer profit de ce qu'il trouve? Savez-vous qu'entre «faire beaucoup de calculs» et «mémoriser beaucoup de données», on peut trouver des compromis et ainsi réussir à casser des mots de passe sinon inaccessibles? Comprendre tout cela sera notre but.

Quand on vous demande un mot de passe, il se situe dans un certain espace de possibilités dont il faut avoir conscience de la taille. Si l'on choisit 6 lettres minuscules pour son mot de passe, par exemple «afzjxd», «tonton», «secret», «wwwwww», l'espace comporte $26^6 = 308\,915\,776$ possibilités. Il y a en effet 26 choix possibles pour la première lettre, 26 choix possibles pour la deuxième, etc. Ces choix sont indépendants, on ne vous impose pas que toutes les lettres soient différentes, et donc la taille de l'espace des mots de passe est le produit des possibilités, ce qui donne $26\times26\times26\times26\times26\times26\times26=26^6$.

DE LA COMBINATOIRE

Si vous choisissez un mot de passe de 12 caractères pris parmi les majuscules, les minuscules, les 10 chiffres et les 10 symboles & é § ? ç ù œ / + =, il y a 72 possibilités pour chacun des 10 signes du mot; l'espace des possibilités a alors pour taille 72^{12}, soit:

$$19\,408\,409\,961\,765\,342\,806\,016 \approx 19\times10^{21}.$$

C'est plus de 62 000 milliards de fois plus que la taille du premier espace. Envisager un à un tous les mots de passe du second espace prendra 62 000 milliards de fois plus de temps. S'il fallait à votre ordinateur une seconde pour visiter le premier espace, il lui faudrait deux millions d'années pour examiner chacun des mots de passe du second espace: la multiplication des possibilités a rendu impraticables certaines opérations envisageables sur le premier espace.

Pour mesurer la taille de ces espaces, il est courant de compter le nombre de chiffres binaires

Jean-Paul Delahaye a récemment publié: **Les Mathématiciens se plient au jeu,** une sélection de ses chroniques parues dans *Pour la Science* (Belin, 2017).

COMBIEN DE TEMPS POUR PARCOURIR L'ENSEMBLE DES MOTS DE PASSE ?

1

Pour qu'un mot de passe soit difficile à trouver, il faut le choisir uniformément dans un grand espace de possibilités. Selon la longueur A de la liste de caractères autorisés pour le composer et selon le nombre N de symboles qui composent le mot de passe, la taille $T = A^N$ de cet espace varie considérablement. Voici cinq exemples.

Dans chacun des exemples, on précise A, N, T et le nombre D d'heures qu'il faut pour explorer l'espace des possibilités, suivi du nombre X d'années qu'il faudrait attendre, en supposant que la loi de Moore (« doublement de capacité de calcul tous les deux ans ») reste toujours valide, pour que cet espace devienne explorable en moins de une heure. On prend comme point de départ l'hypothèse qu'en 2018 votre machine explore un milliard de possibilités par seconde. On a donc :

$T = A^N$, $D = T/(10^9 \times 3\,600)$,
$X = 2 \log_2[T/(10^9 \times 3\,600)]$.

$A = 26$, $N = 6$
$T = 308\,915\,776$
$D = 0{,}000\,085\,8$ heure de calcul
$X = 0$ année d'attente

$A = 26$, $N = 12$
$T = 9{,}5 \times 10^{16}$
$D = 26\,508$ heures de calcul
$X = 29$ années d'attente

$A = 100$, $N = 10$
$T = 10^{20}$
$D = 27\,777\,777$ heures de calcul
$X = 49$ années d'attente

$A = 100$, $N = 15$
$T = 10^{30}$
$D = 2{,}7 \times 10^{17}$ heures de calcul
$X = 115$ années d'attente

$A = 200$, $N = 20$
$T = 1{,}05 \times 10^{46}$
$D = 2{,}7 \times 10^{33}$ heures de calcul
$X = 222$ années d'attente

© Wutzkohphoto/Shutterstock.com

du nombre N de possibilités. Ce nombre est donné par la formule 1+partie entière($\log_2(N)$). Pour le premier exemple, la mesure donne 29 bits, et pour le second 75 bits. On parle aussi d'entropie de 29 bits, ou de 75 bits.

