Groupe de travail Réseau
Request for Comments : 4347

Catégorie : Sur la voie de la normalisation

E. Rescorla, RTFM, Inc.
N. Modadugu, Stanford University
avril 2006
Traduction Claude Brière de L'Isle

Sécurité de la couche transport de datagramme

Résumé
Le présent document spécifie la version 1.0 du protocole de sécurité de couche transport de datagramme (DTLS, *Datagram Transport Layer Security*). Le protocole DTLS assure la confidentialité des communications pour les protocoles de datagrammes. Le protocole permet aux applications de client/serveur de communiquer d'une façon conçue pour empêcher l'espionnage, l'altération ou la falsification du message. Le protocole DTLS se fonde sur le protocole de sécurité de la couche transport (TLS, *Transport Layer Security*) et fournit des garanties de sécurité équivalentes. La sémantique de datagramme du transport sous-jacent est préservée par le protocole DTLS.

Table des matières

1. Introduction

TLS [RFC2246] est le protocole le plus largement déployé pour sécuriser le trafic du réseau. Il est largement utilisé pour protéger le trafic de la Toile et de protocoles de messagerie électronique comme IMAP [RFC3501] et POP [RFC1939]. Le principal avantage de TLS est qu'il fournit un canal transparent en mode connexion. Donc, il est aisé de sécuriser un protocole d'application en insérant TLS entre la couche d'application et la couche transport. Cependant, TLS doit fonctionner sur un canal de transport fiable -- normalement TCP [RFC0793]. Il ne peut donc pas être utilisé pour sécuriser du trafic de datagrammes non fiable.

Cependant, au cours des dernières années un nombre croissant de protocoles de couche application ont été conçus pour utiliser le transport UDP. En particulier des protocoles comme le protocole d'initialisation de session (SIP, *Session*

Initiation Protocol) [RFC3261] et les protocoles de jeu électronique sont de plus en plus populaires. (Noter que SIP peut fonctionner sur TCP et sur UDP, mais il y a des situations où UDP est préférable). Actuellement, les concepteurs de ces applications font face à un certain nombre de choix non satisfaisants. D'abord, ils peuvent utiliser IPsec [RFC2401]. Cependant, pour un certain nombre de raisons détaillées dans la [RFC5406], ceci ne convient que pour certaines applications. Ensuite, ils peuvent concevoir un protocole personnalisé de sécurité de la couche d'application. SIP, par exemple, utiliser un sous ensemble de S/MIME pour sécuriser son trafic. Malheureusement, bien que les protocoles de sécurité de la couche d'application fournissent généralement des propriétés de sécurité supérieures (par exemple, la sécurité de bout en bout dans le cas de S/MIME) ils exigent normalement de gros efforts de conception – contrastant avec les efforts relativement faibles exigés pour faire fonctionner le protocole sur TLS.

Dans de nombreux cas, la façon la plus souhaitable de sécuriser les applications client/serveur va être d'utiliser TLS ; cependant l'exigence d'une sémantique de datagrammes interdit automatiquement l'usage de TLS. Donc, une variante de TLS compatible avec le datagramme serait très désirable. Le présent mémoire décrit un tel protocole : le protocole de sécurité de couche transport de datagramme (DTLS, *Datagram Transport Layer Security*). DTLS est délibérément conçu pour être aussi similaire de TLS que possible, à la fois pour minimiser l'invention d'une nouvelle sécurité et pour maximiser la réutilisation du code et de l'infrastructure.

1.1 Terminologie des exigences

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

2. Modèle d'usage

Le protocole DTLS est conçu pour sécuriser les données entre des applications communiquantes. Il est conçu pour fonctionner dans un espace d'application, sans exiger de modification du noyau.

Le transport de datagrammes n'exige pas ni ne fournit de livraison fiable ou en ordre des données. Le protocole DTLS préserve cette propriété pour les données de charge utile. Des applications comme de flux directs de supports, de téléphonie Internet, et de jeux en ligne utilisent le transport de datagrammes pour la communication du fait de la nature sensible au délai des données transportées. Le comportement de telles applications est inchangé quand le protocole DTLS est utilisé pour sécuriser la communication, car le protocole DTLS ne compense pas le trafic de données perdues ou réordonnées.

3. Vue d'ensemble de DTLS

La philosophie de la conception de base de DTLS est de construire "TLS sur datagramme". La raison pour laquelle TLS ne peut pas être utilisé directement dans les environnements de datagrammes est simplement que les paquets peuvent être perdus ou réordonnés. TLS n'a pas de facilités internes pour traiter cette sorte de non fiabilité, et donc les mises en œuvre de TLS sont défaillantes lorsque elles sont hébergées sur un transport de datagrammes. L'objet de DTLS est de faire seulement le minimum de changements à TLS nécessaire pour corriger ce problème. Dans la mesure du possible, DTLS est identique à TLS. Chaque fois qu'il est nécessaire d'inventer de nouveaux mécanismes, on tente de le faire d'une façon qui préserve le style de TLS.

La non fiabilité crée des problèmes à TLS à deux niveaux :

1. La couche de chiffrement du trafic de TLS ne permet pas un déchiffrement indépendant des enregistrements individuels. Si l'enregistrement N n'est pas reçu, l'enregistrement N+1 ne peut pas être déchiffré.

2. La couche de prise de contact TLS suppose que les messages de prise de contact sont livrés de façon fiable et casse si ces messages sont perdus.

Le reste de cette section décrit l'approche que DTLS utilise pour résoudre ces problèmes.

3.1 Échange de messages insensible à la perte

Dans la couche de chiffrement du trafic de TLS (appelée couche d'enregistrement TLS) les enregistrements ne sont pas indépendants. Il y a deux sortes de dépendance inter enregistrements :

1. Le contexte cryptographique (état CBC, flux de clé de chiffrement de flux) est enchaîné entre les enregistrements.

2. La protection anti-répétition et contre le changement d'ordre de message est fournie par un MAC qui inclut un numéro de séquence, mais les numéros de séquence sont implicites dans les enregistrements.

Le correctif pour ces deux problèmes est direct et bien connu depuis IPsec ESP [RFC2406] : l'ajout d'un état explicite aux enregistrements. TLS 1.1 [RFC4346] ajoute déjà un état CBC explicite aux enregistrements TLS. DTLS emprunte ce mécanisme et y ajoute des numéros de séquence explicites.

