Qui sait bien hacher les mots de passe ?

A. Uteur[1] et A. Nonyme[2]

[1] *Institut secret*
[2] *Need-to-know-basis institute*

Malgré des décennies d'analyses et d'avertissements au sein de la communauté académique, les grands fournisseurs de service n'appliquent correctement le hachage des mots de passe que très rarement, comme en témoignent les fuites régulières de leurs bases de données. La norme actuelle basée sur l'envoi en clair d'un mot de passe qui sera haché sur le serveur correspond d'ailleurs à des contraintes aujourd'hui obsolètes. Nous analysons les avantages et inconvénients de l'alternative que représente le hachage côté client, et les moyens de détecter automatiquement la méthode utilisée. Nous montrons aussi que, sur les 50 services les plus utilisés aujourd'hui, seuls huit utilisent cette architecture, et que ce sont exactement les huit services basés en République Populaire de Chine.

Mots-clefs : Hachage, Authentification, Mots de passe, Standards du Web

1 L'état du hachage aujourd'hui

Même si le hachage des mots de passe avait déjà été suggéré en 1967 [MT79], il reste aujourd'hui rarement utilisé correctement. On retrouve ainsi des mots de passe en clair ou hachés avec des méthodes obsolètes comme le MD5. On ne peut connaître exactement l'ampleur du phénomène, car nos informations proviennent entièrement régulières des fuites de bases de données [KM16]. On peut cependant noter plusieurs graves problèmes dans presque tous les secteurs (l'exception principale étant chez les gestionnaires de mots de passe) :

— Dans une étude de 2016, Jaeger et ses coauteurs analysèrent 31 failles majeures des 8 années précédentes [JPG+16]. Pour donner une idée de l'ampleur du phénomène, mate1.com avait une base de plus de 27 millions de mots de passe en clair, et badoo.com une base de 122 millions de mots de passe hachés en MD5.

— Les développeurs de la seule base de donnée hachée correctement — celle d'Ashley Madison, qui utilisait bcrypt — conservaient sur les mêmes serveurs une copie chiffrée en MD5 au cas où [JPG+16].

— Facebook a révélé en 2019 avoir conservé des logs de mots de passes non hachés (car les logs ne rentraient pas dans la base de donnée standard), avec des centaines de millions de mots de passe accessibles à plus de 2000 développeurs pendant de nombreuses années.

— Vu que les coûts financiers et de réputation proviennent entièrement de ces fuites, la motivation pour améliorer le système après une fuite est faible, ce qui mène des géants comme Yahoo à avoir plusieurs fuites entre 2016 et 2017 qui montrent qu'ils n'ont pas amélioré leur sécurité [Sha18].

Avant d'explorer le hachage côté client, rappelons rapidement les pratiques recommandées aujourd'hui :

— Ne pas hacher les mots de passe ni avec MD5 ou SHA-1 — qui sont obsolète — ni avec les évolutions plus récentes d'algorithme de hachage généraliste. A la place, utiliser de l'étirement de clé comme bcrypt ou PBKDF2, ou idéalement utiliser les algorithmes optimisés contre le calcul en parallèle comme Argon2.

— Saler chaque mot de passe pour éviter les tables arc-en-ciel et la parallélisation des calculs (mais le sel peut sans risque être public, voire simplement le login de l'utilisateur).

— Ne pas conserver — par exemple dans des logs — une version non hachée du mot de passe pour éviter les attaques par canal auxiliaire.

Comme alternative potentielle à tout cela, on peut aussi utiliser un système de type PAKE comme OPAQUE [JKX18], mais ces derniers ont toujours une adoption faible et des problèmes de brevets.

2 Détection du hachage côté client

L'un des principaux intérêts du hachage côté client est qu'il est observable par l'utilisateur. Cependant, sa détection peut être mis en difficulté à cause de l'obfuscation fréquente du code source des applications web. Il existe au moins trois méthodes permettant de vérifier l'utilisation de hachage coté client : les analyses syntactique et sémantique, et la charge de calcul d'une page web. Les deux premières consistent à à vérifier si les librairies utilisées par la page web contiennent des — bonnes — fonctions de hachage. On peut étendre cela en analysant l'objet mémoire correspondant au champ d'entrée du mot de passe, et en vérifiant si une routine de hachage y est bien associée. Vu que ces méthodes sont difficiles à automatiser, une autre méthode consiste à observer la charge de calcul utilisé par la page web. Cette analyse permet de vérifier la bonne présence d'un pic d'utilisation de ressources processeur ainsi que de mémoire lors de la connexion, qui confirme l'utilisation d'une routine de hachage contemporaine. La présence de pics pouvant être lié à d'autres activités, cette méthode permet surtout de détecter la non-utilisation de hachage correct.

