RFC 10033 : Hash-based Signatures: State and Backup Management

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 15 septembre 2026

Date de publication du RFC : Septembre 2026

https://www.bortzmeyer.org/10033.html

Certains algorithmes de cryptographie sont **à état**, c'est-à-dire que les programmes qui les utilisent doivent se souvenir des exécutions précédentes de l'algorithme ; refaire tourner l'algorithme dans les mêmes conditions serait une faille de sécurité. Cette nécessité de mémoriser un état complique évidemment leur utilisation. Ce nouveau RFC documente les bonnes pratiques à ce sujet.

La question des algorithmes à état est particulièrement cruciale dans le contexte de la cryptographie post-quantique. Un problème de beaucoup d'algorithmes post-quantiques est la grande taille de leurs clés et de leurs signatures, ce qui pose problème pour certaines applications sur l'Internet. Une solution possible serait d'utiliser des algorithmes comme ceux fondés sur la condensation, qui ont des clés et des signatures de taille plus raisonnable, et reposent sur des algorithmes de condensation bien connus. Mais ces algorithmes sont à état, ce qui est vraiment pénible à gérer.

Il existe plusieurs algorithmes dans cette famille fondée sur la condensation : LMS ("*Leighton[Caractère Unicode non montré [1]]Micali Signatures*"), HSS ("*Hierarchical Signature System*"), XMSS ("*eXtended Merkle Signature Scheme*"), etc. Tous résistent aux futurs calculateurs quantiques mais tous ont un défaut : signer deux fois avec la même clé permet à la cryptanalyse de découvrir la clé privée. Il faut donc dériver une nouvelle clé à chaque signature et se souvenir des clés précédentes, pour ne pas risquer de réutilisation. C'est évidemment très contraignant et cela explique pourquoi, par exemple, ces algorithmes n'ont pas été sérieusement envisagés pour DNSSEC.

Si vous voulez vous instruire sur ces algorithmes utilisant la condensation, lisez les RFC 8391 [2] (sur XMSS), RFC 8554 (sur LMS) et la norme NIST SP 800-208 <https://nvlpubs.nist.gov/nistpubs/

1. Car trop difficile à faire afficher par LaTeX
2. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8391.txt

SpecialPublications/NIST.SP.800-208.pdf>. Dans ces algorithmes, la clé privée est en fait une série de clés, partant d'une graine et créées par dérivations successives. L'état est simplement un index vers cette liste, indiquant la dernière clé utilisée. Si on réutilise une clé, on dévoile la graine. Il est donc très important de garder trace de l'index, de l'état. À chaque signature effectuée, il faut le mettre à jour. Imaginez par exemple un dispositif de signature qui signerait, diffuserait le message signé mais subirait ensuite une coupure de courant qui l'empêcherait de mettre à jour l'état. Une fois le courant revenu, la signature suivante utilisera le même index, donc la même clé et paf, le ou la cryptanalyste qui lira les deux signatures et connait la clé publique pourra trouver la graine, donc la clé privée. (Si ça vous semble magique, lisez « "*State Management for Hash- Based Signatures*" <https://eprint.iacr.org/2016/357.pdf> », « "*State management for stateful authentication mechanisms*" <https://www.etsi.org/deliver/etsi_tr/103600_103699/103692/01.01.01_60/tr_103692v010101p.pdf> », « "*Oops, I did it again [Caractère Unicode non montré] Security of One-Time Signatures under Two-Message Attacks*" <https://eprint.iacr.org/2016/1042.pdf> » et « "*Oops, I did it again revisited : another look at reusing one-time signatures*" <https://eprint.iacr.org/2023/1905> ». Notre RFC est donc consacré à décrire les mesures qui peuvent empêcher cette situation. Il faut s'assurer que l'état est bien mis à jour à chaque signature, que, dans le cas où il y a plusieurs signeurs, qu'ils ne puissent jamais utiliser la même clé et que si on restaure les sauvegardes, on ne se trouve pas à réutiliser une clé. C'est compliqué ? Oui, et c'est pour cela que les algorithmes à état sont peu populaires. (Ils avaient été envisagés pour DNSSEC mais avaient fait peur à tout le monde.) Comme le note la section 1.1 du RFC, ces algorithmes ne conviennent pas pour un usage généraliste, il faut les réserver à des cas bien précis.

On peut même se demander pourquoi ces algorithmes n'ont pas été jetés à la poubelle tout de suite. C'est parce qu'ils ont quand même certains avantages, comme des tailles de clés et de signatures raisonnables. (DNSSEC, cité plus haut, ne peut pas envisager les énormes clés et signatures de ML-DSA.) Un exemple d'utilisation des systèmes à état est donné dans le RFC 9802.

La section 2 du RFC rappelle la terminologie utilisée. La **clé privée** à proprement parler est la graine dont dérivent les clés de signature. Elle-même est sans état et a potentiellement une longue durée de vie. Elle se gère comme n'importe quelle clé privée (donc, vous ne la mettez pas sur GitHub). L'**état** est au contraire à courte durée de vie et est mis à jour à **chaque** signature. Sa gestion (le mettre à jour, le sauvegarder, le distribuer…) est toute la difficulté des algorithmes de cryptographie à état.

Place à la pratique en section 3. Par exemple, les sauvegardes. Sauvegarder une clé qui peut servir pendant 10 ou 20 ans est une chose. Mais, ici, il faut sauvegarder du matériel cryptographique qui change tout le temps, puisque sauvegarder l'état est indispensable. La section 3.2 est une bonne lecture si vous vous intéressez aux questions de préservation de contenu numérique.

Mais il y a aussi le problème des procédures à suivre. Gérer des clés avec état est plus complexe et nécessite du personnel qualifié et consciencieux. Cela a des conséquences lorsqu'on calcule le coût total d'une solution cryptographique. D'autant plus que la sécurité peut nécessiter davantage de personnel, par exemple lorsqu'on déploie une solution M-sur-N pour contrer le risque d'une attaque menée de l'intérieur.

La section 4 liste les conséquences de ces exigences sous forme d'exigences ACID : la solution déployée doit permettre des transactions tout-ou-rien : l'utilisation de la signature et la mise à jour de l'état doivent être dans la même transaction (atomicité).

Suivre ces exigences en logiciel va être difficile : il peut y avoir une mémorisation de certaines informations, ce qui signifie que la version stockée sur un support stable n'a pas été lise à jour, il peut y avoir copie d'une machine virtuelle vers une autre, l'état étant alors dupliqué, etc. Le RFC conseille de plutôt utiliser un composant matériel dédié pour cette tâche délicate de maintien de l'état. Mais si vous voulez vraiment étudier la question en détail, et notamment les différents moyens d'atteindres nos objectifs, la section 5 discute de nombreuses solutions possibles.

Le RFC note aussi que ces exigences ne s'appliquent qu'au signeur. Le vérificateur des signatures, lui, peut complètement ignorer le problème et ne pas connaitre l'état.