3.2 Assurer la fiabilité de la prise de contact

La prise de contact TLS est une prise de contact cryptographique à étapes verrouillées. Les messages doivent être transmis et reçus dans un ordre défini, et tout autre ordre est une erreur. Ceci est clairement incompatible avec le changement d'ordre et la perte de message. De plus, les messages de prise de contact TLS sont potentiellement plus gros que tout datagramme, créant donc le problème de fragmentation. DTLS doit fournir des correctifs pour ces deux problèmes.

3.2.1 Perte de paquet

DTLS utilise un simple temporisateur de retransmission pour traiter la perte de paquet. La figure suivante montre le concept de base, en utilisant la première phase de la prise de contact DTLS :

```
    Client                  Serveur
ClientHello            ------>
                  X<--- HelloVerifyRequest (perdu)
    [Le temporisateur expire]
ClientHello (retransmis)  ------>
```

Une fois que le client a transmis le message ClientHello, il s'attend à voir un HelloVerifyRequest provenant du serveur. Cependant, si le message du serveur est perdu, le client sait que soit le ClientHello, soit le HelloVerifyRequest a été perdu et il retransmet. Quand le serveur reçoit la retransmission, il sait qu'il doit retransmettre. Le serveur tient aussi un temporisateur de retransmission et retransmet quand ce temporisateur arrive à expiration.

Note : la fin de temporisation et la retransmission ne s'appliquent pas à la HelloVerifyRequest, parce que cela exige de créer un état sur le serveur.

3.2.2 Remise en ordre

Dans DTLS, chaque message de prise de contact reçoit un numéro de séquence spécifique de cette prise de contact. Quand un homologue reçoit un message de prise de contact, il peut rapidement déterminer si ce message est le prochain message qu'il attend. Si il l'est, il le traite alors. Sinon, il le met en file d'attente pour un futur traitement une fois que tous les messages précédents ont été reçus.

3.2.3 Taille de message

Les messages de prise de contact TLS et DTLS peuvent être assez gros (en théorie jusqu'à 2^24-1 octets, en pratique de nombreux kilooctets). À l'opposé, les datagrammes UDP sont souvent limités à < 1500 octets si la fragmentation n'est pas désirée. Afin de compenser cette limitation, chaque message de prise de contact DTLS peut être fragmenté sur plusieurs enregistrements DTLS. Chaque message de prise de contact DTLS contient à la fois un décalage de fragment et une longueur de fragment. Donc, un receveur en possession de tous les octets d'un message de prise de contact peut ré-assembler le message original non fragmenté.

3.3 Détection de répétition

DTLS prend facultativement en charge la détection de répétition d'enregistrement. La technique utilisée est la même que dans IPsec AH/ESP, en tenant une fenêtre de concordance de bits des enregistrements reçus. Les enregistrements qui sont trop vieux pour tenir dans la fenêtre et les enregistrements qui ont déjà été reçus sont éliminés en silence. La caractéristique de détection de répétition est facultative, car la duplication de paquet n'est pas toujours malveillante, mais peut aussi se produire à cause d'erreurs d'acheminement. Les applications peuvent concevoir de détecter les paquets dupliqués et de modifier en conséquence leur politique de transmission des données.

4. Différences avec TLS

Comme mentionné à la Section 3, DTLS est intentionnellement très similaire à TLS. Donc, au lieu de présenter DTLS comme un nouveau protocole, on le présente comme une série de différences à TLS 1.1 [RFC4346]. Lorsque on n'invoque pas explicitement de différences, DTLS est le même que dans la [RFC4346].

4.1 Couche d'enregistrement

La couche d'enregistrement DTLS est extrêmement semblable à celle de TLS 1.1. Le seul changement est l'inclusion d'un numéro de séquence explicite dans l'enregistrement. Ce numéro de séquence permet au receveur de vérifier correctement le MAC TLS. Le format d'enregistrement DTLS est montré ci-dessous :

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

type : équivalent au champ type dans un enregistrement TLS 1.1.

version : version du protocole employé. Le présent document décrit DTLS version 1.0, qui utilise la version { 254, 255 }. La valeur de version de 254.255 est le complément à 1 de DTLS version 1.0. Cet espacement maximal entre les numéros de version de TLS et DTLS assure que les enregistrements provenant des deux protocoles peuvent être aisément distingués. On notera que les futurs numéros de version dans le réseau de DTLS sont de valeur décroissante (alors que le vrai numéro de version est de valeur croissante.)

epoch : valeur de compteur qui est incrémentée à chaque changement d'état de chiffrement.

sequence_number : le numéro de séquence pour cet enregistrement.

length : Identique au champ Longueur dans un enregistrement TLS 1.1. Comme dans TLS 1.1, la longueur ne devrait pas excéder 2^14.

fragment : identique au champ Fragment d'un enregistrement TLS 1.1 .

DTLS utilise un numéro de séquence explicite, plutôt qu'implicite, porté dans le champ sequence_number de l'enregistrement. Comme avec TLS, le numéro de séquence est réglé à zéro après chaque envoi du message ChangeCipherSpec.

Si plusieurs prises de contact sont effectuées en succession rapprochée, il peut y avoir plusieurs enregistrements sur le réseau avec le même numéro de séquence mais provenant de différents états de chiffrement. Le champ epoch permet aux receveurs de distinguer de tels paquets. Le numéro de epoch est initialement zéro et est incrémenté chaque fois que des messages ChangeCipherSpec sont envoyés. Afin de s'assurer que toute paire séquence/epoch est unique, les mises en œuvre NE DOIVENT PAS permettre que la même valeur de epoch soit réutilisée dans deux fois la durée de vie maximum de segment TCP. En pratique, les mises en œuvre de TLS refont rarement des prises de contact et on ne s'attend donc pas à ce que cela pose de problème.

