Assurer l'authenticité des données stockée dans une DHT

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 22 avril 2008

https://www.bortzmeyer.org/authenticite-dans-dht.html

Les DHT sont un moyen entièrement pair-à-pair de stocker de l'information dans le réseau. Leur force est la résistance aux attaques, leur faiblesse le fait qu'on ne puisse pas garantir l'authenticité des données stockées, puisque le mécanisme classique de « faire confiance au serveur qui fait autorité », utilisé par exemple pour le DNS, ne s'applique pas aux DHT où on ignore même sur lequel des milliers de serveurs de la DHT nos précieuses données seront stockées.

Il faut donc utiliser la cryptographie et signer les données.

Voici donc un protocole possible, suivi de son application au service OpenDHT <https://www.bortzmeyer.org/opendht-debut.html>. L'expérience prouve largement que les amateurs (et même parfois les professionnels) ne voient jamais les failles de sécurité des protocoles cryptographiques qu'ils conçoivent : les critiques sont donc les bienvenues.

Pour éviter les collisions sur le mot « clé », je ne l'utilise que pour la clé cryptographique. Je nomme « nom de l'attribut » (ou index) l'« endroit » où on veut stocker et « valeur de l'attribut » la chose qu'on veut stocker.

Chaque entité qui veut stocker des trucs dans la DHT a une clé cryptographique asymétrique. Pour un protocole comme HIP (RFC 5201 [1]), la clé serait le HI ("*Host Identifier*").

— Avant d'écrire, l'entité concatène la partie publique de sa clé et le nom de l'attribut.
— Elle hache le résultat et ça donne un index de la DHT.
— Elle écrit à l'index deux valeurs, la valeur proprement dite et une signature cryptographique.
— L'entité qui lit connait la clé publique et connait le nom de l'attribut. Elle peut donc reproduire les deux premières étapes. Elle lit alors la donnée et et peut vérifier la signature.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc5201.txt

Notez qu'on n'a pas dit comment on connaissait les clés publiques de ses partenaires. Le mécanisme pour cela est en dehors de ce protocole.

Les failles que je vois pour l'instant :

— Une donnée écrite, puis supprimée peut être rejouée par un pair méchant (ou par un "*sniffer*" car je ne suis pas sûr qu'OpenDHT permette du https). Il faudrait un système d'horloge logique pour éviter cela.
— Il doit y en avoir beaucoup d'autres !

Voici une mise en œuvre possible en Python. J'utilise la bibliothèque pyme <https://www.bortzmeyer.org/gpgme.html> pour signer en OpenPGP.

Le programme qui écrit la donnée (en ligne sur https://www.bortzmeyer.org/files/opendht-authentic.py), d'abord. Il s'utilise ainsi :

```
% python put.py -k 82AEEF63  quelquechose.example.org 192.0.2.42
```

La ligne ci-dessus stocke la valeur 192.0.2.42 dans la DHT, indexée par le nom quelquechose.example.org, en signant avec la clé OpenPGP 0x82AEEF63. Il fabrique les deux noms (celui composé de la clé et de l'attribut et celui qui sert à stocket la signature) :

```
key = Binary(sha.new(pgp_key + attribute).digest())
key_for_sig = Binary(sha.new(pgp_key + attribute + ".SIGNATURE").digest())
```

puis écrit dans la DHT :

```
proxy.put(key, bin_value, ttl, "put-authentic.py")]
proxy.put(key_for_sig, Binary(signature), ttl, "put-authentic.py")
```

Le programme qui lit les données (en ligne sur https://www.bortzmeyer.org/files/opendht-authentic.py) vérifie les signatures. Il s'utilise ainsi :

```
%  python get.py -k 82AEEF63  quelquechose.example.org
```

À noter que rien n'empêche plusieurs écritures sur le même nom (les attributs sont multi-valués dans OpenDHT) donc le programme doit boucler sur tous les résultats obtenus. Autrement, il fait simplement l'inverse du programme d'écriture :

```
signatures, pm = proxy.get(key_for_sig, maxvals, pm, "get-authentic.py")
vals, pm = proxy.get(key, maxvals, pm, "get-authentic.py")
```

Si un méchant a essayé d'insérer de fausses données, cela se voit tout de suite (rappelons qu'on ne peut pas empêcher cette insertion, juste la détecter a posteriori) :

```
% python get.py -k 82AEEF63   quelquechose.example.org

Good signature from:
  uid:         OpenDHT Authenticity Test (Test only, not a real user) <stephane+opendht@sources.org>
Value is "192.0.2.42"

BAD signature from:
  uid:         OpenDHT Authenticity Test (Test only, not a real user) <stephane+opendht@sources.org>
Value is "102.0.2.42"
```