RFC 7696 : Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 22 novembre 2015

Date de publication du RFC : Novembre 2015

http://www.bortzmeyer.org/7696.html

Bien des protocoles de l'IETF utilisent la cryptographie. Ils s'en servent pour la confidentialité, pour l'authentification, pour la signature. L'utilisation pour la confidentialité a évidemment crû en importance depuis les révélations d'Edward Snowden, malgré la guerre contre le chiffrement que mènent de nombreux gouvernements, qui voudraient pouvoir continuer à espionner tranquillement leurs citoyens. Or, un problème de la cryptographie est que les algorithmes ne sont pas éternels : la cryptanalyse trouve parfois, au bout d'un temps plus ou moins long, des moyens de casser un algorithme, qui doit alors être abandonné. Ce fut le cas, par exemple, de RC4, rejeté dans le RFC 7465 [1]. Il est donc **indispensable** que les protocoles utilisant la cryptographie aient la propriété d'**agilité**. Cette propriété désigne le fait qu'on puisse changer d'algorithme sans changer le protocole.

Pour des raisons d'interopérabilité, l'IETF désigne toujours un algorithme cryptographique comme obligatoire : ainsi, deux mises en œuvre d'un protocole donné auront toujours au moins un algorithme en commun, l'algorithme obligatoire, et pourront donc interopérer. Mais l'algorithme obligatoire peut lui aussi céder un jour face à la cryptanalyse, et il faut donc également prévoir le moyen de le remplacer.

[Au passage, la plupart des protocoles IETF utilisent non pas un algorithme unique mais une liste d'algorithmes, la "*cipher suite*". Par exemple, pour TLS, cette liste comprend un algorithme de chiffrement asymétrique (comme RSA), un de chiffrement symétrique (comme AES) et un de condensation (comme SHA-2).]

Revenons à la cryptanalyse. Les algorithmes « vieillissent » donc avec le temps, au fur et à mesure que les cryptanalystes découvrent de nouvelles méthodes pour casser l'algorithme. Ce vieillissement

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7465.txt

n'est pas régulier et n'est pas prévisible. Certains algorithmes sont cassés dans les années qui suivent immédiatement leur publication, d'autres résistent encore (plus ou moins bien) des dizaines d'années après. Ce qui est sûr, c'est que les attaques, au pire stagnent, au mieux se perfectionnent avec le temps : un algorithme ne peut donc pas « rajeunir ». C'est ce vieillissement inéluctable (quoique de durée très variable) qui justifie le principe d'**agilité cryptographique**.

Le principe est simple et semble excellent. Mais l'expérience de la communauté Internet est qu'il est difficile de changer les vieux algorithmes vulnérables par des nouveaux. Même une fois le nouvel algorithme normalisé par l'IETF, et une fois qu'il est mis en œuvre dans les logiciels les plus répandus, on constate souvent que la mise à jour des logiciels est lente et que, des années après l'abandon officiel d'un algorithme, il est toujours disponible et activé dans des millions de machines (et parfois, il est même le seul disponible). Il est donc difficile de supprimer un vieil algorithme vulnérable (comme MD5, cf. RFC 6151), ne serait-ce que pour des raisons d'interopérabilité : si une machine ne sait condenser qu'avec MD5, les machines équipées d'un logiciel plus récent, qui gère SHA-2 et d'autres, devront bien accepter de parler avec MD5 ou, sinon, plus de communication. Bref, les algorithmes morts continuent à hanter l'Internet pendant très longtemps. Pensons par exemple aux administrateurs système qui hésitent encore à supprimer le protocole SSLv3 (cf. RFC 7568) de leurs serveurs HTTPS, de peur de ne plus pouvoir accepter les clients MSIE 6. Il faut souvent recourir à des méthodes de "*name and shame*" comme le test de SSLlabs `<https://www.ssllabs.com/ssltest/>` (qui fait aussi de la "*gamification*", avec ses notes) pour TLS : « cette banque est nulle, leur serveur HTTPS accepte encore 3DES » (ce que le RFC nomme pudiquement « "*social pressure*" »).

Passons au concret avec la section 2 de notre RFC. Il s'agit de règles internes à l'IETF qu'il est fortement recommandé de suivre lors du développement de nouveaux protocoles, pour permettre l'agilité cryptographique. D'abord, évidemment, les protocoles IETF ne doivent **pas** être liés à un algorithme unique, puisque le cassage de celui-ci entrainerait la fin du protocole. Pour permettre le choix d'un algorithme, il faut que chaque algorithme (ou chaque liste d'algorithmes) ait un **identificateur**, un nombre ou une chaîne de caractères qui désigne un algorithme donné. Ainsi, pour DNSSEC, ces identificateurs (8 pour la liste RSA avec SHA-256, 13 pour la liste ECDSA avec la courbe P-256 et SHA-256, etc) sont stockés dans un registre IANA `<https://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xml#dns-sec-alg-numbers-1>`. TLS, lui, utilise des chaînes de caractères comme `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` qui veut dire « échange de clés Diffie-Hellman avec courbes elliptiques, authentification avec RSA, chiffrement symétrique en AES mode GCM, condensation avec SHA-256 ». Cet identificateur est donc la première étape indispensable vers l'agilité cryptographique.

