RFC 6920 : Naming Things with Hashes

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 22 avril 2013. Dernière mise à jour le 23 avril 2013

Date de publication du RFC : Avril 2013

https://www.bortzmeyer.org/6920.html

La question des identificateurs sur le Web agite des électrons depuis le début. Il n'existe pas d'identificateurs idéaux, ayant toutes les bonnes propriétés <https://www.bortzmeyer.org/nommage-beurre.html>. Dans la grande famille des URI (RFC 3986 [1]), il faut donc choisir selon l'importance qu'on donne à tel ou tel critère. Par exemple, si on attache du prix à la stabilité de l'identificateur et qu'on veut qu'il ne désigne pas seulement un endroit où se trouve un contenu, mais qu'on veut qu'il désigne ce contenu et pas un autre ? Alors, on peut choisir les nouveaux URI `ni:` ("*Named Information*") qui désignent un contenu par un condensat ("*hash*"). Un URI `ni:` ne change pas si le contenu change de serveur, mais il est modifié si le contenu lui-même change. C'est donc une forme d'adressage par le contenu.

À quoi cela sert ? À éviter un problème de l'indirection : si je dis « regardez l'image en `http://www.example.org/ho` que l'image de vacances à la plage est remplacée par une photo de LOLcat, l'URI sera toujours valable, alors que le contenu a changé. Inversement, si un webmestre incompétent et qui n'a pas lu « "*Cool URIs don't change*" <http://www.w3.org/Provider/Style/URI.html> » réorganise le site et met le contenu en `http://www.example.org/bigcomplicatedcmswithsecurityholes.php?kind=image&tag=beach` l'URI ne marchera plus alors que le contenu est inchangé. Un autre exemple, plus positif, est le cas où un contenu est répliqué en plusieurs endroits, ayant des URL différents. Il faut pouvoir désigner le contenu, indépendamment du service qui l'héberge. C'est pour répondre à ces deux problèmes qu'a été créé par ce RFC le plan d'URI `ni:` (désormais dans le registre IANA des plans d'URI <https://www.iana.org/assignments/uri-schemes.html>). Le contenu en question pourra donc être désigné par `ni:///sha256;rJAeWFhQWIoTExwyEQ8w_L5uB0UkfmnCGfNIPy7CdDc`. Une telle technique a déjà été employée mais de manière non standard (par exemple, on voit parfois des URI du genre `http://www.example.org/hash?function=sha256&value=rJAeWFhQWIoTExwyEQ8w_L5uB0UkfmnCGfNIPy7C` où un programme situé sur `www.example.org` récupère la ressource selon le contenu). Désormais, le but est de permettre du vrai adressage par le contenu sur le Web. Cela se nomme "*information-centric*" sur les PowerPoint des gourous. Sur ce sujet, le RFC recommande la lecture de « "*Design Considerations*

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc3986.txt`

for a Network of Information" <http://conferences.sigcomm.org/co-next/2008/CoNext08_proceedings/ReArch08Papers/1569148930.pdf> » de Ahlgren, D'Ambrosio, Dannewitz, Marchisio, Marsh, Ohlman, Pentikousis, Rembarz, Strandberg, et Vercellone, ainsi que des articles de Van Jacobson <https://www.bortzmeyer.org/van-jacobson-ccn.html>. On peut aussi lire des articles sur les autres mécanismes d'adressage par le contenu comme les "*magnet links*" de plusieurs systèmes pair-à-pair, comme Freenet.

Une fois le contenu récupéré (en HTTP ou par d'autres moyens), le lecteur peut alors recalculer le condensat et vérifier s'il a bien reçu le bon fichier (des protocoles comme BitTorrent utilisent un mécanisme analogue pour s'assurer que le fichier transmis par les pairs est bien celui qu'on voulait).

D'autres informations peuvent se retrouver dans l'URI `ni:` (voir des exemples plus loin) mais la comparaison de deux URI `ni:` se fait uniquement sur le couple {fonction de hachage utilisée, valeur du condensat}.

Le condensat est calculé par une fonction de hachage cryptographique et, par défaut, c'est SHA-256 (vous avez noté le `sha256` dans l'URI `ni:` donné en exemple plus haut ?) Les valeurs possibles pour l'algorithme de hachage figurent dans un nouveau registre <https://www.iana.org/assignments/named-information/named-information.xml#hash-alg>. Les nouvelles valeurs sont enregistrées selon une procédure légère d'examen par un expert (RFC 5226 et section 9.4 de notre RFC).

