# RFC 8937 : Randomness Improvements for Security Protocols

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 17 octobre 2020

Date de publication du RFC : Octobre 2020

https://www.bortzmeyer.org/8937.html

---

Tout le monde sait désormais que la génération de nombres aléatoires (ou, plus rigoureusement, de nombres imprévisibles, pseudo-aléatoires) est un composant crucial des solutions de sécurité à base de cryptographie, comme TLS. Les failles des générateurs de nombres pseudo-aléatoires, ou bien les attaques exploitant une faiblesse de ces générateurs sont les problèmes les plus fréquents en cryptographie. Idéalement, il faut utiliser un générateur sans défauts. Mais, parfois, on n'a pas le choix et on doit se contenter d'un générateur moins satisfaisant. Ce nouveau RFC décrit un mécanisme qui permet d'améliorer la sécurité de ces générateurs imparfaits, en les nourrissant avec des clés privées.

Un générateur de nombres pseudo-aléatoires cryptographique (CSPRNG, pour "*Cryptographically-Strong PseudoRandom Number Generator*") produit une séquence de bits qui est indistinguable d'une séquence aléatoire. Un attaquant qui observe la séquence ne peut pas prédire les bits suivants.

On les nomme générateur pseudo-aléatoires car un générateur réellement aléatoire devrait être fondé sur un phénomène physique aléatoire. Un exemple classique d'un tel phénomène, dont la théorie nous dit qu'il est réellement aléatoire, est la désintégration radioactive. Des exemples plus pratiques sont par exemple des diodes forcées de produire un bruit blanc (comme dans le Cryptech <https://trac.cryptech.is/wiki/NoisyDiode>). (Ou alors un chat marchant sur le clavier <https://twitter.com/pentestmatt/status/1308000188850675712> ?) Mais, en pratique, les ordinateurs se servent de générateurs pseudo-aléatoires, ce qui suffit s'ils sont imprévisibles à un observateur extérieur.

Beaucoup de solutions de sécurité, même en l'absence de cryptographie, dépendent de cette propriété d'imprévisibilité. Ainsi, TCP, pour protéger contre un attaquant aveugle (situé en dehors du chemin du paquet), a besoin de numéros de séquence initiaux qui soient imprévisibles (RFC 5961 [1]). Et la

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc5961.txt

cryptographie consomme énormément de ces nombres pseudo-aléatoires. Ainsi, TLS (RFC 8446) se sert de tels nombres à de nombreux endroits, par exemple les numniques <https://www.bortzmeyer.org/nonce.html> utilisés dans le `ClientHello`, sans compter les nombres pseudo-aléatoires utilisés lors de la génération des clés de session. La plupart des autres protocoles de chiffrement dépendent de tels nombres que l'attaquant ne peut pas prévoir.

Réaliser un générateur pseudo-aléatoire sur une machine aussi déterministe qu'un ordinateur n'est pas facile, en l'absence de source quantique comme un composant radioactif. (Cf. RFC 4086.) La solution adoptée la plus simple est d'utiliser une fonction qui calcule une séquence de nombres pseudo-aléatoires d'une manière que, même en observant la séquence, on ne puisse pas deviner le nombre suivant (la mathématique fournit plusieurs fonctions pour cela). Comme l'algorithme est en général connu, et que l'ordinateur est déterministe, s'il connait le premier nombre, la **graine**, un observateur pourrait calculer toute la séquence. La graine ne doit donc pas être connue des observateurs extérieurs. Elle est par exemple calculée à partir d'éléments qu'un tel observateur ne connait pas, comme des mouvements physiques à l'intérieur de l'ordinateur, ou bien provoqués par l'utilisateur.

De tels générateurs pseudo-aléatoires sont très difficiles à réaliser correctement et, en cryptographie, il y a eu bien plus de failles de sécurité dues à un générateur pseudo-aléatoire que dues aux failles des algorithmes de cryptographie eux-mêmes. L'exemple le plus fameux était la bogue Debian <https://pdfs.semanticscholar.org/fcf9/fe0946c20e936b507c023bbf89160cc995b9.pdf>, où une erreur de programmation avait réduit drastiquement le nombre de graines possibles (l'entropie). Un autre exemple amusant est le cas du générateur cubain qui oubliait le chiffre 9 <https://www.mattblaze.org/blog/neinnines/>.

Mais le générateur pseudo-aléatoire peut aussi être affaibli délibérement, pour faciliter la surveillance, comme l'avait fait le NIST, sur ordre de la NSA, en normalisant le générateur Dual-EC avec une faille connue <https://projectbullrun.org/dual-ec/documents/dual-ec-20150731.pdf>.

Obtenir une graine de qualité est très difficile, surtout sur des appareils ayant peu de pièces mobiles, donc peu de mouvements physiques qui pourraient fournir des données aléatoires. Un truc avait déjà été proposé : combiner la graine avec la clé privée utilisée pour la communication (par exemple en prenant un condensat de la concaténation de la graine avec la clé privée, pour faire une graine de meilleure qualité), puisque la clé privée est secrète. L'idée vient du projet Naxos <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/strongake-submitted.pdf>. (Cela suppose bien sûr que la clé privée n'ait pas été créée par la source qu'on essaie d'améliorer. Elle peut par exemple avoir été produite sur un autre appareil, ayant davantage d'entropie.) Le problème de cette approche avec la clé privée, est que les clés privées sont parfois enfermées dans un HSM, qui ne les laisse pas sortir.

La solution proposée par notre RFC est de ne pas utiliser directement la clé secrète, mais une signature de la graine. C'est une valeur qui est imprévisible par un observateur, puisqu'elle dépend de la clé privée, que l'attaquant ne connait pas. Ainsi, on peut obtenir une séquence de nombres pseudo-aléatoires que l'observateur ne pourra pas distinguer d'une séquence réellement aléatoire, même si la graine initiale n'était pas terrible.

L'algorithme exact est formulé dans la section 3 du RFC. Le Nième nombre pseudo-aléatoire est l'expansion (RFC 5869, section 2.3) de l'extraction (RFC 5869, section 2.2) d'un condensat de la signature d'une valeur qui ne change pas à chaque itération de la séquence, ce qui fait que cet algorithme reste rapide. (J'ai simplifié, il y a d'autres paramètres, notamment la valeur « normale » de la séquence, initiée à partir de la graine.) La signature doit évidemment être secrète (sinon, on retombe dans le générateur pseudo-aléatoire d'avant ce RFC). La fonction de signature doit être déterministe (exemple dans le RFC 6979). La « valeur qui ne change pas à chaque itération » peut, par exemple (section 4 du RFC) être

l'adresse MAC de la machine mais attention aux cas de collisions (entre VM, par exemple). Une autre valeur constante est utilisée par l'algorithme et peut, par exemple, être un compteur, ou bien l'heure de démarrage. Ces deux valeurs n'ont pas besoin d'être secrètes.

Une analyse de sécurité de ce mécanisme a été faite dans l'article « "*Limiting the impact of unreliable randomness in deployed security protocols*" <https://eprint.iacr.org/2018/1057> ». Parmi les points à surveiller (section 9), la nécessité de bien garder secrète la signature, ce qui n'est pas toujours évident notamment en cas d'attaque par canal auxiliaire.

À propos de générateurs pseudo-aléatoires, je vous recommande cet article très détaillé en français <https://linuxfr.org/news/des-nombres-aleatoires-dans-le-noyau-linux> sur la mise en œuvre de ces générateurs sur Linux.

Merci à Kim Minh Kaplan pour une bonne relecture.