RFC 7250 : Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 juillet 2014

Date de publication du RFC : Juin 2014

https://www.bortzmeyer.org/7250.html

Le système de sécurité TLS, normalisé dans le RFC 5246 [1] est souvent confondu avec la norme de certificats X.509. On lit souvent des choses comme « le serveur TLS envoie alors un certificat X.509 au client », voire, pire, on appelle ces certificats « certificats TLS » (ou, encore plus inexact, « certificats SSL »). Dans la réalité, les deux systèmes, TLS et X.509, sont indépendants. X.509 peut servir à autre chose qu'à TLS (et c'est pourquoi parler de « certificats SSL » est une grosse erreur), et TLS peut utiliser d'autres techniques d'authentification que X.509. Une de ces techniques était déjà spécifiée dans le RFC 6091, l'utilisation de clés PGP. Une autre vient d'être normalisée dans ce RFC, l'utilisation de « clés brutes » ("*raw keys*"), sans certificat autour. Comme une clé seule ne peut pas être authentifiée, ces clés brutes n'ont de sens que combinées avec un mécanisme d'authentification externe, comme DANE.

Rappelons en effet qu'un certificat, c'est une clé publique (champ `SubjectPublicKeyInfo` dans X.509) plus un certain nombre de métadonnées (comme la date d'expiration) et une signature par une autorité qui certifie la clé. Pour la session TLS, la seule partie indispensable est la clé (dans un certificat « auto-signé », seule la clé a un sens, le reste du certificat ne pouvant pas être vérifié). Mais, en pratique, le chiffrement sans l'authentification est limité : il ne protège que contre un attaquant purement passif. Dès que l'attaquant est actif (pensons par exemple à un hotspot Wifi tentant... mais tenu par l'attaquant), il peut rediriger le client TLS vers un serveur intermédiaire qui va faire croire au client qu'il est le vrai serveur, avant de retransmettre vers le serveur authentique. Pour contrer cette attaque de l'homme du milieu, on **authentifie** le serveur. Sur l'Internet, la méthode la plus courante est l'utilisation de certificats X.509, ou plus précisement de leur profil PKIX (RFC 5280). Cette méthode est d'une fiabilité très douteuse : des centaines d'organisations dans le monde peuvent émettre un faux certificat pour gmail.com
<http://www.01net.com/editorial/610140/comment-le-ministere-des-finances-espionne-le-traf

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

> et il ne peut donc pas y avoir de confiance rationnelle dans ce système (cf. « "*New Tricks for Defeating SSL in Practice*" <http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf> »).

Il existe d'autres méthodes d'authentification que X.509, permettant de faire un lien entre le serveur qu'on veut contacter, et l'entité qui nous envoie une clé publique en TLS :

— Les certificats ou clés publiés dans le DNS et sécurisés par DNSSEC, à savoir la technique DANE <https://www.bortzmeyer.org/jres-dane-2011.html> du RFC 6698,
— Les certificats ou clés publiés dans un annuaire LDAP,
— Les certificats ou clés stockés en dur chez le client. Cela manque de souplesse (il est difficile de les changer) et ce n'est donc pas très recommandé pour l'Internet public mais c'est envisagé pour des déploiements plus fermés comme ceux du protocole CoAP (RFC 7252), où les objets peuvent sécuriser la communication avec DTLS. (Cette troisième méthode est sans doute la seule possible pour l'authentification du client TLS par le serveur, cf. section 4.3.)

La section 6 de notre RFC revient en détail sur l'authentification de ces clés brutes. Sans cette authentification, une session TLS utilisant ces clés est très vulnérable à l'homme du milieu.

La section 3 décrit les détails de format des clés brutes. Il faut deux extensions au protocole TLS, `client_certificate_type` et `server_certificate_type` (enregistrées dans le registre des extensions <https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-va
xhtml#tls-extensiontype-values-1>), qui permettent d'indiquer (lors de la négociation TLS) séparément le type de certificat utilisé par le serveur et par le client. Par exemple, pour le serveur, cela s'écrit ainsi :

```
struct {
           select(ClientOrServerExtension) {
               case client:
                 CertificateType server_certificate_types<1..2^8-1>;
               case server:
                 CertificateType server_certificate_type;
           }
   } ServerCertTypeExtension;
```

