RFC 8032 : Edwards-curve Digital Signature Algorithm (EdDSA)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 25 janvier 2017

Date de publication du RFC : Janvier 2017

http://www.bortzmeyer.org/8032.html

Ce RFC est la description IETF de l'algorithme de signature cryptographique EdDSA. EdDSA est en fait une famille, il prend un certain nombre de paramètres, comme la courbe elliptique Edwards utilisée et ce RFC décrit son utilisation avec les courbes Edwards25519 et Edwards448.

EdDSA n'avait apparemment été décrit auparavant que dans des publications scientifiques, ce RFC sert à la fois à avoir une référence IETF, et également à décrire EdDSA dans des termes plus familiers aux programmeurs. Pourquoi faire de l'EdDSA, d'ailleurs ? Parce que cet algorithme (ou plutôt cette famille d'algorithmes) a plusieurs avantages, notamment :

— Rapide,
— Ne nécessite pas un nombre aléatoire différent par signature (un problème qui a souvent frappé les mises en œuvres de DSA, par exemple avec la Sony Playstation <http://arstechnica.com/gaming/2010/12/ps3-hacked-through-poor-implementation-of-cryptography/>),
— Clés et signatures de taille réduite.

Un site Web sur EdDSA <http://ed25519.cr.yp.to/> a été créé par ses auteurs. La référence officielle d'EdDSA est l'article « "*High-speed high-security signatures*" <http://ed25519.cr.yp.to/ed25519-20110926.pdf> » de D. Bernstein, N. Duif, T. Lange, P. Schwabe, P., et B. Yang. Une extension à d'autres courbes est décrite dans « "*EdDSA for more curves*" <http://ed25519.cr.yp.to/eddsa-20150704.pdf> ». Sur les courbes elles-mêmes, on peut consulter le RFC 7748 [1].

La section 3 du RFC décrit l'algorithme générique d'EdDSA. Comme il laisse ouvert pas moins de onze paramètres, on voit qu'on peut créer une vaste famille d'algorithmes se réclamant d'EdDSA. Mais, évidemment, toutes les combinaisons possibles pour ces onze paramètres ne sont pas sérieuses du point

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7748.txt

de vue de la sécurité cryptographique, et notre RFC ne décrira que cinq algorithmes spécifiques, dont ed25519 et ed448. L'algorithme générique est surtout utile pour la culture générale.

C'est parce que EdDSA est un algorithme générique (au contraire de ECDSA) que les programmes qui l'utilisent ne donnent pas son nom mais celui de l'algorithme spécifique. Ainsi, OpenSSH vous permet de générer des clés Ed25519 (`ssh-keygen -t ed25519`) mais pas de clés EdDSA (ce qui ne voudrait rien dire).

La section 4 du RFC décrit en détail un des paramètres importants de EdDSA : le choix de la fonction "*prehash*". Celle-ci peut être l'identité (on parle alors de "*Pure EdDSA*") ou bien une fonction de condensation cryptographique (on parle alors de "*Hash EdDSA*").

La section 5, elle, spécifie EdDSA avec les autres paramètres, décrivant notamment Ed25519 et Ed448. Ainsi, Ed25519 est EdDSA avec la courbe Edwards25519, et une fonction "*prehash*" qui est l'identité. (Pour les autres paramètres, voir le RFC.) L'algorithme Ed25519ph est presque identique sauf que sa fonction "*prehash*" est SHA-512.

Comme tout algorithme de cryptographie, il faut évidemment beaucoup de soin quand on le programme. La section 8 du RFC contient de nombreux avertissements indispensables pour le programmeur. Un exemple typique est la qualité du générateur aléatoire. EdDSA n'utilise pas un nombre aléatoire par signature (la plaie de DSA), et est déterministe. La sécurité de la signature ne dépend donc pas d'un bon ou d'un mauvais générateur aléatoire. (Rappelons qu'il est très difficile de faire un bon générateur aléatoire, et que beaucoup de programmes de cryptographie ont eu des failles de sécurité sérieuses à cause d'un mauvais générateur.) Par contre, la génération des clés, elle, dépend de la qualité du générateur aléatoire (RFC 4086).

Il existe désormais pas mal de mises en œuvre d'EdDSA, par exemple dans OpenSSH cité plus haut. Sinon, les annexes A et B du RFC contiennent une mise en œuvre en Python d'EdDSA. Attention, elle est conçue pour illustrer l'algorithme, pas forcément pour être utilisée en production. Par exemple, elle n'offre aucune protection contre les attaques exploitant la différence de temps de calcul selon les valeurs de la clé privée (cf. la section 8.1). J'ai extrait ces deux fichiers, la bibliothèque (en ligne sur `http://www.bortzmeyer.org/files/eddsalib.py`) et le programme de test (en ligne sur `http://www.bortzmeyer.org/files/eddsa-test.py`) (ils nécessitent Python 3). Le programme de test prend comme entrée un fichier composé de plusieurs vecteurs de test, chacun comprenant quatre champs, séparés par des deux-points, clé secrète, clé publique, message et signature. La section 7 du RFC contient des vecteurs de test pour de nombreux cas. Par exemple, le test 2 de la section 7.1 du RFC s'écrit `4ccd089b28ff96da9db6c346ec114e0f5b8a319f35aba624da8cf6ed4fb8a6fb:3d4017c3e843895a92b70` et l'exécution du programme de test affiche juste le numéro de ligne quand tout va bien :

```
% python3 eddsa-test.py < vector2.txt
1
```

On peut aussi utiliser des tests plus détaillés comme ce fichier de vecteurs de test <`http://ed25519.cr.yp.to/python/sign.input`> :

```
% python3 eddsa-test.py < sign.input
1
2
3
...
1024
```

Si on change le message, la signature ne correspond évidemment plus et le programme de test indique une assertion erronée :

```
% python3 eddsa-test.py < vector2-modified.txt
1
Traceback (most recent call last):
  File "eddsa-test.py", line 30, in <module>
    assert signature == Ed25519.sign(privkey, pubkey, msg)
AssertionError
```