Les condensats de SHA-256 sont de grande taille, parfois trop pour certaines utilisations. On a donc le droit de les tronquer à leurs N premiers bits et le nom d'algorithme indiqué doit préciser cette troncation. Ainsi, si on garde les 32 premiers bits, on doit indiquer `sha256-32` et pas juste `sha256`. Attention, c'est évidemment au détriment de la sécurité (si la sortie de SHA-256 est si longue, c'est pour une bonne raison, cf. RFC 3766) et ces condensats raccourcis, quoique simples à manipuler, ne protègent plus tellement. (Notez que le VCS git, qui identifie les "*commits*" par un condensat cryptographique, permet également de les raccourcir, pour faciliter son utilisation, par exemple depuis le "*shell*".)

Le format exact des URI `ni:` figure en section 3. On note un composant dont je n'ai pas encore parlé, l'autorité. On peut écrire `ni:///sha256;rJAeWFhQWIoTExwyEQ8w_L5uB0UkfmnCGfNIPy7CdDc` mais aussi `ni://www.example.net/sha256;rJAeWFhQWIoTExwyEQ8w_L5uB0UkfmnCGfNIPy7CdDc`. L'autorité (ici, `www.example.net`) désigne un endroit où on pourra peut-être récupérer la ressource. Le gros problème des identificateurs fondés sur le contenu du fichier, en effet, est qu'ils sont inutiles pour accéder effectivement au fichier : pas moyen de télécharger un fichier dont on ne connait que le condensat ! Il existe plusieurs solutions et l'une d'elles est de passer par l'autorité. L'idée est que l'URI `ni:` est automatiquement transformé en un URL HTTP sous `.well-known` (cf. RFC 8615 et section 4 de notre RFC). Ainsi, `ni://www.example.net/sha256;rJAeWFhQWIoTExwyEQ8w_L5uB0UkfmnCGfNIPy7CdDc` devient `http://www.example.net/.well-known/ni/sha256/rJAeWFhQWIoTExwyEQ8w_L5uB0UkfmnCGfN` qu'on peut ensuite récupérer par des moyens classiques. On combine alors les avantages de l'adressage par le contenu (le condensat est là pour vérifier le contenu) et du fait que les URL marchent bien pour récupérer un contenu. On notera que le système d'identificateurs ARK a un mécanisme analogue (un identificateur stable et non résolvable **plus** un préfixe qui permet de faire un URL résolvable et actionnable). On a ainsi le beurre et l'argent du beurre. `ni` fait désormais partie des termes enregistrés dans le registre "well-known" <https://www.iana.org/assignments/well-known-uris/well-known-uris.xml>.

Petite question : pourquoi `http:` et pas `https:`, qui serait plus sûr ? Parce que tous les serveurs ne gèrent pas HTTPS et aussi parce que ce n'est pas nécessaire pour s'assurer de l'intégrité du fichier récupéré, le condensat cryptographique suffit. Bien sûr, une mise en œuvre de ce RFC est autorisée à essayer avec HTTPS, par exemple pour la confidentialité.

Comme indiqué plus haut, la comparaison entre deux URI `ni:` se fait uniquement sur le couple {algorithme, condensat} donc `ni://datastore.example/sha256;rJAeWFhQWIoTExwyEQ8w_L5uB0UkfmnCGfNIPy` et `ni://www.example.net/sha256;rJAeWFhQWIoTExwyEQ8w_L5uB0UkfmnCGfNIPy7CdDc` sont identiques.

Au fait, SHA-256 peut être affiché en binaire ou bien sous une forme hexadécimale. Laquelle est utilisée ? Le format binaire **suivi** d'un encodage en Base64 (RFC 4648), sous sa forme "*URL encoding*", celle qui utilise _ et - au lieu de / et + (ce n'est pas la forme par défaut : songez que l'outil `base64` sur Unix ne permet pas de produire cette variante).

Si on n'a pas besoin d'être lisible par des humains et transmissible dans des textes, il existe aussi une forme binaire, plus compacte, des URI `ni:`, spécifiée en section 6.