Lorsque ces extensions TLS sont utilisées, la structure `Certificate` est modifiée ainsi :

```
struct {
       select(certificate_type){

           // certificate type defined in this document.
           case RawPublicKey:
             opaque ASN.1_subjectPublicKeyInfo<1..2^24-1>;

           // X.509 certificate defined in RFC 5246
           case X.509:
             ASN.1Cert certificate_list<0..2^24-1>;

           // Additional certificate type based on TLS
           //   Certificate Type Registry
           //   https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml#tls
           // for instance PGP keys
       };
   } Certificate;
```

(On notera que le RFC 6091 utilisait un autre mécanisme, l'extension `cert_type`, numéro 9, mais qui n'a pas été réutilisée ici, le RFC 6091 n'ayant pas le statut « Chemin des Normes » <http://www.ietf.org/mail-archive/web/tls/current/msg08726.html> et le mécanisme `cert_type` ne permettant pas des types différents pour le client et le serveur <http://www.ietf.org/mail-archive/web/tls/current/msg08733.html>.) Le champ `subjectPublicKeyInfo` contient un encodage DER de la clé et d'un identificateur d'algorithme (un OID comme 1.2.840.10045.2.1 pour ECDSA) :

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
      algorithm               AlgorithmIdentifier,
      subjectPublicKey        BIT STRING  }
```

L'omission d'une grande partie du certificat permet de se contenter d'un analyseur ASN.1 plus réduit. Mais les clés brutes ne sont malheureusement pas réellement brutes, c'est quand même une structure décrite en ASN.1 et il faut donc un analyseur minimal.

La section 4 décrit les modifications à la négociation initiale de TLS (les autres parties de la communication ne sont pas modifiées par ce RFC). Le client ajoute une ou deux des extensions TLS (la plupart du temps, en TLS, le client ne s'authentifie pas, et on n'aura donc que l'extension `server_certificate_type`). La ou les extensions indiquent les types de certificat reconnus par le pair TLS (si l'extension est absente, seul X.509 est reconu). Ainsi, un serveur nouveau, intégrant les clés brutes, sait tout de suite si un client pourrait les traiter (si le client n'a pas mis `server_certificate_type`, ou bien si son `server_certificate_type` n'inclut pas les clés brutes, ce n'est pas la peine d'en envoyer). Si c'est le client qui est nouveau et le serveur ancien, la réponse du serveur ne contiendra pas les extensions et le client saura donc qu'il ne faut pas s'attendre aux clés brutes. Si le serveur reconnait l'extension, mais aucun des types de certificats indiqués, il termine la session. À noter que ces clés brutes marchent avec tous les algorithmes de cryptographie déjà reconnus.

La section 5 du RFC fournit des exemples de sessions TLS avec cette nouvelle possibilité. Par exemple, ce dialogue montre un client qui gère les clés brutes du serveur (et uniquement celles-ci, ce n'est donc pas un navigateur Web typique) et un serveur qui accepte d'envoyer sa clé ainsi. Le client n'est pas authentifié (il n'envoie pas sa clé ou son certificat et ne se sert donc pas de `client_certificate_type`) :

```
client_hello,
server_certificate_type=(RawPublicKey) // [1]
                        ->
                        <- server_hello,
                           server_certificate_type=(RawPublicKey), // [2]
                           certificate, // [3]
                           server_key_exchange,
                           server_hello_done

client_key_exchange,
change_cipher_spec,
finished                ->

                        <- change_cipher_spec,
                           finished

Application Data        <-------> Application Data
```

Ce cas d'un client ne comprenant que les clés brutes est un cas qui peut se produire pour l'Internet des objets, par exemple avec CoAP. Mais un client différent, par exemple un navigateur Web, va accepter d'autres types de certificats. Le dialogue pourrait ressembler à :

```
client_hello,
server_certificate_type=(X.509, RawPublicKey)
client_certificate_type=(RawPublicKey) // [1]
                         ->
                         <-  server_hello,
                             server_certificate_type=(X.509)//[2]
                             certificate, // [3]
                             client_certificate_type=(RawPublicKey)//[4]
                             certificate_request, // [5]
                             server_key_exchange,
                             server_hello_done
certificate, // [6]
client_key_exchange,
change_cipher_spec,
finished                  ->

                          <- change_cipher_spec,
                             finished

Application Data        <------->     Application Data
```

Ici, le serveur reconnait la nouvelle extension mais ne sait envoyer que des certificats X.509 (ce qui est donc fait par la suite). Le client, lui, s'authentifie en envoyant une clé brute (même si ce message est marqué `certificate` dans le dialogue).

Il existe apparemment déjà plusieurs mises en œuvre de TLS qui utilisent ces extensions, par exemple GnuTLS.