4.1.1 Transposition de la couche transport

Chaque enregistrement DTLS DOIT tenir dans un seul datagramme. Afin d'éviter la fragmentation IP [RFC1191], les mises en œuvre de DTLS DEVRAIENT déterminer la MTU et envoyer des enregistrements plus petits que la MTU. Les mises en œuvre de DTLS DEVRAIENT fournir un moyen pour que les applications déterminent la valeur de la PMTU (ou, autrement, la taille maximum de datagramme de l'application, qui est la PMTU moins les frais généraux de DTLS par enregistrement). Si l'application tente d'envoyer un enregistrement plus grand que la MTU, la mise en œuvre de DTLS DEVRAIT générer une erreur, évitant donc d'envoyer un paquet qui va être fragmenté.

Noter qu'à la différence de IPsec, les enregistrements DTLS ne contiennent aucun identifiant d'association. Les applications doivent s'arranger pour multiplexer entre les associations. Avec UDP, ceci est supposé être fait avec le numéro d'hôte/accès.

Plusieurs enregistrements DTLS peuvent être placés dans un seul datagramme. Ils sont simplement codés consécutivement. Le tramage d'enregistrement DTLS est suffisant pour déterminer les limites. Noter cependant, que le premier octet de la charge utile du datagramme doit être le début d'un enregistrement. Les enregistrements ne peuvent pas s'étendre sur plusieurs datagrammes.

Certains transports, comme DCCP [RFC4340] fournissent leur propre numéro de séquence. Quand ils sont portés sur ces transports, les deux numéros de séquence DTLS et de transport vont être présents. Bien que cela introduise un petit peu d'inefficacité, les numéros de séquence de couche transport et de DTLS servent des objets différents, et donc pour la simplicité conceptuelle, il est préférable d'utiliser les deux numéros de séquence. À l'avenir, des extensions à DTLS pourront être spécifiées pour permettre seulement un jeu de numéros de séquence pour des déploiements dans des environnements contraints.

Certains transports, comme DCCP, fournissent le contrôle d'encombrement pour le trafic qu'ils potent. Si la fenêtre d'encombrement est suffisamment étroite, des retransmissions de prise de contact DTLS peuvent être retenues plutôt que transmises immédiatement, conduisant potentiellement à des fins de temporisation et des retransmissions parasites. Quand DTLS est utilisé sur de tels transports, il faut veiller à ne pas surdimensionner la fenêtre d'encombrement probable. À l'avenir, une transposition DTLS-DCCP pourrait être spécifiée pour fournir un comportement optimal pour cette interaction.

4.1.1.1 Découverte de la PMTU

En général, la philosophie de DTLS est d'éviter de traiter les questions de PMTU. La politique générale est de commencer avec une MTU prudente et ensuite de la mettre à jour si des événements durant la prise de contact ou la phase réelle de transport des données d'application l'exigent.

La PMTU DEVRAIT être initialisée à partir de la MTU de l'interface qui va être utilisée pour envoyer des paquets. Si la mise en œuvre de DTLS reçoit un message ICMP "Destination injoignable" [RFC1191] avec le code "fragmentation nécessaire et DF établi" (aussi appelé "Datagramme trop gros") elle devrait diminuer son estimation de PMTU pour celle donnée dans le message ICMP. Une mise en œuvre de DTLS DEVRAIT permettre à l'application de réinitialiser occasionnellement son estimation de PMTU. La mise en œuvre de DTLS DEVRAIT aussi permettre aux applications de contrôler l'état du bit DF. Ces contrôles permettent à l'application d'effectuer la découverte de la PMTU. Les procédures de la [RFC1981] DEVRAIENT être suivies pour IPv6.

Un cas particulier est le système de prise de contact DTLS. Les messages de prise de contact devraient être envoyés avec le bit DF établi. Comme certains pare-feu et routeurs examinent les messages ICMP, il est difficile à la couche de prise de contact de distinguer une perte de paquet d'une estimation trop large de la PMTU. Afin de permettre les connexions dans ces circonstances, les mises en œuvre de DTLS DEVRAIENT sauvegarder la taille du paquet de prise de contact durant le repli de retransmission décrit au paragraphe 4.2.4. Par exemple, si un gros paquet est envoyé, après 3 retransmissions la couche de prise de contact pourrait choisir de fragmenter le message de prise de contact à la retransmission. En général, le choix d'une MTU initiale prudente va éviter ce problème.

4.1.2 Protection de la charge utile d'enregistrement

Comme TLS, DTLS transmet les données comme une série d'enregistrements protégés. Les paragraphes suivants décrivent les détails de ce format.

4.1.2.1 MAC

Le MAC DTLS est le même que celui de TLS 1.1. Cependant, plutôt que d'utiliser le numéro de séquence implicite de TLS, le numéro de séquence utilisé pour calculer le MAC est la valeur de 64 bits formée par l'enchaînement de l'epoch et du numéro de séquence dans l'ordre où ils apparaissent sur le réseau. Noter que epoch + numéro de séquence dans DTLS fait la même longueur que le numéro de séquence TLS.

Le calcul du MAC TLS est paramétré sur le numéro de version du protocole, qui, dans le cas de DTLS, est la version, sur le réseau, c'est-à-dire, {254, 255 } pour DTLS 1.0.

Noter qu'une importante différence entre le traitement du MAC de DTLS et de TLS est que dans le MAC TLS les erreurs doivent résulter en la terminaison de la connexion. Dans DTLS, la mise en œuvre receveuse PEUT simplement éliminer l'enregistrement fautif et continuer la connexion. Ce changement est possible parce que les enregistrements DTLS ne sont pas dépendants les uns des autres comme le sont les enregistrements TLS.

En général, les mises en œuvre de DTLS DEVRAIENT éliminer en silence les données qui ont un mauvais MAC. Si une mise en œuvre de DTLS choisit de générer une alerte quand elle reçoit un message avec un MAC invalide, elle DOIT générer l'alerte mauvais_enregistrement_mac avec le niveau fatal et terminer son état de connexion.

4.1.2.2 Chiffrement de flux nul ou standard

Le chiffrement DTLS NULL est effectué exactement comme le chiffrement TLS 1.1 NULL.

Le seul chiffrement de flux décrit dans TLS 1.1 est RC4, qui ne peut pas être accédé aléatoirement. RC4 NE DOIT PAS être utilisé avec DTLS.

4.1.2.3 Chiffrement de bloc

Le chiffrement et le déchiffrement par chiffrement de bloc de DTLS sont effectués exactement comme avec TLS 1.1.