Signe de leur souplesse, les URI `ni:` ont également une forme « prononçable ». S'il faut dicter un URI au téléphone, l'encodage par défaut est très ambigu (par exemple, minuscules et majuscules n'ont pas la même valeur). D'où la syntaxe `nih:` ("*NI for Humans*" et non pas "*Not Invented Here*", cf. RFC 5513) de la section 7. Les URI sous leur forme `nih:` sont en Base16 (RFC 4648), peuvent inclure des tirets supplémentaires, pour la lisibilité, et incluent une somme de contrôle (le dernier chiffre, calculé selon l'algorithme de Luhn de la norme ISO 7812) pour détecter les erreurs de compréhension. Un exemple est `nih:sha-256-120;5326-9057-e12f-e2b7-4ba0-7c89-2560-a2;f` .

Enfin, des paramètres peuvent apparaître dans l'URI, par exemple pour indiquer le type de la ressource (`ni:///sha256;rJAeWFhQWIoTExwyEQ8w_L5uB0UkfmnCGfNIPy7CdDc?ct=image/png`). Une liste de paramètres possibles est enregistrée <`https://www.iana.org/assignments/named-information/named-information.xml#uri-parameter`>.

À noter que l'abréviation `ni` veut officiellement dire "*Named Information*" mais que tout "*geek*" va évidemment penser aux chevaliers qui disent Ni...

Avant d'utiliser les URI `ni:`, il est prudent de lire la section 10, consacrée aux questions de sécurité. Ainsi, il ne faut pas oublier que le condensat cryptographique n'est **pas** une signature. Il sert à vérifier l'intégrité mais, comme n'importe qui peut générer un condensat pour n'importe quel contenu, il ne prouve rien quant à l'authenticité.

La fonction SHA-256 et ses camarades ne sont pas inversibles. D'un condensat, on ne peut pas remonter au contenu. Toutefois, celui-ci n'est pas vraiment secret. Un attaquant peut toujours deviner plus ou moins le contenu (c'est particulièrement facile si le contenu est très structuré, avec peu de variations possibles) et tester les différentes possibilités. Il peut aussi utiliser un moteur de recherche, si des pages existent déjà avec la correspondance entre une ressource et son condensat (pour empêcher cela, il faudrait que les condensats `ni:` soient salés, ce qui n'est pas le cas).

Et, naturellement, si vous utilisez des condensats tronqués, comme le permet de RFC, vous perdez beaucoup en sécurité.

La notion d'**autorité** dans les URI `ni:` peut être trompeuse. Le nom de domaine qui peut apparaître dans un URI `ni:` n'est pas forcément la source du contenu, puisque n'importe qui peut copier la ressource, et la servir depuis un programme. Il ne faut donc pas attribuer de sémantique à la soi-disant « autorité ».

Si vous voulez regarder un tel système « en vrai », les articles de ce blog sont tous accessibles via un URI `ni:`. L'identificateur est calculé toutes les nuits et stocké dans une base de données. Un simple petit programme WSGI <https://www.bortzmeyer.org/wsgi.html> permet ensuite de récupérer un fichier en fonction de son identificateur. À noter que ce n'est pas la forme HTML qui est utilisée mais le source en XML (la forme en HTML change trop souvent, par exemple si les outils qui la produisent automatiquement à partir du source sont modifiés). Ainsi, l'URI `ni:///sha256;6OuucQ1RgugCDVinT2RGmzYYpra0fe` correspond au source XML de cet article <https://www.bortzmeyer.org/dame-blanche.html>.
En indiquant explicitement l'autorité (le serveur qui permet de faire la récupération), c'est l'URI `ni://www.bortzmey`
Et la version sous forme d'URL est . Si vous préferez un autre article, `ni:///sha256;1etMCVZtd7_-cq38MrtnQcoZW_e7J2cslulrFp92lueI` correspond au source de cet article <https://www.bortzmeyer.org/samknows.html>.

Notez qu'il n'est pas possible de mettre l'URI `ni:` de l'article que vous êtes en train de lire dans cet article (inclure le condensat change l'article et il faut donc changer le condensat, ce qui change l'article...)

Vous voulez vérifier ? Allons-y.

```
% wget -O /tmp/x.xml https://www.bortzmeyer.org/.well-known/ni/sha256/1etMCVZtd7_cq38MrtnQcoZW_e7J2cslulrFp
...
% openssl dgst -sha256 -binary /tmp/x.xml | base64
1etMCVZtd7/cq38MrtnQcoZW/e7J2cslulrFp92lueI=
```

Et on retrouve bien l'identificateur `1etMCVZtd...` (aux transformations "*URL encoding*" près).

