RFC 6698 : The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 19 août 2012

Date de publication du RFC : Août 2012

http://www.bortzmeyer.org/6698.html

À chaque seconde, d'innombrables transactions sur l'Internet sont protégées contre l'écoute et la modification malveillante par le protocole TLS. Ce dernier est capable de chiffrer la session (pour la protéger contre l'écoute par un tiers) et d'authentifier le pair avec qui on parle (pour s'assurer qu'on n'est pas en train de communiquer avec un usurpateur). Comment TLS authentifie t-il ? La méthode la plus courante aujourd'hui est de se servir de certificats X.509 vérifiés (en théorie) par une Autorité de Certification. Ce mécanisme repose sur une confiance aveugle dans de très nombreuses organisations, et a de nombreuses faiblesses, comme on l'a vu en 2011 où le piratage de plusieurs AC a permis à des attaquants d'obtenir de « vrais-faux » certificats, y compris pour des organisations qui n'étaient pas clientes des AC piratées. La démonstration étant ainsi faite que X.509 n'était pas digne de la confiance que certains lui accordent, il restait à concevoir une meilleure solution. Rendue possible par le déploiement de DNSSEC, voici DANE ("*DNS-based Authentication of Named Entities*") et ses enregistrements DNS `TLSA`. DANE permet à chacun de publier de manière sécurisés ses certificats, bouchant ainsi les vulnérabilités de X.509, ou permettant même de s'en passer complètement.

Avant de décrire DANE, il faut comprendre le fonctionnement de X.509, afin de voir pourquoi il ne garantit pas grand'chose en matière de sécurité. (La section 1 du RFC contient une description plus détaillée.) Un client qui se connecte à un serveur et qui craint, souvent à juste titre, que la session soit écoutée, voire modifiée par un attaquant situé sur le chemin, va utiliser le protocole TLS, normalisé dans le RFC 5246 [1] pour chiffrer la session. (TLS est utilisé dans de nombreuses applications, la plus connue étant le HTTPS du RFC 2818, celui du petit cadenas dans le navigateur Web.) Ce chiffrement le protège contre l'écoute passive. Mais d'autres risques se cachent derrière. Si le client croit parler à une machine alors qu'en fait, par le biais de piratages portant sur DNS, BGP ou un autre protocole, il

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc5246.txt`

parle à la machine de l'attaquant, le chiffrement seul ne le protégera pas. Pour cette raison, TLS ajoute l'idée d'authentification. Le client peut vérifier que le serveur est bien ce qu'il prétend être (l'inverse, l'authentification du client par le serveur, existe aussi mais est plus rare).

Aujourd'hui, cette authentification se fait essentiellement via X.509 (il existe d'autres voies, très marginales, comme celle du RFC 6091). Le principe de X.509 est le suivant. La clé publique du serveur TLS est contenue dans un certificat, contenant le nom du serveur, et signé par une autorité reconnue. En vérifiant la signature, le client TLS peut s'assurer qu'il parle bien à la machine souhaitée (on dit que TLS **associe** un nom à une clé). Si un navigateur Web se connecte à `https://www.dns-oarc.net/`, il doit recevoir un certificat signé et comportant le nom `www.dns-oarc.net` (la question du nom à indiquer est très compliquée dans X.509, voir le RFC 6125). Notez qu'en fait, TLS utilise un sous-ensemble de X.509, spécifié dans le RFC 5280 mais ce n'est pas essentiel pour cette analyse.

Mais quelles sont les « autorités reconnues » ? La plupart du temps, elles ont été choisies et configurées par l'éditeur du logiciel utilisé. Si vous être adepte de Firefox, c'est la fondation Mozilla qui a décidé que les centaines d'autorités de certification (AC) que connait ce logiciel sont dignes de confiance.