4.1.2.4 Nouvelles suites de chiffrement

À l'enregistrement, les nouvelles suites de chiffrement TLS DOIVENT indiquer si elles sont convenables pour l'usage de DTLS et quelles adaptations, s'il en est, doivent être faites.

4.1.2.5 Anti-répétition

Les enregistrements DTLS contiennent un numéro de séquence pour assurer la protection contre la répétition. La vérification du numéro de séquence DEVRAIT être effectuée en utilisant la procédure de fenêtre glissante suivante, empruntée au paragraphe 3.4.3 de la [RFC 2402].

Le compteur de paquets receveur pour cette session DOIT être initialisé à zéro à l'établissement de la session. Pour chaque enregistrement reçu, le receveur DOIT vérifier que l'enregistrement contient un numéro de séquence qui ne duplique pas le numéro de séquence de tout autre enregistrement reçu durant la vie de cette session. Ce DEVRAIT être la première vérification appliquée à un paquet après qu'il a été confronté à une session, pour accélérer le rejet des enregistrements dupliqués.

Les dupliqués sont rejetés par l'utilisation d'une fenêtre de réception glissante. (Comment la fenêtre est mise en œuvre est une question locale, mais le texte suivant décrit la fonctionnalité que la mise en œuvre doit exhiber.) Une taille minimum de fenêtre de 32 DOIT être prise en charge, mais une taille de fenêtre de 64 est préférée et DEVRAIT être employée par défaut. Une autre taille de fenêtre (plus grande que le minimum) PEUT être choisie par le receveur. (Le receveur ne notifie pas à l'envoyeur la taille de fenêtre.)

Le bord "droit" de la fenêtre représente la plus forte valeur de numéro de séquence validée reçue sur la session. Les enregistrements qui contiennent des numéros de séquence inférieurs à celui du bord "gauche" de la fenêtre sont rejetés. Les paquets qui tombent dans la fenêtre sont confrontés à une liste de paquets reçus au sein de la fenêtre. Un moyen efficace d'effectuer cette vérification, fondée sur l'utilisation d'un gabarit binaire, est décrite à l'Appendice C de la [RFC2401].

Si l'enregistrement reçu tombe dans la fenêtre et est nouveau, ou si le paquet est à droite de la fenêtre, alors le receveur

procède à la vérification du MAC. Si la validation du MAC échoue, le receveur DOIT éliminer l'enregistrement reçu comme invalide. La fenêtre de réception n'est mise à jour que si la vérification du MAC réussit.

4.2 Protocole DTLS de prise de contact

DTLS utilise les mêmes messages et flux de prise de contact que TLS, avec trois changements principaux :
1. Un échange de mouchard sans état a été ajouté pour empêcher les attaques de déni de service.
2. Modifications de l'en-tête de prise de contact pour traiter la perte de message, le reclassement, et la fragmentation.
3. Des temporisateurs de retransmission pour traiter la perte de message.

À ces exceptions près, les formats, flux, et logique du message DTLS sont les mêmes que ceux de TLS 1.1.

4.2.1 Contre-mesures du déni de service

Les protocoles de sécurité de datagramme sont extrêmement susceptibles à diverses attaques de déni de service (DoS). Deux attaques posent des problèmes particuliers :

1. Un attaquant peut consommer des ressources excessives sur le serveur en transmettant une série de demandes d'initiation de prise de contact, causant l'allocation d'état par le serveur et potentiellement d'effectuer de coûteuses opérations cryptographiques.

2. Un attaquant peut utiliser le serveur comme amplificateur en envoyant des messages d'initiation de connexion avec une source falsifiée de la victime. Le serveur envoie alors son prochain message (dans DTLS, un message Certificate, qui peut être assez grand) à la machine de la victime, l'inondant ainsi.

Afin de contrer ces deux attaques, DTLS emprunte la technique du mouchard sans état utilisée par Photuris [RFC2521] et IKE [RFC2409]. Quand le client envoie son message ClientHello au serveur, le serveur PEUT répondre par un message HelloVerifyRequest. Ce message contient un mouchard sans état généré en utilisant la technique de la [RFC2521]. Le client DOIT retransmettre le ClientHello en ajoutant le mouchard. Le serveur vérifie alors le mouchard et ne poursuit la prise de contact que si il est valide. Ce mécanisme force l'attaquant/client à être capable de recevoir le mouchard, ce qui rend difficiles les attaques de DoS avec des adresses IP usurpées. Ce mécanisme ne fournit pas de défense contre les attaques de DoS montées à partir d'une adresse IP valide.

L'échange est montré ci-dessous :

```
      Client                              Serveur
ClientHello          ---------------------->
                     <----- HelloVerifyRequest (contient le mouchard)
ClientHello  (avec mouchard)   ------>
             [Reste de la prise de contact]
```

DTLS modifie donc le message ClientHello en ajoutant la valeur du mouchard.

```
struct {
  ProtocolVersion client_version;
  Random random;
  SessionID session_id;
  mouchard opaque <0..32>;                    // Nouveau champ
  CipherSuite cipher_suites<2..2^16-1>;
  CompressionMethod compression_methods<1..2^8-1>;
} ClientHello;
```

Quand il envoie le premier ClientHello, le client n'a pas encore de mouchard ; dans ce cas, le champ Moucharrd reste vide (longueur zéro).
La définition de HelloVerifyRequest est la suivante :

```
struct {
  ProtocolVersion serveur_version;
  Mouchard opaque <0..32>;
} HelloVerifyRequest;
```

Le type du message HelloVerifyRequest est hello_verify_request(3).
Le champ server_version est défini dans TLS.

Quand il répond à une HelloVerifyRequest, le client DOIT utiliser les mêmes valeurs de paramètres (version, random, session_id, cipher_suites, compression_method) que dans le ClientHello d'origine. Le serveur DEVRAIT utiliser ces valeurs pour générer son mouchard et vérifier qu'elles sont correctes à réception du mouchard. Le serveur DOIT utiliser le même numéro de version dans la HelloVerifyRequest qu'il aurait utilisé quand il envoie un ServerHello. À réception du ServerHello, le client DOIT vérifier que les valeurs de version du serveur correspondent.

