RFC 7914 : The scrypt Password-Based Key Derivation Function

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 29 août 2016

Date de publication du RFC : Août 2016

https://www.bortzmeyer.org/7914.html

Ce RFC normalise la fonction de dérivation de clé scrypt.

À quoi ça sert, une fonction de dérivation de clé ? Comme leur nom l'indique, elles permettent d'obtenir des clés cryptographiques (ou autre matériel cryptographique) à partir des données qu'on leur fournit. Une utilisation courante est de fabriquer une clé pour un algorithme de chiffrement, à partir d'une phrase de passe. Cela permet d'obtenir une clé (longueur fixe, format donné) pour les opérations cryptographiques tout en laissant l'utilisateur manipuler uniquement des textes mémorisables. Par exemple, pour chiffrer un disque dur, l'utilisateur va indiquer une phrase de passe, mais le disque sera chiffré à partir de la clé obtenue en appliquant la fonction de dérivation de clé (KDF, pour "*Key Derivation Function*") à cette phrase. Une autre utilisation est pour transformer un mot de passe qu'on doit stocker dans un fichier en une information inutilisable pour un attaquant qui mettrait la main dessus. Pour se connecter, on tape le mot de passe, on refait tourner la KDF et on vérifie qu'on obtient bien le résultat stocké.

Problème de cette méthode, l'ennemi peut tenter de faire lui-même la dérivation : il essaie des tas de mots de passe et regarde s'il obtient le résultat stocké. C'est pour cela qu'une des qualités d'une bonne fonction de dérivation est, paradoxalement, d'être **lente** : les gens qui connaissent le mot de passe ne seront pas gênés (ils ne font qu'un seul essai) alors que l'attaquant par force brute qui essaie des milliards de mots sera ralenti. Idéalement, on voudrait une fonction qui ne puisse pas facilement être mise en œuvre dans des ASIC, pour qu'un attaquant riche ne puisse pas investir dans une « machine à deviner les mots de passe ». C'est l'un des avantages de scrypt. (Les fanas de chaîne de blocs noteront que des chaînes comme Litecoin utilisent scrypt justement pour cette raison : rendre le minage plus accessible à tous en contrariant les ASIC.)

Pourquoi encore ajouter des fonctions de dérivation de clés (section 1 du RFC) ? Il y en a eu plein dans l'histoire, de la vénérable crypt à PBKDF2 (RFC 2898 [1]) puis aux récents bcrypt et Argon2. On en imagine souvent de nouvelles, par exemple celle-ci qui n'utilise pas du tout de chiffrement, juste de la condensation <https://www.akkadia.org/drepper/SHA-crypt.txt>. Pour résister aux attaques par force brute (que la loi de Moore rend plus efficace chaque année), certaines ont un nombre d'itérations variables. On applique plusieurs fois l'opération de dérivation, si les machines deviennent plus rapides, on augmente ce nombre d'itérations. Cela ne marche bien que si l'attaquant utilise les mêmes logiciels que les utilisateurs normaux. Mais si la fonction de dérivation est facilement programmable dans des circuits matériels spécialisés, l'attaquant pas trop pauvre pourra s'acheter une ferme de cassage de mots de passe, remplie de circuits conçus spécifiquement, et travaillant en parallèle (les circuits deviennent plus rapides mais aussi plus petits : on peut en entasser davantage). Il ne joue alors plus dans la même catégorie que les utilisateurs légitimes.

C'est là que scrypt intervient : l'algorithme a été délibérement conçu pour être difficile à mettre dans un ASIC. scrypt a été publié en 2009 (voir l'article original <http://www.tarsnap.com/scrypt/scrypt.pdf> qui fut présenté à USENIX). Ce RFC a commencé en 2013 et a eu une longue gestation. Il ne décrit pas scrypt, renvoyant au papier original, mais se contente de préciser les points qui sont nécessaires pour des mises en œuvre interopérables.

La section 2 du RFC décrit les paramètres de la fonction. Le plus évident est la phrase de passe, souvent choisie par un humain. Il y a aussi un sel, en général choisi aléatoirement (RFC 4086), et divers paramètres techniques, permettant notamment d'ajuster l'algorithme aux caractéristiques des machines dont on dispose. Une taille de bloc de 8 et un facteur de parallélisation de 1 conviennent bien à l'heure actuelle, mais vont sans doute augmenter dans le futur.

scrypt dépend de la fonction de condensation Salsa20 Core, plus exactement de sa version simplifiée Salsa20/8 Core (section 3 du RFC). Une mise en œuvre en C est incluse dans le RFC. (Voir la description originelle <http://cr.yp.to/salsa20.html> et la spécification de Salsa20 <http://cr.yp.to/snuffle/spec.pdf>.)

scrypt est un « chef d'orchestre », qui dépend de plusieurs autres algorithmes comme BlockMix (section 4), ROMix (section 5), le PBKDF2 du RFC 2898 et le HMAC-SHA-256 du RFC 6234. L'algorithme de scrypt, qui fait fonctionner ensemble tout cela, figure en section 6.

Si vous êtes programmeur et que vous mettez en œuvre scrypt, les sections 8 à 13 du RFC contiennent des vecteurs de test pour les différents algorithmes utilisés. Par exemple, avec la phrase de passe "*pleaseletmein*", le sel `SodiumChloride` (exemple contestable, ce sel n'a pas été généré aléatoirement), le facteur CPU/mémoire à 16384, la taille de bloc 8 et le facteur de parallélisation 1, la clé dérivée par scrypt sera `70 23 bd cb 3a fd 73 48 46 1c 06 cd 81 fd 38 eb fd a8 fb ba 90 4f 8e 3e a9 b5 43 f6 54 5d a1 f2 d5 43 29 55 61 3f 0f cf 62 d4 97 05 24 2a 9a f9 e6 1e 85 dc 0d 65 1e 40 df cf 01 7b 45 57 58 87`. Si vous trouvez une autre valeur, vérifiez votre programme.

Lisez aussi la section 14, consacrée aux questions de sécurité. Par exemple, scrypt peut consommer beaucoup de mémoire (c'est fait exprès, cela fait partie des techniques qui rendent difficile sa mise en œuvre en ASIC) et il y a donc un risque de déni de service si on accepte d'exécuter scrypt avec des paramètres quelconques, fournis depuis l'extérieur.

scrypt est, entre autres, présents dans OpenSSL depuis le 1.1.0, officiellement publiée juste après le RFC.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc2898.txt`