Tout repose donc sur l'AC. Si une AC est malhonnête, ou bien a été piratée, elle peut émettre des vrais-faux certificats, qui tromperont l'utilisateur. Cette dépendance vis-à-vis d'un tiers est traditionnellement une des faiblesses les plus critiquées de X.509, d'autant plus que les dites AC, pressées par des considérations économiques, ne font pas toujours de gros efforts pour vérifier les titulaires de certificats.

Mais il y a pire. Rappelez-vous qu'il suffit à un certificat de comporter la signature d'**une** AC pour être accepté. Il n'y a pas de limites aux noms pour lesquelles les AC peuvent s'engager. Si l'AC DigiNotar est reconnue comme AC de confiance par un client TLS, elle peut signer des certificats pour **n'importe quel nom**, même si le nom est géré par une organisation qui a choisi une autre AC. C'est le problème le plus fondamental de X.509. Ce système, conçu par l'UIT, supposait que le monde se divisait en deux, une masse de clients payants et une oligarchie d'AC toutes parfaites. Ce modèle a éclaté devant la réalité des choses : les AC ne sont pas toutes parfaites. L'année 2011 a montré que cette vulnérabilité, connue depuis longtemps, n'était pas purement théorique. Le détournement de l'AC Comodo, puis le piratage de DigiNotar, au cours de l'opération Tulipe Noire <`http://www.rijksoverheid.nl/ministeries/bzk/documenten-en-publicaties/rapporten/2011/09/05/diginotar-public-report-version-1.html`>, ont montré que les AC, non seulement étaient vulnérables mais que, en outre, les vrais-faux certificats marchaient bien. C'est ainsi que le gouvernement iranien a pu, suite à Tulipe Noire, obtenir de vrais-faux certificats pour Gmail (alors que Google n'était **pas** client de DigiNotar), lui permettant d'observer le trafic de ses citoyens.

Bon, tout cela est bien triste, mais comment DANE empêche t-il cela ? Le principe est, plutôt que d'introduire un nouvel acteur, de réutiliser une infrastructure existante, le DNS avec des identificateurs existants, les noms de domaine. Le certificat est publié dans le DNS et signé avec DNSSEC (je schématise : lisez plus loin pour une description complète) et le client TLS peut alors le vérifier. Bien sûr, le piratage d'un registre DNS (ou d'autres acteurs) permettra de court-circuiter cette protection mais l'énorme avantage de cette technique est qu'on n'a pas besoin de faire confiance à toute la planète, seulement aux fournisseurs qu'on a choisi (et desquels on dépendait déjà). Ainsi, si on décide de fonder son identité en ligne sur un `.com`, on dépend de la sécurité de VeriSign mais **seulement de celle-ci**. En cas de piratage ou de malhonnêteté de VeriSign, seuls les `.com` seront affectés, les gens qui ont un `.fr` (par exemple) ne seront pas touchés.

Cette idée avait été documentée dans le RFC 6394, premier RFC du groupe de travail DANE <`http://tools.ietf.org/wg/dane`>, créé à la fois en raison des faiblesses manifestes de X.509, et parce que le déploiement de DNSSEC rendait réaliste une solution alternative. Le protocole DANE réalise le cahier des charges du RFC 6394 et crée pour cela un nouveau type d'enregistrement DNS, `TLSA`

(ne cherchez pas une signification à ce sigle, il n'en a officiellement pas). Tel qu'actuellement normalisé, DANE couvre les besoins de HTTPS mais, pour des protocoles avec une indirection dans le DNS (comme les MX du RFC 5321 ou les SRV du RFC 6120), c'est plus compliqué et il faudra attendre de futurs RFC <http://www.bortzmeyer.org/redirection-certificat.html> pour adapter les TLSA à ces cas.

Notez que d'autres protocoles peuvent aussi mettre leurs clés dans le DNS comme SSH avec le RFC 4255 et IPsec avec le RFC 4025.

À noter que notre RFC demande que les enregistrements DANE/TLSA soient protégés par DNSSEC mais ne précise pas comment le client le vérifie. Il existe plusieurs façons de déployer DNSSEC et notre RFC préfère ne pas trancher prématurément.