L'Agence nationale de la sécurité des systèmes d'information (Anssi) formule des conseils sur ces questions. Quand il s'agit des mots de passe ou de clés secrètes pour des systèmes de chiffrage dont on veut absolument assurer la sécurité, elle conseille de n'utiliser que des espaces de 100 bits au moins. Il est même recommandé une taille d'au moins 128 bits si l'on souhaite que la sécurité soit assurée pour plusieurs années. L'Anssi précise qu'il faut considérer comme très faibles les espaces de moins de 64 bits, comme faibles ceux entre 64 et 80 bits, comme moyens ceux entre 80 et 100. La différence entre la sécurité instantanée et la sécurité à long terme provient de la loi de Moore, selon laquelle la puissance de calcul d'un ordinateur d'un prix donné double tous les deux ans environ. Même si cette loi de Moore semble voir son rythme >

> diminuer, il est sage de la prendre en compte pour des mots de passe qu'on veut sécuriser durablement.

Pour un mot de passe vraiment robuste au sens de l'Anssi, il faudrait par exemple demander une suite de 16 caractères pris chacun dans un ensemble de 200 caractères. Cela ferait un espace de 123 bits.

Ce n'est guère envisageable, aussi les concepteurs de systèmes sont-ils moins exigeants et se contentent-ils d'espaces de mots de passe faibles ou moyens; ils n'appliquent l'exigence de clés longues que lorsque l'utilisateur n'a pas à la mémoriser lui-même, car elle est gérée automatiquement par le système.

D'autres procédés sont mis en œuvre pour empêcher l'exploration de l'espace des possibilités et casser les mots de passe. Le plus simple est bien connu et utilisé par les cartes bancaires: au bout de trois essais infructueux, la carte est bloquée. D'autres idées ont été proposées comme doubler les temps d'attente entre deux essais à chaque nouvelle erreur, sauf s'il s'est passé un long moment, 24 heures par exemple, auquel cas le système se remet à zéro.

Ces méthodes sont cependant inopérantes quand l'attaquant du système peut travailler hors ligne et mener des essais secrètement de manière isolée, ou que le système ne peut pas être configuré pour arrêter et freiner les essais infructueux.

Assez souvent, un attaquant réussit à se procurer le mot de passe chiffré ou l'empreinte du mot de passe (nous allons voir ce qu'est une empreinte) en accédant à des données du système qu'il attaque. Il peut alors, en toute tranquillité, pendant des jours entiers ou même des semaines, mener toutes les tentatives qu'il souhaite pour casser les mots de passe dont il s'est emparé sous forme chiffrée.

Avant d'expliquer les subtils procédés utilisés par de tels attaquants, revenons sur la taille de l'espace des possibilités.

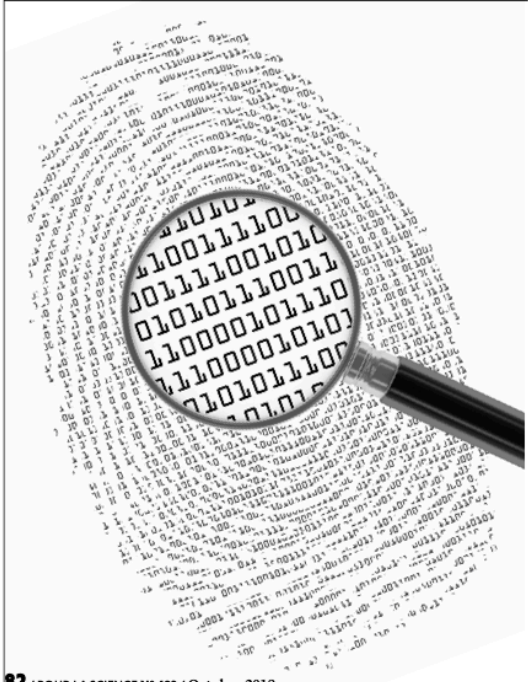
Quand nous avons évalué la taille en bits d'un espace de clés ou mots de passe, dénommée entropie, nous avons implicitement supposé que l'utilisateur choisit au hasard uniformément son mot de passe dans cet espace. Le plus souvent, ce n'est pas le cas, car pour mémoriser un mot de passe on préférera un mot courant, par exemple «locomotive», à un mot vraiment quelconque, par exemple «xdichqewax».

DES DICTIONNAIRES POUR ATTAQUER

C'est là un grave problème qui donne lieu aux attaques par dictionnaires. Des listes de mots de passe couramment utilisés ont été collectées et classées en fonction de la fréquence avec laquelle ces mots de passe sont utilisés. Un attaquant tentera des essais en prenant l'une de ces listes et en l'utilisant dans l'ordre.

