RFC 6347 : Datagram Transport Layer Security version 1.2

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 21 janvier 2012

Date de publication du RFC : Janvier 2012

https://www.bortzmeyer.org/6347.html

Le protocole de cryptographie TLS, normalisé dans le RFC 5246 [1], ne s'appliquait traditionnellement qu'à TCP. Les applications utilisant UDP, comme le fait souvent la téléphonie sur IP, ne pouvaient pas utiliser TLS pour protéger leur trafic contre l'écoute ou la modification des données. Mais cette limitation a disparu avec DTLS, qui permet de protéger du trafic UDP. Ce RFC met à jour DTLS (initialement normalisé dans le RFC 4347) pour TLS 1.2. (Depuis, la version 1.3 est sortie, dans le RFC 9147.)

TLS ne tournait que sur TCP car il avait besoin d'un transport fiable, garantissant que les données arrivent toutes, et dans l'ordre. Le grand succès de TLS (notamment utilisé pour HTTP et IMAP) vient de sa simplicité pour le programmeur : rendre une application capable de faire du TLS ne nécessite que très peu de code, exécuté juste avant l'envoi des données. Par contre, cela laissait les applications UDP comme SIP non protégées (section 1 de notre RFC). Les solutions existantes comme IPsec ne sont pas satisfaisantes (cf. RFC 5406), notamment parce qu'elles n'offrent pas la même facilité de déploiement que TLS, qui tourne typiquement dans l'application et pas dans le noyau.

DTLS a été conçu pour fournir TLS aux applications UDP. Il offre les mêmes services que TLS : garantie de l'intégrité des données et confidentialité.

La section 3 explique les principes de DTLS : protégé ou pas par DTLS, UDP a la même sémantique, celle d'un service de datagrammes non fiable. D'autre part, DTLS est une légère modification de TLS : il en garde les principales propriétés, bonnes ou mauvaises.

Mais pourquoi ne peut-on pas faire du TLS normal sur UDP ? Parce que TLS n'a pas été conçu pour tourner au dessus d'un protocole non fiable. TLS organise les données en enregistrements ("*records*") et

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

il ne permet pas de déchiffrer indépendamment les enregistrements. Si l'enregistrement N est perdu, le N+1 ne peut pas être déchiffré. De même, la procédure d'association initiale de TLS ("*handshake*") ne prévoit pas de perte de messages et ne se termine pas si un message est perdu.

Le premier problème fait l'objet de la section 3.1. La dépendance des enregistrements TLS vis-à-vis de leurs prédécesseurs vient du chaînage cryptographique (le chiffrement par chaînage - "*stream cipher*" - est donc supprimé en DTLS) et de fonctions anti-rejeu qui utilisent un numéro de séquence, qui est implicitement le rang de l'enregistrement. DTLS résout le problème en indiquant explicitement le rang dans les enregistrements.

Et la question de l'association initiale est vue dans la section 3.2. Pour la perte de paquets lors de l'association, DTLS utilise un système de retransmission (section 3.2.1) et pour l'éventuelle réorganisation des paquets, DTLS introduit un numéro de séquence (section 3.2.2). En prime, DTLS doit gérer la taille importante des messages TLS (souvent plusieurs kilo-octets), qui peut être supérieure à la MTU. DTLS permet donc une fragmentation des paquets au niveau applicatif, un message pouvant être réparti dans plusieurs enregistrements (section 3.2.3).

Enfin, l'anti-rejeu a été modifié pour tenir compte du fait que la duplication de paquets, en UDP, n'est pas forcément malveillante (sections 3.3 et 4.1.2.6).

La définition formelle du nouveau protocole est en section 4. DTLS étant une légère évolution de TLS, la définition se fait uniquement en listant les différences avec TLS. Il faut donc garder le RFC 5246 sous la main.