Bien, maintenant, place à DANE lui-même. Le cœur de ce protocole est le nouveau type d'enregistrement DNS TLSA, présenté en section 2. Il a le type 52 dans le registre IANA <https://www.iana.org/assignments/dns-parameters> (cf. section 7).

Un enregistrement TLSA comprend :

— un champ « Utilisation du certificat » (*"Certificate usage"*),
— un champ « Sélecteur » (*"Selector"*),
— un champ « Méthode de correspondance » (*"Matching type"*),
— un champ « Données » (*"Data"*).

Le premier, « Utilisation du certificat », voit ses valeurs possibles enregistrées dans un nouveau registre IANA <https://www.iana.org/assignments/dane-parameters/dane-parameters.xml#certificate-usages> (voir aussi la section 7). On pourra ainsi ajouter de nouvelles utilisations sans modifier ce RFC. Pour l'instant, sont définies les valeurs numériques :

— 0 : l'enregistrement TLSA est alors utilisé comme « contrainte sur l'AC ». Il spécifie un certificat qui doit être présent dans la chaîne des certificats utilisés pour la validation X.509. Il vient donc en complément de X.509. Son rôle est de lutter contre des attaques par une AC qui n'est pas celle choisie. Si on est client de GeoTrust, on peut mettre le certificat de l'AC GeoTrust dans l'enregistrement TLSA et, ainsi, un vrai/faux certificat émis par Comodo ou DigiNotar sera refusé. L'utilisation 0 est la plus traditionnelle, sans remplacer X.509 et bouchant simplement sa principale vulnérabilité. Attention si vous changez d'AC, il faudra quand même penser à mettre à jour vos enregistrements TLSA.
— 1 : l'enregistrement TLSA indique le certificat du serveur TLS (qui devra toujours, comme dans l'utilisation 0, être validé selon les règles du RFC 5280). Il s'agit d'une « contrainte sur le certificat » et plus seulement sur l'AC. Il permet donc de se protéger contre une attaque par sa propre AC. Si elle décide d'émettre un nouveau certificat sans vous prévenir, il ne sera pas accepté. Attention en pratique, on change de certificats plus souvent que d'AC et il ne faut pas oublier de mettre à jour le TLSA.
— 2 : cette fois, on quitte les rivages de X.509. Un certificat ainsi publié n'a plus besoin de passer la validation X.509 ou, plus exactement, la validation ne se fait que via l'AC qui détient ce certificat. Il s'agit donc de déclarer dans le DNS une nouvelle AC, en ignorant les AC pré-définies du client TLS. Un exemple typique est le cas d'une organisation qui est sa propre AC et émet ses propres certificats. Elle peut ainsi les voir acceptés automatiquement, sans payer une AC extérieure dont la fiabilité est douteuse.
— 3 : comme l'utilisation 2, on se passe de l'infrastructure des AC traditionnelles, et on déclare dans le DNS, non pas une AC à soi comme dans l'utilisation 2, mais un certificat. C'est l'équivalent des certificats auto-signés mais avec cette fois une preuve de validité, offerte par DNSSEC.

Depuis le RFC 7218, il y a également des mnémoniques standards pour ces valeurs, ainsi que celles des deux champs suivants.

Le sélecteur, quant à lui, fait également l'objet d'un registre IANA <https://www.iana.org/assignments/dane-parameters/dane-parameters.xml#selectors>. Actuellement, il compte deux valeurs numériques. D'autres pourront être ajoutées, sous la seule obligation d'une spécification écrite et stable :

— 0 : l'enregistrement TLSA contient (dans son champ « Données ») le certificat complet.
— 1 : l'enregistrement TLSA ne contient que la clé publique, omettant le reste du certificat.