Nous avons décidé d'utiliser une analyse sémantique manuelle pour vérifier quels sites implémentent le hachage côté client parmi les 50 sites web mondiaux les plus consultés, dont les résultats indiqués dans la Table 1. La Figure 2 montre un exemple de mot de passe en clair envoyé au serveur (en utilisant TLS, mais sans hachage) sur facebook.com et l'équivalent sur baidu.com.

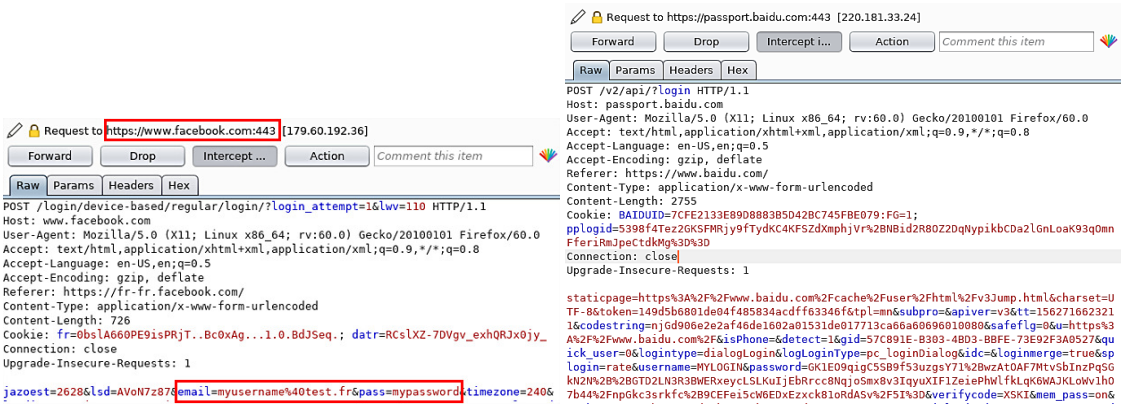

FIGURE 1: Requêtes envoyées à facebook.com (gauche) et baidu.com (droite) par TLS. Le mot de passe en clair est encadré en rouge pour facebook, et le mot de passe chiffré est sur la troisième ligne en partant du bas pour baidu.

google.com	amazon.com	vk.com	netflix.com	**csdn.net**	aliexpress.com	amazon.co.jp
youtube.com	**taobao.com**	sohu.com	linkedin.com	yahoo.co.jp	livejasmin.com	google.co.in
facebook.com	twitter.com	jd.com	bilibili.com	mail.ru	microsoftonline.com	github.com
baidu.com	tmall.com	yandex.ru	twitch.tv	bing.com	**alipay.com**	okezone.com
wikipedia.org	reddit.com	**sina.com.cn**	pornhub.com	microsoft.com	ebay.com	imdb.com
qq.com	instagram.com	**weibo.com**	login.tmall.com	whatsapp.com	xvideos.com	google.com.hk
yahoo.com	live.com	blogspot.com	**360.cn**	naver.com	tribunnews.com	pages.tmall.com

TABLE 1: Résultat d'une analyse manuelle des sites du Alexa Top 50 global websites le 07-07-2019. Les sites en gras sont ceux où le mot de passe n'était pas simplement chiffré via TLS.

Cette analyse manuelle indique un résultat surprenant en apparence : les huit sites n'envoyant pas le mot de passe en clair sont exactement les huit sites basés en RPC. En réalité, plusieurs sites (dont 360.cn et qq.com) réutilisent directement le système de baidu, et d'autres s'en inspirent. Nous savons cependant que la chine impose des règles strictes aux entreprises sur son territoire en matière de cryptographie [MH19] (nous n'avons d'ailleurs pas réussi à recalculer les hachés observés sur baidu avec les protocoles standards de hachage en occident). Les systèmes chinois découragent aussi l'authentification classique nom d'utilisateur/mot de passe, favorisant l'utilisation du double facteur, notamment via le téléphone. On ne peut que spéculer sur les raisons politiques de son implémentation, mais ce choix d'architecture en RPC a plusieurs avantages. Il peut tout d'abord protéger ses utilisateurs, notamment si les protocoles occidentaux se

retrouvent vulnérables, notamment si ces vulnérabilités sont intentionnelles [Hal13, TCCA16], mais il peut aussi aider le gouvernement à traquer ses citoyens en ligne [Sta02].