Le serveur DTLS DEVRAIT générer des mouchards d'une façon telle qu'ils puissent être vérifiés sans conserver d'état par client sur le serveur. Une technique est d'avoir un secret généré aléatoirement et de générer les mouchards comme :
Mouchard = HMAC(Secret, Client-IP, Client-Paramètres)

Quand le second ClientHello est reçu, le serveur peut vérifier que le mouchard est valide et que le client peut recevoir les paquets à l'adresse IP donnée.

Une attaque potentielle sur ce schéma est que l'attaquant collecte un certain nombre de mouchards provenant de différentes adresses et les réutilise ensuite pour attaquer le serveur. Le serveur peut se défendre contre cette attaque en changeant la valeur du secret fréquemment, invalidant ainsi ces mouchards. Si le serveur souhaite que les clients légitimes soient capables de prendre contact à travers la transition (par exemple, ils ont reçu un mouchard avec Secret 1 et ensuite envoient le second ClientHello après que le serveur a changé en Secret 2) le serveur peut avoir une fenêtre limitée durant laquelle il va accepter les deux secrets. [IKEv2] suggère d'ajouter un numéro de version aux mouchards pour détecter ce cas. Une autre approche est de simplement essayer de vérifier avec les deux secrets.

Les serveurs DTLS DEVRAIENT effectuer un échange de mouchard chaque fois qu'une nouvelle prise de contact est effectuée. Si le serveur fonctionne dans un environnement où l'amplification n'est pas un problème, le serveur PEUT être configuré à ne pas effectuer d'échange de mouchard. L'échange DEVRAIT cependant être effectué par défaut. De plus, le serveur PEUT choisir de ne pas faire d'échange de mouchard quand une session est reprise. Les clients DOIVENT être prêts à faire un échange de mouchard à chaque prise de contact.

Si HelloVerifyRequest est utilisé, le ClientHello et la HelloVerifyRequest initiaux ne sont pas inclus dans le calcul des verify_data pour le message Finished.

4.2.2 Format du message de prise de contact

Afin de prendre en charge la perte de message, la réorganisation et la fragmentation, DTLS modifie l'en-tête de prise de contact de TLS 1.1 :

```
 struct {
   HandshakeType msg_type;
   uint24 length;
   uint16 message_seq;                    // Nouveau champ
   uint24 fragment_offset;                // Nouveau champ
   uint24 fragment_length;                // Nouveau champ
   select (HandshakeType) {
     cas hello_request: HelloRequest;
     cas client_hello:  ClientHello;
     cas hello_verify_request: HelloVerifyRequest;  // Nouveau type
     cas serveur_hello:  ServerHello;
     cas certificate:Certificate;
     cas serveur_key_exchange: ServerKeyExchange;
     cas certificate_request: CertificateRequest;
     cas serveur_hello_done: ServerHelloDone;
     cas certificate_verify: CertificateVerify;
     cas client_key_exchange: ClientKeyExchange;
     cas finished:Finished;
   } body;
 } Handshake;
```

Le premier message que chaque côté transmet dans chaque prise de contact a toujours message_seq = 0. Chaque fois qu'un

nouveau message est généré, la valeur de message_seq est incrémentée de un. Quand un message est retransmis, la même valeur de message_seq est utilisée. Par exemple :

```
    Client                  Serveur
ClientHello (seq=0)  ------>
                     X<-- HelloVerifyRequest (seq=0) (perdu)
              [Temporisateur expire]
ClientHello (seq=0)  ------>
    (retransmis)
                  <------ HelloVerifyRequest (seq=0)
    ClientHello (seq=1)  ------>
    (avec mouchard)
                  <------      ServerHello (seq=1)
                  <------      Certificate (seq=2)
                  <------   ServerHelloDone (seq=3)
         [Reste de la prise de contact]
```

Noter cependant, que du point de vue de la couche d'enregistrement DTLS, la retransmission est un nouvel enregistrement. Cet enregistrement va avoir une nouvelle valeur de DTLSPlaintext.sequence_number.

Les mises en œuvre de DTLS tiennent (au moins conceptuellement) un compteur next_receive_seq. Ce compteur est initialement réglé à zéro. Quand un message est reçu, si son numéro de séquence correspond à next_receive_seq, next_receive_seq est incrémenté et le message est traité. Si le numéro de séquence est inférieur à next_receive_seq, le message DOIT être éliminé. Si le numéro de séquence est supérieur à next_receive_seq, la mise en œuvre DEVRAIT mettre le message en file d'attente mais PEUT l'éliminer. (C'est un simple compromis espace/bande passante).

4.2.3 Fragmentation et réassemblage de message

Comme noté au paragraphe 4.1.1, chaque message DTLS DOIT tenir dans un seul datagramme de couche transport. Cependant, les messages de prise de contact sont potentiellement plus gros que la taille maximum d'enregistrement. Donc, DTLS fournit un mécanisme pour fragmenter un message de prise de contact sur un certain nombre d'enregistrements.

Quand il transmet le message de prise de contact, l'envoyeur divise le message en une série de N gammes contiguës de données. Ces gammes NE DOIVENT PAS être plus grandes que la taille maximum de fragment de prise de contact et DOIVENT conjointement contenir le message entier de prise de contact. Les gammes NE DEVRAIENT PAS se chevaucher. L'envoyeur crée ensuite N messages de prise de contact, tous avec la même valeur de message_seq comme le message original de prise de contact. Chaque nouveau message est étiqueté avec le fragment_offset (le nombre d'octets contenus dans les fragments précédents) et le fragment_length (la longueur de ce fragment). Le champ Longueur dans tous les messages est le même que le champ Longueur du message original. Un message non fragmenté est un cas particulier qui a fragment_offset=0 et fragment_length=length.

Quand une mise en œuvre de DTLS reçoit un fragment de message de prise de contact, il DOIT le mettre en mémoire tampon jusqu'à ce qu'il ait le message de prise de contact entier. Les mises en œuvre de DTLS DOIVENT être capables de traiter des gammes de fragments qui se chevauchent. Cela permet aux envoyeurs de retransmettre les messages de prise de contact avec des tailles de fragment plus petites durant la découverte de la MTU de chemin.

Noter que comme avec TLS, plusieurs messages de prise de contact peuvent être placés dans le même enregistrement DTLS, pourvu qu'il y ait la place de le faire et qu'ils fassent partie du même envoi. Donc, il y a deux façons acceptables d'empaqueter deux messages DTLS dans le même datagramme : dans le même enregistrement ou dans des enregistrements séparés.