Et la méthode de correspondance ? Voici un nouveau registre IANA <https://www.iana.org/assignments/dane-parameters/dane-parameters.xml#matching-types>, et ses valeurs :

— 0 : la valeur dans le champ « Données » de l'enregistrement TLSA est la valeur exacte, à comparer bit à bit avec celle récupérée dans la session TLS.
— 1 : la valeur dans le champ « Données » de l'enregistrement TLSA est le condensat SHA-256 de celle récupérée dans la session TLS.
— 2 : idem, avec SHA-512.

Enfin, le champ de données contient, selon la valeur des autres champs, le certificat ou la clé publique, éventuellement condensées.

Voici des exemples d'enregistrement TLSA au format de présentation standard :

```
; Utilisation 0 (contrainte sur l'AC)
; Les données sont le condensat du certificat
_443._tcp.www.example.com. IN TLSA (
      0 0 1 d2abde240d7cd3ee6b4b28c54df034b9
            7983a1d16e8a410e4561cb106618e971 )

; Utilisation 1 (contrainte sur le certificat)
; Les données sont le condensat de la clé publique
_443._tcp.www.example.com. IN TLSA (
      1 1 2 92003ba34942dc74152e2f2c408d29ec
            a5a520e7f2e06bb944f4dca346baf63c
            1b177615d466f6c4b71c216a50292bd5
            8c9ebdd2f74e38fe51ffd48c43326cbc )

; Utilisation 3 (déclaration d'un certificat local)
; Le certificat entier (abrégé ici) est dans les données
_443._tcp.www.example.com. IN TLSA (
      3 0 0 30820307308201efa003020102020... )
```

Mais quel est ce nom bizarre dans les enregistrements, _443._tcp.www.example.com ? Décrit en section 3, le nom de domaine des enregistrements TLSA est la concaténation du numéro de port d'écoute du serveur TLS, du protocole (TCP, SCTP, etc) et du nom du serveur TLS. Ici, les enregistrements TLSA concernaient un serveur TLS www.example.com écoutant en TCP sur le port 443, donc, sans doute du HTTPS. Avec du SMTP sur TLS, ce serait sans doute _25._tcp.mail.example.com.

À noter que cette convention de nommage est générique (pour tous les protocoles) mais qu'il est prévu que, dans le futur, d'autres conventions soient adoptées, par exemple pour des protocoles où la relation entre service et serveur est plus complexe que dans le cas d'HTTP.

Une fois ces enregistrements publiés, comment le client TLS les utilise-t-il ? La section 4 décrit les règles à suivre. D'abord, le client TLS doit demander les enregistrements TLSA et vérifier s'ils sont **utilisables** :

— s'ils sont signés avec DNSSEC et valides, c'est bon,
— même signés correctement, si les enregistrements TLSA contiennent une utilisation, un sélecteur ou une méthode de correspondances inconnues, ils doivent être ignorés,
— s'ils ne sont pas signés, ils doivent être ignorés (on ne peut pas leur faire confiance), on se rabat sur du TLS classique. Même chose si aucun des enregistrements TLSA n'est utilisable,
— s'ils sont signés et invalides, cela signifie qu'un gros problème est en cours, peut-être une tentative d'empoisonnement DNS. Le client TLS doit alors avorter sa connexion.

Le but de ces règles est de s'assurer qu'un attaquant ne peut pas effectuer une attaque par repli, où il convaincrait le client qu'il n'y a pas d'enregistrements `TLSA` (RFC 6394, section 4). DNSSEC empêchera ce repli et DANE coupera alors la connexion TLS.

Pour les gens qui préfèrent le pseudo-code, les mêmes règles sont disponibles sous cette forme dans l'annexe B. Attention, c'est le texte de la section 4 qui est normatif, le pseudo-code n'étant là qu'à titre d'illustration. Par exemple, comme tous les pseudo-codes procéduraux, il « sur-spécifie » en indiquant un ordre d'évaluation qui n'est pas obligatoire.