Dans le "*Record Protocol*" de TLS, l'enregistrement spécifié dans la section 6.2.1 du RFC 5246 gagne deux champs (section 4.1) :

```
struct {
       ContentType type;
       ProtocolVersion version;
       uint16 epoch;                                  // NOUVEAU
       uint48 sequence_number;                        // NOUVEAU
       uint16 length;
       opaque fragment[DTLSPlaintext.length];
     } DTLSPlaintext;
```

notamment le numéro de séquence `sequence_number` (qui était implicite dans TLS, puisque TCP garantissait l'ordre des messages). Pour éviter la fragmentation et les ennuis associés <https://www.bortzmeyer.org/mtu-et-mss-sont-dans-un-reseau.html>, les mises en œuvre de DTLS doivent déterminer la MTU du chemin et n'envoyer que des enregistrements plus petits que cette MTU (section 4.1.1).

Contrairement à IPsec, DTLS n'a pas la notion d'identificateur d'association. Une machine qui reçoit du TLS doit trouver l'association toute seule, typiquement en utilisant le tuple (adresse IP, port).

En toute rigueur, DTLS n'est pas spécifique à UDP, il peut marcher sur n'importe quel protocole de transport ayant une sémantique « datagrammes ». Certains de ces protocoles, comme DCCP (cf. RFC 5238), ont leur propres numéros de séquence et ils font donc double emploi avec ceux de DTLS. Petite inefficacité pas trop grave.

Au niveau du "*Hanshake protocol*", les modifications que DTLS apporte à TLS font l'objet de la section 4.2. Les trois principales sont :

— L'ajout d'un gâteau pour limiter les risques de DoS,
— Modification des en-têtes pour gérer les pertes ou réordonnancements des paquets (section 4.2.2),
— Ajouts de minuteries pour détecter les pertes de paquets (section 4.2.4).

Les gâteaux de DTLS sont analogues à ceux de Photuris ou IKE (section 4.2.1). Le message `ClientHello` de la section 7.4.1.2 du RFC 5246 y gagne un champ :

```
opaque cookie<0..32>;          // NOUVEAU
```

Évidemment, ils ne protègent pas contre un attaquant qui utilise sa vraie adresse IP, puisque celui-ci pourra lire la réponse.

OpenSSL gère DTLS depuis la version 0.9.8 (on peut aussi consulter le site Web du développeur `<http://crypto.stanford.edu/~nagendra/projects/dtls/>`). Un exemple d'utilisation se trouve dans `<http://linux.softpedia.com/get/Security/DTLS-Client-Server-Example-19026.shtml>`. Il me reste à inclure ce protocole dans echoping `<http://echoping.sourceforge.net/>`. GnuTLS a un support DTLS plus récent.

Et les nouveautés de DTLS 1.2 par rapport au 1.0 du RFC 4347 ? (Il n'y a pas eu de DTLS 1.1.) Elles sont résumées dans la section 8. La principale nouveauté est que DTLS est désormais défini par rapport à TLS 1.2 et non plus 1.0. Cela a notamment permis d'inclure les numéros de séquence explicites de TLS 1.1, nécessaires contre l'attaque BEAST `<https://www.bortzmeyer.org/beast-tls.html>` (section 4.1.2.1). Une autre nouveauté est la section 4.1.2.7 qui discute le traitement des paquets invalides. Contrairement à TLS, DTLS peut fonctionner en présence de tels paquets, avec une sémantique proche de celle d'UDP : les laisser tomber et attendre que l'application se débrouille (en demandant leur réémission ou bien en passant à autre chose). Enfin, d'autres clarifications ont été apportées, par exemple à la détection de la PMTU ou bien à l'enregistrement dans les registres IANA.

Une très bonne description de la conception de "*Datagram TLS*" et des choix qui ont été faits lors de sa mise au point, se trouve dans l'article "*The Design and Implementation of Datagram TLS*" `<http://crypto.stanford.edu/~nagendra/papers/dtls.pdf>`, écrit par les auteurs du RFC. C'est une lecture très recommandée.