2

LES PIRATES DÉÇUS PAR LES FONCTIONS DE HACHAGE



Les fonctions de hachage cryptographique sont des procédés soigneusement mis au point qui transforment un fichier informatique F, aussi long soit-il, en une suite *h*(F) de petite taille appelée empreinte de F. Par exemple, la fonction notée SHA256 transforme la phrase « Il fait beau » en :
316FF650DC5FFC6F8B9CFF25069942F4AA5088
78930A1410D81D5F1DD8C71077
(64 caractères hexadécimaux qui sont équivalents à 256 bits).

Changer un seul caractère du fichier change totalement son empreinte. Par exemple, pour « il fait beau », la fonction SHA256 donne l'empreinte :
D66D08EC93EBA6ACCDD552E2A38EBE6474285
E0D4F2370A87E206553E29AB93C.

Vous pouvez refaire et vérifier ces calculs à l'adresse :
https://passwordsgenerator.net/sha256-hash-generator/
ou https://www.xorbin.com/tools/sha256-hash-calculator.

Les bonnes fonctions de hachage produisent des résultats analogues à ceux qui seraient obtenus s'ils étaient tirés uniformément au hasard. Surtout, pour un résultat possible arbitraire (une suite de 64 caractères hexadécimaux), il est impossible de trouver en un temps raisonnable un fichier F ayant cette empreinte.

Il existe plusieurs générations de fonctions de hachage. Les générations SHA0 et SHA1 sont considérées comme obsolètes et déconseillées. La génération SHA2, à laquelle appartient SHA256, est considérée comme sûre.

Au lieu de mémoriser les mots de passe de leur client, les serveurs internet doivent mémoriser les empreintes de ces mots de passe. En cas d'intrusion, il sera alors impossible ou très difficile au pirate informatique d'exploiter ce qu'il trouve.

Cela fonctionne remarquablement bien car, sans contraintes particulières, nous sommes tous tentés de choisir des mots simples, des noms, des prénoms, des petites phrases, et que les possibilités sont alors relativement peu nombreuses.

Cette utilisation non uniforme des possibilités équivaut à une réduction de l'espace des possibilités, ce qui diminue le nombre moyen d'essais pour casser un mot de passe.

Voici les 25 premiers éléments d'un de ces dictionnaires de mots de passe. Elle est tirée d'une base de 5 millions de mots de passe ayant fuité en 2017 et repérés par SplashData:

1. 123456; 2. Password; 3. 12345678; 4. qwerty; 5. 12345; 6. 123456789; 7. letmein; 8. 1234567; 9. football; 10. iloveyou; 11. admin; 12. welcome; 13. monkey; 14. login; 15. abc123; 16. starwars; 17. 123123; 18. dragon; 19. password; 20. master; 21. hello; 22. freedom; 23. whatever; 24. qazwsx; 25. trustno1.

Remarquez le «password» en seconde position et le sympathique «iloveyou» en position 10. Bien sûr, des telles listes changent selon le pays où elles sont collectées, les sites concernés et au cours du temps.

Pour les mots de passe de 4 chiffres (par exemple le code PIN des cartes SIM des téléphones), les résultats sont encore plus étonnants. En 2013, le site DataGenetics, s'appuyant sur une collecte de 3,4 millions de mots de passe, chacun comportant 4 chiffres, a mesuré que la suite de 4 chiffres la plus utilisée était 1234, qui représentait 11% des choix, suivie par 1111 et 0000 avec des scores de 6% et 2%.

Le mot de passe de 4 chiffres le moins utilisé était 8068. Méfiez-vous quand même, ce n'est peut-être plus vrai maintenant que ces résultats ont été publiés. Le choix 8068 n'était présent que 25 fois parmi les 3,4 millions de suites de 4 chiffres de la base, ce qui est beaucoup moins que les 340 utilisations qu'on aurait trouvées si chaque combinaison de 4 chiffres avait été utilisée avec la même fréquence.

Les 20 premières séries de 4 chiffres sont: 1234; 1111; 0000; 1212; 7777; 1004; 2000; 4444; 2222; 6969; 9999; 3333; 5555; 6666; 1122; 1313; 8888; 4321; 2001; 1010.

Même sans dictionnaire de mots de passe, l'utilisation des différences entre fréquences d'usage des lettres (ou des doubles lettres) dans une langue permet d'organiser efficacement une attaque. Certaines méthodes d'attaque prennent aussi en compte que, pour faciliter la mémorisation, on choisit des mots ayant une certaine structure comme A1=B2=C3, AwX2AwX2, Ooo.Ili. (que j'ai longtemps utilisé), ou alors obtenus en combinant plusieurs mots simples comme password123 ou johnABCoooo. En exploitant de telles régularités, on détermine un ordre d'attaque des mots de passe qui accélère la recherche. La conclusion est simple: dans un espace donné, il faut vraiment choisir au hasard son mot de passe. Certains logiciels le font pour vous et vous proposent donc un mot de passe au hasard. Il faut cependant être conscient qu'un logiciel qui engendre un mot de passe au hasard risque d'utiliser, délibérément ou non, un mauvais générateur pseudoaléatoire, auquel cas ce qu'il propose sera imparfait.