Si le client TLS fait la validation DNSSEC lui-même (comme le permettent des bibliothèques comme libunbound <`http://www.unbound.net/documentation/libunbound-tutorial-6.html`>), rien de spécial à dire. Mais s'il confie la validation à un résolveur externe (et se contente de tester que le bit `ad` - "*Authentic Data*" - est bien mis à un dans la réponse), il doit s'assurer que le chemin avec ce résolveur est sûr : résolveur sur `localhost`, ou bien communication avec le résolveur sécurisée par TSIG (RFC 2845) ou IPsec (RFC 6071). C'est le problème dit de « la sécurité du dernier kilomètre ».

Comme vous voyez, le principe est très simple. Mais, comme souvent en sécurité, il y a plein de détails pièges derrière. La section 8 se consacre à l'examen de ces pièges. D'abord, DANE dépend de DNSSEC (RFC 4033). Si on n'a pas de DNSSEC, ou s'il est mal géré, tout s'écroule. Pas question d'envisager de déployer DANE avant d'avoir une configuration DNSSEC solide. (Au début du groupe de travail DANE à l'IETF, il était prévu que DNSSEC soit facultatif pour des utilisations comme 0 et 1, où DANE ne vient qu'en addition de X.509. Techniquement, ça se tenait mais cela compliquait l'analyse de sécurité, et les messages à faire passer. Donc, désormais, DNSSEC est obligatoire et on ne discute pas !)

DANE dépendant du DNS, cela veut dire qu'un administrateur DNS qui devient fou ou méchant peut changer les `A`, `AAAA` et `TLSA` d'un serveur et sans que cela soit détecté. Beaucoup d'AC émettant des certificats juste après avoir vérifié un échange de courrier, cela ne change donc pas grand'chose par rapport à X.509.

Par contre, si jamais le mécanisme de sécurité pour changer les `TLSA` était plus faible que celui pour changer les `A` et `AAAA`, alors, là, il y aurait un problème. La solution est simple : tous les enregistrements DNS doivent avoir le même niveau de sécurité.

Les enregistrements `TLSA` d'utilisation 0 et 1 viennent en supplément de X.509. Leur analyse de sécurité est donc relativement simple, ils ne peuvent pas affaiblir la sécurité qui existe. En revanche, ceux d'utilisation 2 et 3 sont nouveaux : ils peuvent permettre des choses qui n'existaient pas en X.509. Il faut donc bien réfléchir à l'AC qu'on indique par un certificat d'utilisation 2. En mettant son certificat dans un enregistrement `TLSA`, on lui donne tous les droits.

Comment fait-on une révocation avec DANE ? Les TTL normaux du DNS s'appliquent mais, si un méchant fait de l'empoisonnement DNS, il pourra réinjecter de vieux enregistrements. La seule protection est donc la durée de validité des signatures DNSSEC : trop courte, elle peut entraîner des risques pour la résilience de la zone (signatures expirées alors qu'on n'a pas eu le temps de les refaire), trop longue, elle permet à un méchant de continuer à utiliser un certificat normalement révoqué (attaque par rejeu, voir l'article de Florian Maury <`https://x-cli.eu/secfiles/pki.pdf`>).

Au fait, petit piège avec les TTL : si tous les serveurs résolveurs gèrent les TTL proprement, il est courant que le résultat d'une requête DNS soit gardé dans l'application sans tenir compte de ces TTL, ce qui peut être gênant pour des applications qui restent ouvertes longtemps (comme un navigateur Web). Un client DANE doit donc veiller à toujours obtenir ses données d'un résolveur (ou à gérer les TTL lui-même) pour ne pas risquer de se servir de vieilles données (section 8.2).