Si vous voulez faire depuis le shell Unix les calculs nécessaires, voici quelques exemples avec OpenSSL. Pour calculer le NI de "*Hello World !*" :

```
% echo -n 'Hello World!' | openssl dgst -sha256 -binary | base64
f4OxZX/x/FO5LcGBSKHWXfwtSx+j1ncoSt3SABJtkGk=
```

Il faut ensuite rechercher/remplacer car `base64` (ou bien la commande `openssl enc -base64`) ne sait pas faire de l'"*URL encoding*" de Base64. Avec sed :

```
% echo -n 'Hello World!' | openssl dgst -sha256 -binary | base64 | sed -e 's#/#_#g' -e 's#+#-#g' -e 's#=$##'
f4OxZX_x_FO5LcGBSKHWXfwtSx-j1ncoSt3SABJtkGk
```

(Pour la même manipulation, on peut aussi utiliser `tr` : `tr -cd a-zA-Z0-9+/ | tr +/ -_-`. Comme expliqué par Kim-Minh Kaplan : « Note le `tr -cd` pour nettoyer le résultat de l[Caractère Unicode non montré [2]]encodage en Base 64. Si avec SHA-256 il n[Caractère Unicode non montré]est pas nécessaire, avec SHA-512, l[Caractère Unicode non montré]encodeur d[Caractère Unicode non montré]OpenSSL introduira un retour à la ligne qu[Caractère Unicode non montré]il faudra aussi supprimer. ») Si on reprend l'exemple plus haut, on peut combiner les deux opérations : on récupère le fichier grâce au condensat et on vérifie que le contenu est bien le contenu attendu. Utilisons ce simple script :

2. Car trop difficile à faire afficher par LaTeX

```
#!/bin/sh

BASE_URL="https://www.bortzmeyer.org/.well-known/ni/sha256/"

if [ -z "$1" ]; then
    echo "Usage: $0 ni" >&2
    exit 1
fi

ni=$1

hash=$(wget -q -O - ${BASE_URL}/${ni} | openssl dgst -sha256 -binary | openssl enc -base64 | \
    sed -e 's#/#_#g' -e 's#+#-#g' -e 's#=$##')

if [ "$hash" != "$ni" ]; then
    echo "Error: hash is $hash instead of $ni" >&2
    exit 1
else
    exit 0
fi
```

Et voyons :

```
% get-and-check-ni 1etMCVZtd7_cq38MrtnQcoZW_e7J2cslulrFp92lueI
%
```

Si on met une valeur fausse :

```
% get-and-check-ni 1etMCVZtd7_cq38MrtnQcoZW_e7J2cslulrFp922ueI
Error: hash is 47DEQpj8HBSa-_TImW-5JCeuQeRkm5NMpJWZG3hSuFU instead of 1etMCVZtd7_cq38MrtnQcoZW_e7J2cslulrFp922ue
```

Si vous voulez lire de source de mon programme, il est dans les fichiers de mon blog (en ligne sur `https://www.bortzmeyer.org/files/blog-support-files.tar.gz`), `scripts/blog2db/blog2db.py` pour le stockage dans la base de données et `wsgis/ni.py` pour la récupération.

Il existe une autre implémentation, par certains des auteurs du RFC, en <`http://sourceforge.net/projects/netinf/`>.

Si vous voulez d'autres lectures, le RFC 1737. sur les URN, parlait déjà (en 1994) d'utiliser un condensat cryptographque (avec MD5) pour désigner un contenu. En 2003, une proposition plus élaborée, `draft-thiemann-hash-urn` décrivait un système d'URN avec adressage par le contenu (des choses comme `urn:hash::sha1:LBPI666ED2QSWVD3VSO5BG5R54TE22QL`). Mais, avec `draft-thiemann-hash-urn`, il y a un gros manque : pas moyen d'indiquer un (ou plusieurs, comme dans le cas des "*magnets*") serveur pouvant servir le document en question. Avec `draft-thiemann-hash-urn`, on n'a que des URN (ils ne sont pas « actionnables »). Enfin, si vous voulez une critique des `ni:` par rapport aux "*magnets*", voyez la discussion sur LinuxFr <`http://linuxfr.org/nodes/98097/comments/1446522`>.