4.2.4 Fin de temporisation et retransmission

Les messages DTLS sont groupés en une série d'envois de messages, selon les diagrammes ci-dessous. Bien que chaque envoi de messages puisse consister en un certain nombre de messages, ils devraient être vus comme monolithiques pour les besoins des fins de temporisation et de retransmission.

```
   Client                  Serveur
ClientHello        -------->                 Envoi 1
```

```
                  <-------   HelloVerifyRequest     Envoi 2
ClientHello          -------->                     Envoi 3
                                ServerHello        \
                               Certificate*         \
                          ServerKeyExchange*        Envoi 4
                         CertificateRequest*       /
                  <--------     ServerHelloDone    /
Certificate*                                      \
ClientKeyExchange                                  \
CertificateVerify*                                 Envoi 5
[ChangeCipherSpec]                                 /
Finished             -------->                    /
                                [ChangeCipherSpec]  \ Envoi 6
                  <--------          Finished      /
```

Figure 1. Envois de messages pour prise de contact complète

```
   Client                          Serveur
ClientHello          -------->                    Envoi 1
                                ServerHello        \
                                [ChangeCipherSpec]  Envoi 2
                  <--------          Finished      /
  [ChangeCipherSpec]                               \ Envoi 3
  Finished           -------->                     /
```

Figure 2. Envois de messages pour prise de contact de reprise de session (pas d'échange de mouchard)

DTLS utilise un simple schéma de fin de temporisation et retransmission avec l'automate à état suivant. Parce que les clients DTLS envoient le premier message (ClientHello), ils commencent dans l'état PREPARING. Les serveurs DTLS commencent dans l'état WAITING, mais avec des mémoires tampons vides et pas de temporisateur de retransmission.

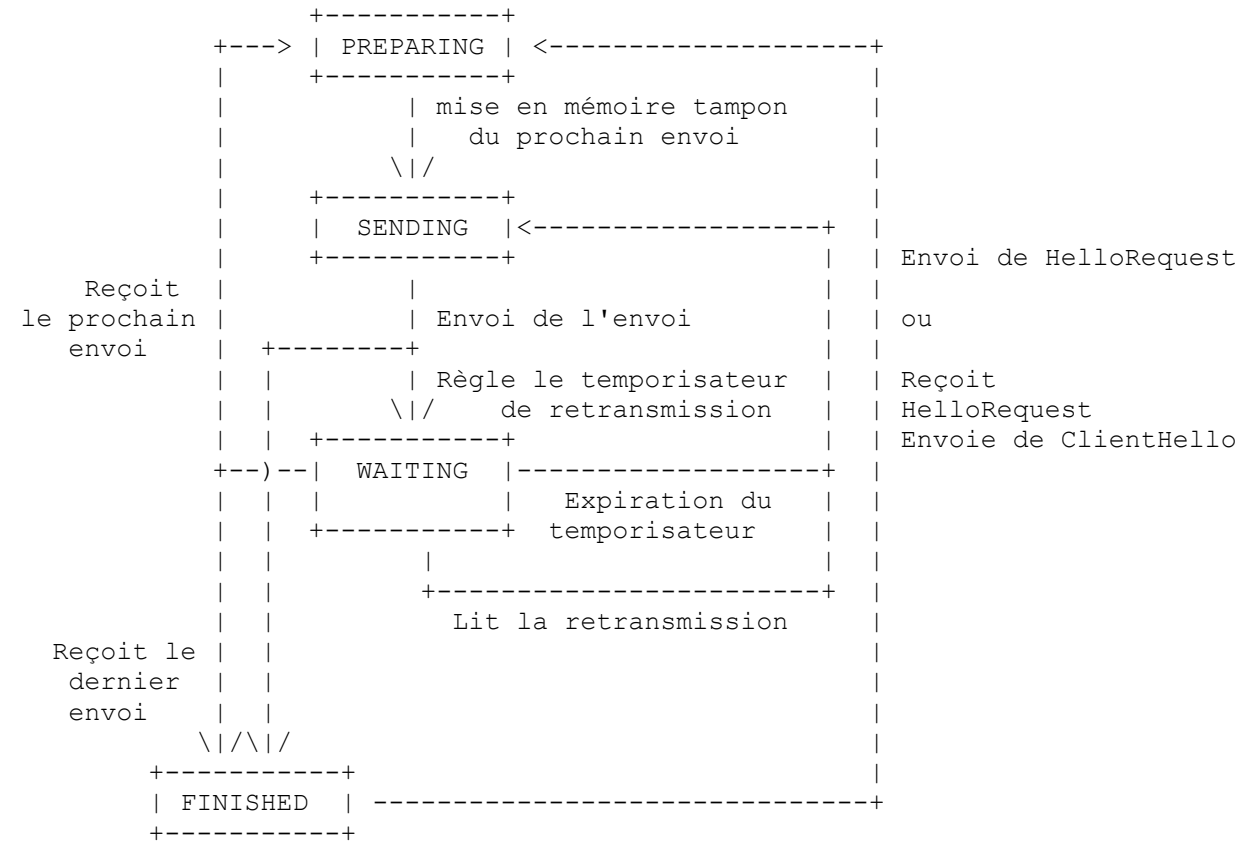

Figure 3. Temporisations et retransmissions de l'automate à états DTLS

L'automate à états a trois états de base.

Dans l'état PREPARING la mise en œuvre fait tous les calculs nécessaires pour préparer le prochain envoi de messages. Elle les met ensuite en mémoire tampon pour la transmission (vidant d'abord la mémoire) et entre dans l'état SENDING.

Dans l'état SENDING, la mise en œuvre transmet l'envoi des messages mis en mémoire tampon. Une fois les messages envoyés, la mise en œuvre entre alors dans l'état FINISHED si c'est le dernier envoi dans la prise de contact. Ou, si la mise en œuvre s'attend à recevoir plus de messages, elle établit un temporisateur de retransmission et entre dans l'état WAITING.

Il y a trois façons de sortir de l'état WAITING :

1. Le temporisateur de retransmission expire : la mise en œuvre passe à l'état SENDING, où elle retransmet l'envoi, remet le temporisateur de retransmission, et retourne à l'état WAITING.