Dans le reste du monde, la question plus appropriée ne porterait par sur l'absence de hachage côté client mais sur la présence de hachage tout court. On sait que blâmer les développeurs sans pousser les entreprises à investir dans ces ressources ne fonctionne pas [FHvO14]. De plus, même si le milieu académique n'est que partiellement écouté, les travaux sur la question du hachage côté client ou côté serveur y demeurent presque absents, avec de rares exceptions [MKV+13]. Ainsi, même dans les compétitions d'algorithmes de hachage, la question de les exécuter côté serveur ou côté client n'est pas à priori posée [HPM15].

3 Avantages et inconvénients du hachage côté client

Le hachage côté client a six avantages principaux :

— **Pas de réutilisation des données fuitées** : les mots de passe étant hachés, accéder à une base de donnée de mots de passe ne permet pas de monter une attaque contre une autre base de donnée.
— **Réduction des coûts serveurs** : le hachage — qui nécessite de la puissance de calcul pour être de bonne qualité — a lieu sur le client, ce qui décharge les serveurs.
— **Meilleur hachage** : il n'y a plus un choix à faire entre coût serveur et qualité du hachage, ce qui permet d'utiliser la sécurité la plus élevée atteignable par la machine cliente. Cela enlève aussi la possibilité de faire des *attaques de bicyclettes* [HMB+20].
— **Difficulté accrue du phishing** : si la méthode est généralisée avec des protocoles normalisés, elle peut former un obstacle de plus au phishing, notamment par homoglyphe [HWG06].
— **Simplicité** : si la méthode se généralise côté client, elle peut se normaliser et se stabiliser en étant indépendante des choix de back-end.
— **Responsabilité publique** : l'avantage le plus important est que cette méthode rend visible le hachage utilisé, et que sa généralisation créerait donc un coût social pour les entreprises ayant des méthodes obsolètes. Cela transformerait le système de réactif en proactif.

Il a cependant aussi quatre inconvénients :

— **Authentification prolongée après une fuite** : si un adversaire obtient une copie de la base de donnée, il peut continuer à s'authentifier durablement. Cette méthode est prévenue par un simple round de hachage (même avec quelques rounds de MD5) côté serveur.
— **Puissance de calcul limitée** : cet inconvénient est mineur pour l'instant, mais la puissance de calcul limitée peut se retrouver plus critique dans un contexte où de nombreux objets connectés doivent s'authentifier par ces méthodes.
— **Blocage des scripts** : le hachage côté client nécessite l'exécution de scripts sur sa machine, ce qui peut déranger certains utilisateurs, ou déclencher des alertes au cryptojacking [ELMC18].
— **Incompatibilité avec d'anciens protocoles** : certains anciens protocoles avaient besoin de recevoir des mots de passe en clair, mais ils étaient déjà considérés presque obsolètes en 2012 [MKV+13] et ne devraient donc pas poser de souci.

4 Changer le hachage en pratique

Faire changer les méthodes de hachage utilisées en pratique demanderait beaucoup d'efforts coordonnées sur plusieurs fronts en parallèle. En effet, le gain réel entre le hachage côté client et le hachage côté serveur est faible par rapport au gain du passage du stockage en clair au hachage simple, passage qui n'est toujours pas entièrement terminé.

Du point de vue d'un simple utilisateur, il n'y a hélas pas grand chose à faire †. Bien qu'il serait relativement facile de créer un plugin détectant l'absence de hachage côté client, faire cela comme personne isolée aurait un large coût — avec un avertissement sur presque tous les sites — sans avoir d'influence sur les fournisseurs de service si le plugin n'est pas adapté à grande échelle.

†. Un utilisateur avancé pourrait développer son propre système hachant automatiquement de manière déterministe le mot de passe lors de son entrée, ce qui serait presque assimilé à un gestionnaire de mot de passe avec quelques particularités.