La section 8.1 compare DANE aux AC classiques. Un registre de noms de domaines qui fait du DNSSEC a des responsabilités qui ressemblent assez à celles d'une AC : distribuer la clé publique (qui doit être toujours disponible) de manière authentifiée, garder la clé privée vraiment secrète (par exemple dans des HSM), avoir des plans prêts d'avance pour le cas d'une compromission de la clé, etc. D'abord, une évidence : les acteurs de l'industrie des noms de domaine (registres, "*registrars*", hébergeurs DNS) ne sont pas meilleurs (ou pires) que les AC X.509. Comme elles, ils peuvent être incompétents ou malhonnêtes ou se faire pirater. On peut même penser que certains de ces acteurs ont une culture sécurité faible ou inexistante, très inférieure à celle de la pire AC. Mais la force de DANE ne vient pas de ce que les acteurs du DNS seraient magiquement meilleurs que ceux de X.509. Elle vient du fait qu'on choisit ses fournisseurs. Avec X.509, c'est la sécurité de la plus mauvaise AC qui compte puisque n'importe quelle AC peut émettre des certificats pour n'importe qui (**la** faille énorme de X.509). Avec DANE+DNSSEC, on revient à un modèle plus sérieux en matière de sécurité : il faut faire confiance à ses fournisseurs, certes, mais au moins on les choisit. Un déploiement correct de DANE implique donc une évaluation des acteurs qu'on a choisi. Si on a un `.com`, on dépend de la sécurité et de l'honnêteté de VeriSign. En revanche, un piratage du registre de `.lu` n'aura pas de conséquence. Si on choisit à la place un `.lu`, on dépend de la sécurité et de l'honnêteté de Restena <`http://www.dns.lu/`> mais plus de celle de VeriSign. Et ainsi de suite. (Tout ceci est également vrai pour les autres techniques de pubication de clés dans le DNS comme celles du RFC 4255.)

Le DNS étant hiérarchique, il faut ajouter à ses fournisseurs directement choisis, les registres situés « au dessus ». Ainsi, une compromission de la clé privée de la racine affecterait **tous** les domaines (avec X.509, c'est la compromission de n'importe laquelle des centaines d'AC qui aurait ce résultat), ce qui justifie les mesures coûteuses adoptées pour gérer la clé de la racine (meilleures que celles de la plupart des AC X.509).

Comme indiqué plus haut à propos de la sécurité du dernier kilomètre, un problème classique avec DNSSEC est le lien entre l'application cliente et le résolveur validant. Le RFC recommande (section 8.3), pour une sécurité correcte, que le résolveur soit sur la même machine (c'est très facile avec des logiciels comme dnssec-trigger <`http://www.bortzmeyer.org/dnssec-trigger.html`>). Aujourd'hui où le moindre téléphone sait jouer des films en haute résolution, le temps de calcul lié à la validation cryptographique n'est plus forcément un obstacle. La section A.3 revient en détail sur ce problème du dernier kilomètre.

Les gens qui aiment les problèmes opérationnels liront avec plaisir l'annexe A, qui rassemble plein de bons conseils sur le déploiement de DANE.

Maintenant, voyons les mises en œuvre existantes de DANE/TLSA. Il faut bien voir que ce mécanisme est encore récent. On ne trouve que peu d'enregistrements TLSA dans la nature, et aucun navigateur Web connu ne gère encore le TLSA standard (DANE ne nécessite par contre aucune modification du serveur Web, uniquement du client Web, et des serveurs DNS). Néanmoins, il existe plusieurs outils.

D'abord, côté DNS. BIND et NSD peuvent tous les deux être serveur maître pour une zone qui contient des enregistrements TLSA (nsd connait DANE/TLSA à partir de la version 3.2.11). Pour un serveur esclave, pas de problème, TLSA est juste un type d'enregistrement inconnu, que le serveur peut servir sans le comprendre. Pour valider ses zones DNS, l'excellent outil validns <`https://github.com/tobez/validns/`> lit et vérifie le TLSA depuis juillet 2012 <`https://github.com/tobez/validns/commit/117db4994fdeadca6798a8a399ffb0364a11dd42`>.