Informés de tout cela, les sites internet convenablement conçus analysent les mots de passe proposés au moment de leur création et refusent ceux qu'il serait trop facile de retrouver. C'est agaçant, mais c'est pour votre bien!

TESTER LES MOTS DE PASSE...

Un outil permet de vérifier si l'un de vos mots de passe n'a pas déjà été piraté. Il utilise une base de plus de 500 millions de mots de passe obtenue en regroupant des listes dévoilées à la suite d'attaques diverses:

https://haveibeenpwned.com/Passwords

J'ai essayé e=mc2e=mc2 que j'aimais bien et pensais sûr, et j'ai obtenu la réponse déconcertante: «This password has been seen 110 times before.» Quelques autres essais montrent qu'il est difficile de proposer un mot de passe facile à mémoriser et que la base ne connaît pas. Par exemple, aaaaaa est apparu 353071 fois; a1b2c3d4, 105134 fois; abcdcba, 353 fois; abczyx: 158 fois; acegi, 111 fois; chirac, 600 fois; sarkozy, 680 fois; hollande, 832 fois; macron, 279 fois.

Notons qu'être original reste possible. Voici cinq exemples de mots de passe que le site n'a pas dans sa liste: eyahaled (mon nom à l'envers); bizzzzard; meaudepace; modeuxpass; abcdef2018. Maintenant que je les ai essayés, je me demande si la base ne pourrait pas les ajouter lors de sa prochaine mise à jour et je ne les utiliserai donc pas!

Le NIST (National Institute of Standards and Technology), aux États-Unis, a récemment publié un avis où il recommande la méthode des dictionnaires pour filtrer le mot de passe choisi par un utilisateur qui est en train d'en créer.

... ET SAVOIR LES CONSERVER

Parmi les règles qu'un bon concepteur de serveur internet doit absolument appliquer, il y a celle-ci: ne pas garder dans les mémoires de l'ordinateur qui fait fonctionner le site internet la liste en clair des couples [identifiant d'utilisateur, mot de passe].

La raison est évidente: un pirate pourrait accéder à l'ordinateur contenant cette liste, soit parce que le site est mal protégé, soit parce qu'une faille profonde dans le système, ou dans la puce au cœur de la machine, aura été exploitée, alors qu'elle est inconnue de tous (faille Zero-day)... sauf de l'attaquant. >

BIBLIOGRAPHIE

Wikipedia, Mot de passe et Rainbow table, entrées consultées en juin 2018.

G. Avoine *et al.*, **How to handle rainbow tables with external memory**, dans J. Pieprzyk et S. Suriadi (éd.), *Information Security and Privacy - ACISP 2017*, pp. 306-323, Springer, Cham, 2017.

P. A. Grassi *et al.*, **Digital Identity Guidelines**, NIST Special Publication 800-63-3, 2017 (https://doi.org/10.6028/NIST.SP.800-63-3).

G. Avoine *et al.*, **Characterization and improvement of time-memory trade-off based on perfect tables**, *ACM Transactions on Information and System Security*, vol. 11(4), article 17, 2008.

P. Oechslin, **Making a faster cryptanalytic time-memory trade-off**, dans D. Boneh (éd.), *Advances in Cryptology - CRYPTO 2003*, pp. 617-630, Springer, 2003.

M. E. Hellman, **A cryptanalytic time-memory trade-off**, *IEEE Transactions on Information Theory*, vol. IT-26, n° 4, pp. 401-406, 1980.

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

$$M_0 \xrightarrow{h} h(M_0) \xrightarrow{C} M_1 \xrightarrow{h} h(M_1) \xrightarrow{C} M_2 \xrightarrow{h} \dots \xrightarrow{C} M_n$$
$$N_0 \xrightarrow{h} h(N_0) \xrightarrow{C} N_1 \xrightarrow{h} h(N_1) \xrightarrow{C} N_2 \xrightarrow{h} \dots \xrightarrow{C} N_p$$
....
$$Q_0 \xrightarrow{h} h(Q_0) \xrightarrow{C} Q_1 \xrightarrow{h} h(Q_1) \xrightarrow{C} Q_2 \xrightarrow{h} \dots \xrightarrow{C} Q_r$$

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.

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. ■