2. La mise en œuvre lit un envoi retransmis de l'homologue : la mise en œuvre passe à l'état SENDING, où elle retransmet l'envoi, remet le temporisateur de retransmission, et retourne à l'état WAITING. La raison en est que la réception d'un message dupliqué est probablement le résultat de l'expiration d'un temporisateur chez l'homologue et donc suggère qu'une partie de l'envoi précédent a été perdue.

3. La mise en œuvre reçoit le prochain envoi de messages : si c'est l'envoi final de messages, la mise en œuvre passe à l'état FINISHED. Si la mise en œuvre a besoin de faire un nouvel envoi, elle passe à l'état PREPARING. Les lectures partielles (de message partiel ou de seulement certains des messages de l'envoi) ne causent pas de transition d'état ou de réinitialisation de temporisateur.

Parce que les clients DTLS envoient le premier message (ClientHello), ils commencent dans l'état PREPARING. Les serveurs DTLS commencent dans l'état WAITING, mais avec des mémoires tampon vides et pas de temporisateur de retransmission.

Quand le serveur désire refaire une prise de contact, il passe de l'état FINISHED à l'état PREPARING pour transmettre le HelloRequest. Quand le client reçoit un HelloRequest il passe de FINISHED à PREPARING pour transmettre le ClientHello.

4.2.4.1 Valeurs de temporisateur

Bien que les valeurs de temporisateur soient du choix de la mise en œuvre, un mauvais réglage du temporisateur peut conduire à de sérieux problèmes d'encombrement ; par exemple, si de nombreuses instances d'un DTLS arrivent précocement en fin de temporisation et retransmettent trop vite sur une liaison encombrée. Les mises en œuvre DEVRAIENT utiliser une valeur initiale de temporisateur de 1 seconde (le minimum défini dans la [RFC2988]) et doubler la valeur à chaque retransmission, jusqu'à pas moins que le maximum de 60 secondes de la RFC 2988. Noter qu'on recommande un temporisateur d'une seconde plutôt que la valeur par défaut de 3 secondes de la RFC 2988 afin d'améliorer la latence pour les applications sensibles au temps. Parce que DTLS utilise seulement la retransmission pour la prise de contact et non pour le flux de données, l'effet sur l'encombrement devrait être minimal.

Les mises en œuvre DEVRAIENT conserver la valeur courante du temporisateur jusqu'à ce qu'une transmission sans perte se produise, moment auquel la valeur peut être remise à sa valeur initiale. Après une longue période de repos, pas moins de dix fois la valeur courante du temporisateur, les mises en œuvre peuvent remettre le temporisateur à la valeur initiale. Une situation où cela pourrait arriver est quand une reprise de contact est utilisée après un transfert de données substantiel.

4.2.5 ChangeCipherSpec

Comme avec TLS, le message ChangeCipherSpec n'est pas techniquement un message de prise de contact mais DOIT être traité comme faisant partie du même envoi que le message Finished associé pour les besoins de temporisation et retransmission.

4.2.6 Messages Finished

Les messages Finished ont le même format que dans TLS. Cependant, afin de supprimer la sensibilité à la fragmentation, le MAC Finished DOIT être calculé comme si chaque message de prise de contact avait été envoyé comme un seul fragment.

Noter que dans les cas où l'échange de mouchard est utilisé, les ClientHello et HelloVerifyRequest initiaux NE DOIVENT PAS être inclus dans le MAC Finished.

4.2.7 Messages Alert

Noter que les messages Alert ne sont pas retransmis du tout, même quand ils surviennent dans le contexte d'une prise de contact. Cependant, une mise en œuvre de DTLS DEVRAIT générer un nouveau message Alert si l'enregistrement en cause est reçu à nouveau (par exemple, comme message de prise de contact retransmis). Les mises en œuvre DEVRAIENT détecter quand un homologue envoie de façon persistante de mauvais messages et terminer l'état de connexion local après qu'un mauvais comportement a été détecté.

4.3 Résumé de la nouvelle syntaxe

Ce paragraphe comporte des spécifications pour les structures de données qui ont changé entre TLS 1.1 et DTLS.

4.3.1 Couche d'enregistrement

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

struct {
  ContentType type;
  ProtocolVersion version;
  uint16 epoch;                                 // Nouveau champ
  uint48 sequence_number;                       // Nouveau champ
  uint16 length;
  opaque fragment[DTLSCompressed.length];
} DTLSCompressed;

struct {
  ContentType type;
  ProtocolVersion version;
  uint16 epoch;                                  // Nouveau champ
  uint48 sequence_number;                        // Nouveau champ
  uint16 length;
  select (CipherSpec.cipher_type) {
    case block:  GenericBlockCipher;
  } fragment;
} DTLSCiphertext;
```

4.3.2 Protocole de prise de contact

```
enum {
  hello_request(0), client_hello(1), server_hello(2),
  hello_verify_request(3),                              // Nouveau champ
  certificate(11), server_key_exchange (12),
  certificate_request(13), server_hello_done(14),
  certificate_verify(15), client_key_exchange(16),
  finished(20), (255)
} HandshakeType;

struct {
```

```
  HandshakeType msg_type;
  uint24 length;
  uint16 message_seq;                    // Nouveau champ
  uint24 fragment_offset;                // Nouveau champ
  uint24 fragment_length;                // Nouveau champ
  select (HandshakeType) {
    cas hello_request: HelloRequest;
    cas client_hello:  ClientHello;
    cas server_hello:  ServerHello;
    cas hello_verify_request: HelloVerifyRequest;  // Nouveau champ
    cas certificate:Certificate;
    cas server_key_exchange: ServerKeyExchange;
    cas certificate_request: CertificateRequest;
    cas server_hello_done:ServerHelloDone;
    cas certificate_verify:  CertificateVerify;
    cas client_key_exchange: ClientKeyExchange;
    cas finished:Finished;
  } body;
} Handshake;

struct {
  ProtocolVersion client_version;
  Random random;
  SessionID session_id;
  opaque cookie<0..32>;                  // Nouveau champ
  CipherSuite cipher_suites<2..2^16-1>;
  CompressionMethod compression_methods<1..2^8-1>;
} ClientHello;