D'accord, mais un être humain normal ne va pas fabriquer les enregistrements TLSA à la main. Et, même s'il le faisait, il serait souhaitable de vérifier leur cohérence avec le certificat présenté par le serveur HTTP. C'est le rôle de l'outil Swede <`https://github.com/pieterlexis/swede`>. Il permet, par exemple, de créer l'enregistrement TLSA à partir du certificat actuellement distribué par le serveur HTTPS :

```
% ./swede create www.afnic.fr
No certificate specified on the commandline, attempting to retrieve it from the server www.afnic.fr.
Attempting to get certificate from 192.134.4.20
Got a certificate with Subject: /1.3.6.1.4.1.311.60.2.1.3=FR/1.3.6.1.4.1.311.60.2.1.2=Ile de France/1.3.6.1.4.1.
_443._tcp.www.afnic.fr. IN TYPE52 \# 35 010001ed8a70a9c8783777e463232ef343aa32f2bbd6aa0182e6a0fc929b64c8e06518
Attempting to get certificate from 2001:67c:2218:2::4:20
Got a certificate with Subject: /1.3.6.1.4.1.311.60.2.1.3=FR/1.3.6.1.4.1.311.60.2.1.2=Ile de France/1.3.6.1.4.1.
_443._tcp.www.afnic.fr. IN TYPE52 \# 35 010001ed8a70a9c8783777e463232ef343aa32f2bbd6aa0182e6a0fc929b64c8e06518
```

[Le certificat est le même, qu'on accède au serveur en IPv4 ou en IPv6 mais Swede ne le sait pas à l'avance, d'où les deux connexions.] Notez aussi que Swede formatte l'enregistrement DNS comme un type inconnu, ce qui permet de le charger dans n'importe quel serveur. Il ne reste donc qu'à mettre le `_443._tcp.www.afnic.fr. IN TYPE52 \# 35 010001ed8a70a9c8783777e463232ef343aa32f2bbd6aa0182e` dans son fichier de zone.

Swede permet également de vérifier un enregistrement TLSA existant.

Maintenant, côté « divers », la bibliothèque NSS <http://www.mozilla.org/projects/security/pki/nss/> a un "*patch*" pour faire du TLSA mais il semble ancien en non maintenu : <https://mattmccutchen.net/cryptid/#nss-dane>.

L'outil sshfp <https://github.com/xelerance/sshfp>, en dépit de son nom, fait aussi du TLSA. Il devrait être remplacé par hash-slinger <http://people.redhat.com/pwouters/hash-slinger/>.

Quelques petits trucs utiles, maintenant. Pour créer une clé à partir d'un certificat (afin d'utiliser le sélecteur 1), en Python, avec M2Crypto <http://www.bortzmeyer.org/python-crypto.html> :

```
from M2Crypto import X509
cert = X509.load_cert('/path/to/cert')
print cert.get_pubkey().as_der()
print cert.get_pubkey().as_text()
```

Avec OpenSSL, pour un sélecteur 1 (uniquement la clé) `openssl x509 -pubkey -in ... | openssl rsa -pubin | head -n -1 | tail -n +2 | base64 -d` doit marcher (ajouter `sha512sum -` à la fin du tube pour condenser, méthode de correspondance 2). Pour l'extraire d'un serveur TLS existant :

```
% openssl s_client -showcerts -connect mattmccutchen.net:443 < /dev/null 2> /dev/null | \
        openssl x509 -pubkey | openssl rsa -pubin | \
        head -n -1 | tail -n +2 | base64 -d | sha256sum -
writing RSA key
62d5414cd1cc657e3d30ea5e6d0136e92306e725413c616a51cab4b852c70a1c  -
```