Faut-il un identificateur par algorithme, identificateurs que l'on combine ensuite pour obtenir la liste utilisée, comme le fait IKE (RFC 7296) ou bien un identificateur par liste, comme c'est le cas pour TLS (RFC 5246) ? La question n'est pas évidente. La première méthode permet d'avoir moins d'enregistrements. Mais toutes les combinaisons possibles n'ont pas forcément un sens donc enregistrer les listes (les combinaisons) peut être préférable. Notre RFC ne tranche pas là-dessus.

Dans tous les cas, pour que nos amis Alice et Bob puissent communiquer, il faut qu'ils aient un jeu d'algorithmes en commun (RFC 3365). D'une manière générale, l'IETF prête une grande attention à cette interopérabilité, d'où la notion d'algorithme obligatoire, ou plutôt « obligatoire à implémenter » (à mettre dans les programmes). Sans ces algorithmes obligatoires, on pourrait avoir deux implémentations parfaitement correctes du protocole, mais qui seraient dans l'impossibilité de dialoguer, puisqu'elles n'auraient aucun algorithme en commun. C'est pour cela que chaque protocole doit avoir un ou plusieurs algorithmes (ou listes, si on enregistre les listes) qui soient obligatoires. Pour DNSSEC, par exemple, c'est RSA-SHA1.

Et comme les algorithmes obligatoires peuvent eux aussi être cassés un jour, il est recommandé, pour faciliter leur changement, que le RFC spécifiant le protocole n'indique pas quels sont les algorithmes

obligatoires, mais se réfère à un RFC plus court, sur ce sujet. On pourra ainsi modifier le RFC « Algorithmes » sans toucher au protocole. DNSSEC n'utilise pas cette méthode (l'algorithme obligatoire est défini dans le RFC 4034) mais JOSE <http://www.bortzmeyer.org/jose.html> (cryptographie pour JSON) le fait (les algorithmes sont définis dans le RFC 7518).

Pour la taille des clés utilisées pour ces algorithmes, notre RFC renvoie au RFC 3766 pour la cryptographie asymétrique et au RFC 7525 pour la symétrique.

En général, la mort d'un algorithme cryptographique n'est pas instantanée. Si, en théorie, il est possible qu'un mathématicien génial fasse une percée théorique inattendue qui rende tout à coup le cassage de tel algorithme trivial, en pratique, cela n'arrive que très rarement : on voit plutôt des attaques de plus en plus perfectionnées, un algorithme de plus en plus déconseillé jusqu'à son abandon officiel. C'est un domaine où il y a peu de « coups de tonnerre dans un ciel bleu ». Le passage de DES à AES s'est ainsi étalé sur des dizaines d'années, et celui de RSA vers les courbes elliptiques prendra sans doute autant de temps.

Pour prévenir les programmeurs des changements qu'on peut sans doute attendre, notre RFC recommande, dans l'esprit du RFC 4307, d'augmenter les termes normatifs définis dans le RFC 2119. Ce RFC définissait l'usage de "*MUST*", "*SHOULD*" et "*MAY*" (en majuscules) dans les RFC. Notre nouveau RFC recommande d'utiliser également "*SHOULD+*", "*SHOULD-*" et "*MUST-*". Un "*SHOULD+*" est la même chose qu'un "*SHOULD*" aujourd'hui (« cet algorithme devrait être implémenté, sauf si on a une très bonne raison et qu'on sait ce qu'on fait »). Mais, dans le futur, il est probable qu'il deviendra un "*MUST*" (algorithme obligatoire). Un "*SHOULD-*" est un "*SHOULD*" qui sera probablement dégradé en un simple "*MAY*" dans le futur. Quant au "*MUST-*", il indique que l'algorithme est aujourd'hui obligatoire mais ne le restera sans doute pas.

Passer d'un algorithme à l'autre, on l'a vu, n'est pas trivial. Intuitivement, on se dit que l'IETF normalise le nouvel (et meilleur algorithme), que les programmeurs le programment dans les logiciels et que, au bout d'un moment, les nouveaux logiciels sont assez répandus pour qu'on puisse abandonner officiellement l'ancien algorithme.

Cela suppose qu'on sache mesurer l'état du déploiement et, pour beaucoup de protocoles, il n'y a pas de mécanisme de centralisation des statistiques. On ne sait donc pas si on peut « couper le cordon » et abandonner l'ancien algorithme.

C'est pour cela que la tendance est à mettre dans les nouveaux protocoles un mécanisme de signalisation, permettant de voir le déploiement du nouvel algorithme (par exemple le RFC 6975 pour DNSSEC).