C'est justement pour cette raison qu'une piste intéressante serait de regarder les structures de motivation (*incentives*), et de s'adresser directement aux développeurs de navigateurs. On sait par expérience que les avertissements implémentés directement dans le navigateur peuvent avoir un impact plutôt rapide, comme avec le passage du HTTP au HTTPS [KB15, FBK+17], même si l'indication par le cadenas vert a aussi introduit de nouvelles attaques, liées à une confiance infondée qui faisait baisser la méfiance des utilisateurs [Cim17]. Un avantage supplémentaire de cette méthode est que convaincre un navigateur serait probablement suffisant pour que d'autres navigateurs imitent l'idée (pour ne pas paraître en retard sur la sécurité).

Nous terminons ainsi avec quelques questions :

— Pourquoi les sites de RPC sont-ils les seuls à hacher leurs mots de passe, et quelles méthodes utilisent-ils ?

— Existe-t-il des méthodes plus efficaces et sûres pour détecter le hachage ou son absence ?

— Comment pouvons-nous convaincre la communauté d'avoir cette discussion, et idéalement d'opérer ce changement majeur ?

Références

[Cim17] Catalin Cimpanu. Extended validation (ev) certificates abused to create insanely believable phishing sites, 2017. `https://www.bleepingcomputer.com/news/security/extended-validation-ev-certificates-abused-to-create-insanely-believable-phishing-sites/`.

[ELMC18] Shayan Eskandari, Andreas Leoutsarakos, Troy Mursch, and Jeremy Clark. A first look at browser-based cryptojacking. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 58–66. IEEE, 2018.

[FBK+17] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring {HTTPS} adoption on the web. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1323–1338, 2017.

[FHvO14] Dinei Florêncio, Cormac Herley, and Paul C. van Oorschot. An administrator's guide to internet password research. In *LISA*, volume 14, pages 35–52, 2014.

[Hal13] Thomas C. Hales. The NSA back door to NIST. *Notices of the AMS*, 61(2) :190–19, 2013.

[HMB+20] Benjamin Harsha, Robert Morton, Jeremiah Blocki, John Springer, and Melissa Dark. Bicycle attacks considered harmful : Quantifying the damage of widespread password length leakage, 2020. https ://arxiv.org/abs/2002.01513.

[HPM15] George Hatzivasilis, Ioannis Papaefstathiou, and Charalampos Manifavas. Password hashing competition-survey and benchmark. *IACR Cryptology ePrint Archive*, 2015.

[HWG06] Tobias Holgers, David E Watson, and Steven D Gribble. Cutting through the confusion : A measurement study of homograph attacks. In *USENIX Annual Technical Conference, General Track*, pages 261–266, 2006.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. Opaque : an asymmetric pake protocol secure against pre-computation attacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 456–486. Springer, 2018.

[JPG+16] David Jaeger, Chris Pelchen, Hendrik Graupner, Feng Cheng, and Christoph Meinel. Analysis of publicly leaked credentials and the long story of password (re-) use. In *Proc. Int. Conf. Passwords*, 2016.

[KB15] Michael Kranch and Joseph Bonneau. Upgrading https in mid-air. In *Proceedings of thz 2015 Network and Distributed System Security Symposium*. NDSS, 2015.

[KM16] Maria Karyda and Lilian Mitrou. Data breach notification : Issues and challenges for security management. In *Mediterranean Conference on Information Systems*, 2016.

[MH19] Louise Bergman MartinKauppi and Qiuping He. Performance evaluation and comparison of standard cryptographic algorithms and chinese cryptographic algorithms. Master's thesis, 2019.

[MKV+13] Michelle L. Mazurek, Saranga Komanduri, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Richard Shay, and Blase Ur. Measuring password guessability for an entire university. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*, CCS '13, pages 173–186, New York, NY, USA, 2013. ACM.

[MT79] Robert Morris and Ken Thompson. Password security : A case history. *Communications of the ACM*, 22(11) :594–597, November 1979.

[Sha18] Shape. 2018 credential spill report. Technical report, Shape Security, 2018.

[Sta02] State Council of the People's Republic of China. Regulations on administration of business premises for internet access services, article 23, 2002.

[TCCA16] Theo Tryfonas, Michael Carter, Tom Crick, and Panagiotis Andriotis. Mass surveillance in cyberspace and the lost art of keeping a secret. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*, pages 174–185. Springer, 2016.