On trouve aujourd'hui des enregistrements `TLSA` dans de nombreux domaines <http://www.internetsociety.org/deploy360/resources/dane-test-sites/>. Voici quelques exemples avec le dig de BIND 9.9.1-P2 (sorti en juillet 2012). Notez le bit `ad` mais rappelez-vous qu'on ne peut s'y fier que si le résolveur est honnête **et** que la liaison avec le résolveur est protégée (même machine, ou bien TSIG, IPsec, tunnel TLS ou méthode équivalente) :

```
% dig TLSA _443._tcp.dane.rd.nic.fr
...
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 5
...
;; ANSWER SECTION:
_443._tcp.dane.rd.nic.fr. 1     IN      TLSA    3 0 1 63C43B664AB2EC24E5F65BF6671EE1BE48E6F12660FEEF9D7BB1ED
_443._tcp.dane.rd.nic.fr. 1     IN      RRSIG   TLSA 5 6 1 20120810085609 20120711085609 24765 dane.rd.nic.f

...

% dig  TLSA  _5269._tcp.proxima.lp0.eu
...
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 4, AUTHORITY: 5, ADDITIONAL: 5
...
;; ANSWER SECTION:
_5269._tcp.proxima.lp0.eu. 10800 IN     CNAME   _tlsa.proxima.lp0.eu.
_5269._tcp.proxima.lp0.eu. 10800 IN     RRSIG   CNAME 10 5 10800 20120920174820 20120718212409 42615 lp0.eu.
_tlsa.proxima.lp0.eu.   3600    IN      TLSA    3 1 2 490D884C778E9031D8F1BDFB4B6E7673418BAD66CB8115E36CED91
_tlsa.proxima.lp0.eu.   3600    IN      RRSIG   TLSA 10 4 3600 20120920183558 20120719092413 42615 lp0.eu. t
```

À noter la section A.2.1.1 pour l'utilisation des alias avec DANE, comme dans ce dernier cas.

```
% dig  TLSA  _443._tcp.ulthar.us
...
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 5, ADDITIONAL: 5
...
;; ANSWER SECTION:
_443._tcp.ulthar.us.    3600    IN      TLSA    3 1 1 7EB3A92457701C1A99742E902337BAE786B656ABAF1B13DFE3616D
_443._tcp.ulthar.us.    3600    IN      RRSIG   TLSA 8 4 3600 20120812104422 20120713104422 8550 ulthar.us.

...

% dig  TLSA  _443._tcp.nohats.ca
...
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 5, ADDITIONAL: 7
...
;; ANSWER SECTION:
_443._tcp.nohats.ca.    3600    IN      TLSA    1 0 1 6BCFF9A283336DD1ED99A9C40427741B5658863BD54F0A876A2BC4
_443._tcp.nohats.ca.    3600    IN      RRSIG   TLSA 8 4 3600 20120730210435 20120716182008 28758 nohats.ca.
```

On peut vérifier le `TLSA` avec OpenSSL. Ici, sur `mattmccutchen.net` :

```
% dig +short TLSA _443._tcp.mattmccutchen.net
3 0 1 F9A682126F2E90E080A3A621134523C6DE4475C632A00CF773E6C812 A321BB25
```

Utilisation 3 (donc le serveur TLS doit envoyer le certificat correspondant) avec sélection 0 (tout le certificat) condensation en SHA-256. Testons :

```
% openssl s_client -showcerts  -connect mattmccutchen.net:443 < /dev/null 2> /dev/null | \
      openssl x509 -outform der | sha256sum -
f9a682126f2e90e080a3a621134523c6de4475c632a00cf773e6c812a321bb25  -
```

et ça a marché, on trouve la même valeur.

Je n'ai pas encore trouvé de vrai enregistrement TLSA pour d'autres protocoles que HTTP. C'est logique, puisque ceux-ci impliquent en général une indirection <http://www.bortzmeyer.org/redirection-certificat.html> et sont donc plus complexes.

Enfin, si vous voulez un serveur HTTPS où DANE ne va **pas** marcher, `rogue.nohats.ca` a un enregistrement TLSA délibérement incorrect.

Autres articles sur DANE/TLSA :

— Mon article à JRES 2011 <http://www.bortzmeyer.org/jres-dane-2011.html> (ancien, mais plus long et explique plus en détail les motivations pour DANE)