Comme le déploiement est toujours trop long, on peut potentiellement se retrouver dans des situations ennuyeuses où l'ancien algorithme, désormais facilement cassé, est encore largement utilisé. Faut-il alors le supprimer des implémentations, remettant ainsi en cause l'interopérabilité, ou bien le laisser alors qu'on sait que cela met en cause la sécurité ? Débat cornélien qui agite régulièrement le monde TLS, par exemple. Il y a quelques années, la tendance était plutôt à laisser les algorithmes mourir tranquillement, de nos jours, elle est plutôt à éliminer rapidement les « canards boiteux », quitte à renoncer à la communication avec les vieux logiciels. C'est ainsi que l'abandon progressif de SHA-1, désormais largement attaqué <https://sites.google.com/site/itstheshappening/>, a été souvent fait de manière brutale (par exemple Google Chrome décidant soudainement de marquer les certificats utilisant SHA-1 comme étant dangereux).

L'agilité cryptographique n'a pas que des avantages. On a vu qu'elle était indispensable, pour tenir compte des progrès de la cryptanalyse. Mais, si un protocole accepte plusieurs algorithmes, comment

s'assurer que les deux pairs qui négocient utilisent bien le plus fort ? Imaginons un protocole de chiffrement où le serveur indique à la connexion les algorithmes qu'il sait gérer et où le client en choisit un dans la liste, a priori le plus solide, et où le chiffrement démarre ensuite, avec l'algorithme choisi. Un attaquant actif, par exemple un Homme du Milieu, peut modifier la liste d'algorithmes pour mettre l'algorithme le plus faible, qu'il pourra ensuite casser. C'est ce qu'on nomme une attaque par repli ("*downgrade attack*") et c'est la plaie des protocoles qui offrent l'agilité cryptographique. Il faut donc trouver un mécanisme de protection et ne pas garder éternellement les algorithmes trop vieux.

Un problème du même ordre se pose avec les protocoles qui permettent, non seulement de négocier les algorithmes de cryptographie, mais également le mécanisme d'échange initial des clés. C'est le cas d'EAP, par exemple (RFC 3748). Les concepteurs de protocole doivent se souvenir que la complexité est souvent la principale ennemie de la sécurité.

Compte tenu des problèmes que peut entrainer l'agilité cryptographique, notamment la complexité accrue du code, certains pourraient être tentés de concevoir des protocoles à un seul algorithme. Le protocole serait plus simple, et n'offrirait pas de possibilités d'attaques par repli. Un exemple de ce choix avait été WEP. Lié à un seul algorithme, ne pouvant pas évoluer, WEP avait été très long et compliqué à remplacer par un nouveau protocole, qu'il avait fallu concevoir, implémenter puis déployer. WEP est donc un bon exemple de l'importance de l'agilité cryptographique.

La section 3 de notre RFC examine sur quels critères choisir les algorithmes et notamment ceux qui seront obligatoires, et qui doivent donc être de haute qualité cryptographique. Évidemment, le choix ne doit pas être laissé à l'utilisateur final, qui n'a pas les éléments pour le faire. Il vaut mieux aussi qu'il ne soit pas laissé à l'administrateur système, qui n'est typiquement pas un expert en cryptographie. (Regardez la difficulté qu'il y a aujourd'hui à gérer un serveur HTTPS, avec tous ces choix pointus à faire en matière d'algorithmes. TLS est probablement un bon exemple du syndrome « trop de choix ».)

D'abord, les algorithmes obligatoires doivent évidemment faire l'objet d'une spécification publique (pendant longtemps, RC4 était secret). Il doivent avoir fait l'objet d'examens attentifs et de tentatives de cassage par des experts. De préférence, ils ne doivent pas faire l'objet de brevets. Ils doivent évidemment aussi répondre à des exigences plus techniques : ils doivent être rapides, implémentables dans un code de taille réduite, résister aux attaques par canal auxiliaire, etc.

Il existe des algorithmes de cryptographie dits « nationaux » parce que normalisés par une organisation de normalisation nationale officielle et parfois rendus obligatoires par une loi locale (si l'organisation de normalisation est le NIST, on ne les appelle pas « nationaux », le terme est réservé à ceux normalisés en dehors des États-Unis). Ils n'ont pas forcément fait l'objet d'une étude sérieuse et leur but est souvent de simple protectionnisme de l'industrie de sécurité nationale. Auquel cas ils ne doivent pas être choisis dans les protocoles IETF et en tout cas pas activés par défaut. (La section 3.4 du RFC fait pas mal de FUD : il existe d'excellents algorithmes « nationaux » comme ceux du GOST.)

Enfin, la section 4 du RFC, qui examine les questions de sécurité restantes, note que la force de l'algorithme n'est pas tout. Des faiblesses peuvent apparaitre, par exemple en raison de l'ordre des opérations de cryptographie (cf. RFC 7366)