struct {
  ProtocolVersion server_version;
  opaque cookie<0..32>;
} HelloVerifyRequest;
```

5. Considérations sur la sécurité

Le présent document décrit une variante de TLS 1.1 et donc la plupart des considérations de sécurité sont les mêmes que celles de TLS 1.1, décrites dans les Appendices D, E, et F de la [RFC4346].

La principale considération de sécurité supplémentaire soulevée par DTLS est celle du déni de service. DTLS comporte un échange de mouchard conçu pour protéger contre le déni de service. Cependant, les mises en œuvre qui n'utilisent pas cet échange de mouchard sont alors vulnérables à cette attaque. En particulier, les serveurs DTLS qui n'utilisent pas l'échange de mouchard peuvent être utilisés comme amplificateurs d'attaque même si ils ne subissent pas eux-même le déni de service. Donc, les serveurs DTLS DEVRAIENT utiliser l'échange de mouchard sauf si il y a de bonnes raisons de croire que l'amplification n'est pas une menace dans leur environnement. Les clients DOIVENT être prêts à faire un échange de mouchard à chaque prise de contact.

6. Remerciements

Les auteurs tiennent à remercier Dan Boneh, Eu-Jin Goh, Russ Housley, Constantine Sapuntzakis, et Hovav Shacham des discussions et commentaires sur la conception de DTLS. Merci aux relecteurs anonymes de NDSS, notre article original sur [DTLS] pour leurs commentaires. Merci aussi à Steve Kent pour ses retours qui nous ont aidé à préciser de nombreux points. Le paragraphe sur la PMTU a été emprunté à la spécification de DCCP [RFC4340]. Pasi Eronen a effectué une relecture détaillée de cette specification. Des commentaires utiles sur le document ont aussi été reçus de Mark Allman, Jari Arkko, Joel Halpern, Ted Hardie, et Allison Mankin.

7. Considérations relatives à l'IANA

Le présent document utilise le même espace d'identifiants que TLS [RFC4346], de sorte qu'aucun nouveau registre de l'IANA n'est nécessaire. Quand de nouveaux identifiants sont alloués pour TLS, les auteurs DOIVENT spécifier si ils conviennent pour DTLS.

Le présent document définit un nouveau message de prise de contact, hello_verify_request, dont la valeur a été allouée dans le registre TLS HandshakeType défini dans la [RFC4346]. La valeur "3" a été allouée par l'IANA.

8. Références

8.1 Références normatives

[RFC0793] J. Postel (éd.), "Protocole de commande de transmission – Spécification du protocole du programme Internet DARPA", STD 7, septembre 1981.

[RFC1191] J. Mogul et S. Deering, "Découverte de la MTU de chemin", novembre 1990.

[RFC1981] J. McCann, S. Deering, J. Mogul, "Découverte de la MTU de chemin pour IP version 6", août 1996. *(D.S. ; Remplacée par* [RFC8201], STD87)

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174)

[RFC2401] S. Kent et R. Atkinson, "Architecture de sécurité pour le protocole Internet", novembre 1998. (*Obsolète, voir* RFC4301)

[RFC2988] V. Paxson, M. Allman, "Calcul du temporisateur de retransmission de TCP", novembre 2000. *(P.S.)(Obs., voir* RFC6298)

[RFC4346] T. Dierks et E. Rescorla, "Protocole de sécurité de la couche Transport (TLS) version 1.1", avril 2006. *(Remplace* RFC2246 *; Remplacée par* RFC5246 *; MàJ par* RFC4366, 4680, 4681, 5746, 6176, 7465, 7507, 7919)

8.2 Références pour information

[AESCACHE] Bernstein, D.J., "Cache-timing attacks on AES" http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[DTLS] Modadugu, N., Rescorla, E., "The Design and Implementation of Datagram TLS", Proceedings of ISOC NDSS 2004, février 2004.

[RFC1035] P. Mockapetris, "Noms de domaines – Mise en œuvre et spécification", STD 13, novembre 1987. (*MàJ par* RFC1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604, 7766, 8482, 8767)

[RFC1939] J. Myers, M. Rose, "Protocole Post Office - version 3", mai 1996. (*MàJ par* RFC1957, 2449, 8314) (STD0053)

[RFC2246] T. Dierks et C. Allen, "Protocole TLS version 1.0", janvier 1999. *(P.S. ; MàJ par* RFC7919)

[RFC2402] S. Kent et R. Atkinson, "En-tête d'authentification IP", novembre 1998. (*Obsolète, voir* RFC4302, 4305)

[RFC2406] S. Kent et R. Atkinson, "Encapsulation de charge utile de sécurité IP (ESP)", novembre 1998. *(Obsolète, voir* RFC4303)

[RFC2409] D. Harkins et D. Carrel, "L'échange de clés Internet (IKE)", novembre 1998. *(Obsolète, voir la* RFC4306)

[RFC2521] P. Karn, W. Simpson, "Messages d'échec de sécurité ICMP", mars 1999. *(Expérimentale)*

[RFC2960] R. Stewart et autres, "Protocole de transmission de commandes de flux", octobre 2000. (*Obsolète, voir* RFC4960) *(P.S.)*

[RFC3261] J. Rosenberg et autres, "SIP : Protocole d'initialisation de session", juin 2002. (*Mise à jour par* 3265, 3853, 4320, 4916, 5393, 6665, 8217, 8760)

[RFC3501] M. Crispin, "Protocole d'accès au message Internet - version 4rev1", mars 2003. *(P.S. ; MàJ par* RFC4466, 4469, 4551, 5032, 5182, 7817, 8314, 8437, 8474)

[RFC4306] C. Kaufman, "Protocole d'échange de clés sur Internet (IKEv2)", décembre 2005. *(Obsolète, voir la* RFC5996)

[RFC4340] E. Kohler et autres, "Protocole de contrôle d'encombrement de datagrammes (DCCP)", mars 2006. *(P.S.) (MàJ par* 6773)

[RFC5406] S. Bellovin, "Lignes directrices pour spécifier l'utilisation de IPsec version 2", février 2009. (BCP0146)

Adresse des auteurs

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA
mél : ekr@rtfm.com

Nagendra Modadugu
Computer Science Department
Stanford University
Stanford, CA 94305
USA
mél : nagendra@cs.stanford.edu

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est fourni par l'activité de soutien administatif (IASA) de l'IETF.