Groupe de travail Réseau
Request for Comments : 4366
RFC rendue obsolète : 3546
RFC mise à jour : 4346
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle, décembre 2007

S. Blake-Wilson, BCI
M. Nystrom, RSA Security
D. Hopwood, Independent Consultant
J. Mikkelsen, Transactionware
T. Wright, Vodafone
avril 2006

Extensions à la sécurité de la couche Transport (TLS)

Résumé
Le présent document décrit les extensions qui peuvent être utilisées pour ajouter des fonctionnalités à la sécurité de la couche Transport (TLS, *Transport Layer Security*). Il fournit à la fois des mécanismes génériques d'extension pour les messages (hello) client et serveur de prise de contact TLS et des extensions spécifiques qui utilisent ces mécanismes génériques.

Les extensions peuvent être utilisées par les clients et serveurs TLS. Les extensions sont rétro compatibles : la communication est possible entre les clients TLS qui prennent en charge ces extensions et les serveurs TLS qui ne prennent pas en charge ces extensions, et vice versa.

Table des matières

1 Introduction

Le présent document décrit des extensions qui peuvent être utilisées pour ajouter des fonctionnalités à la sécurité de couche transport (TLS, *Transport Layer Security*). Il fournit à la fois des mécanismes d'extension génériques pour les éléments de prise de contact TLS entre client et serveur, et des extensions spécifiques qui utilisent ces mécanismes génériques.

TLS est maintenant utilisé dans une diversité croissante d'environnements de fonctionnement, dont nombre d'entre eux n'étaient pas envisagés lorsque furent déterminés les critères originels de conception de TLS. Les extensions introduites dans ce document sont conçues pour permettre à TLS de fonctionner aussi efficacement que possible dans de nouveaux environnements comme les réseaux sans fil.

Les environnements sans fils souffrent souvent d'un certain nombre de contraintes qui sont rarement présentes dans les environnements filaires. Ces contraintes peuvent inclure des limitations de bande passante, des limitations de puissance de calcul, des limitations de mémoire, et des limitations de durée de vie de batterie.

Les extensions décrites ici se concentrent sur l'extension des fonctionnalités fournies par les formats de message de protocole TLS. D'autres questions, comme l'ajout de nouvelle suites de chiffrement, sont remises à plus tard.

Précisément, les extensions décrites dans ce document :

- Permettent aux clients TLS de fournir au serveur TLS le nom du serveur qu'ils contactent. Cette fonctionnalité est désirable afin de faciliter des connexions sûres avec les serveurs qui hébergent plusieurs serveurs 'virtuels' à une même adresse réseau sous-jacente.

- Permettre aux clients et serveurs TLS de négocier la longueur maximale de fragment à envoyer. Cette fonctionnalité est désirable au titre des contraintes de mémoire pesant sur certains clients, et des contraintes de bande passante pesant sur certains réseaux d'accès.

- Permettre aux clients et serveurs TLS de négocier l'utilisation des URL de certificat de client. Cette fonctionnalité est désirable afin de conserver des archives sur les contraintes des clients.

- Permettre aux clients TLS d'indiquer aux serveurs TLS quels CA racine ils possèdent. Cette fonctionnalité est désirable afin d'empêcher de multiples défaillances de prise de contact impliquant les clients TLS qui ne sont capables de mémoriser qu'un petit nombre de clés de CA racine du fait des limitations de leur mémoire.

- Permettre aux clients et serveurs TLS de négocier l'utilisation de MAC tronqué. Cette fonctionnalité est désirable afin de préserver la bande passante dans les réseaux d'accès limités dans cette ressource.

- Permettre aux clients et serveurs TLS de négocier que le serveur envoie les informations d'état de certificat du client (par exemple, une réponse du protocole d'état de certificat en ligne (OCSP, *Online Certificate Status Protocol*) [OCSP]) pendant une prise de contact TLS. Cette fonctionnalité est souhaitable afin d'éviter d'envoyer une liste de révocation de certificats (CRL) sur un réseau d'accès à ressources limités et donc d'économiser la bande passante.

Pour prendre en charge les extensions ci-dessus, des mécanismes d'extension généraux pour le message hello du client et le message hello du serveur sont introduits.

Les extensions décrites dans le présent document peuvent être utilisées par les clients et les serveurs TLS. Les extensions sont conçues pour être rétro compatibles, ce qui signifie que les clients TLS qui prennent en charge les extensions peuvent parler aux serveurs TLS qui ne prennent pas en charge les extensions, et vice versa. Le document met donc à jour les TLS 1.0 [TLS] et TLS 1.1 [TLSbis].

La rétro compatibilité est principalement réalisée via deux considérations :

- Les clients demandent normalement l'utilisation des extensions via le message hello étendu de client décrit au paragraphe 2.1. TLS exige que les serveurs acceptent les messages hello étendu de client, même si le serveur ne "comprend" pas l'extension.
- Pour les extensions spécifiques décrites ici, aucune réponse obligatoire du serveur n'est exigée quand les clients demandent des fonctionnalités étendues.

La rétro compatibilité est essentiellement réalisée sur la base de l'exigence de TLS que les serveurs qui ne sont pas "au courant des extensions" ignorent les données, ajoutées aux hello des clients, qu'ils ne reconnaissent pas ; par exemple, voir au paragraphe 7.4.1.2 de [TLS].

Noter, cependant, que bien que la rétro compatibilité soit prise en charge, certains clients limités peuvent être forcés de rejeter des communications avec des serveurs qui ne prennent pas en charge les extensions par suite des capacités limitées de tels clients.

Le présent document est une révision de la RFC 3546. Le seul changement majeur concerne la définition de nouvelles extensions. De nouvelles extensions peuvent maintenant être définies via le processus de consensus de l'IETF (plutôt que d'exiger une RFC de normalisation). De plus, quelques éclaircissements mineurs et améliorations rédactionnelles ont été faits.

Le reste de ce document est organisé comme suit. La Section 2 décrit les mécanismes généraux d'extension pour les messages de prise de contact hello du client et du serveur. La Section 3 décrit les extensions spécifiques de TLS. La Section 4 décrit les nouvelles alertes d'erreur à utiliser avec les extensions TLS. Les dernières sections du document traitent des DPI, des considérations sur la sécurité, de l'enregistrement de type MIME d'application/pkix-pkipath, des remerciements et des références.

1.1 Conventions utilisées dans le présent document

Les mots clé "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDÉ", "PEUT", et "FACULTATIF" dans le présent document sont à interpréter comme décrit dans le BCP 14 RFC 2119 [KEYWORDS].

2 Mécanismes généraux d'extension

La présente section présente les mécanismes généraux d'extension pour les messages hello de prise de contact client et serveur TLS.

Ces mécanismes généraux d'extension sont nécessaires afin de permettre aux clients et serveurs de négocier l'utilisation d'extensions spécifique, et comment utiliser les extensions spécifiques. Les formats d'extension décrits sont fondés sur [MAILINGLIST].

Le paragraphe 2.1 spécifie le format de message hello étendu de client,
le paragraphe 2.2 spécifie le format de message hello étendu de serveur, et
le paragraphe 2.3 décrit le format réel d'extension utilisé avec les hellos étendus de client et de serveur.

2.1 Hello étendu de client

Les clients PEUVENT demander la fonction d'extension aux serveurs en envoyant le format de message hello étendu de client à la place du format de message hello de client. Le format de message hello étendu de client est :

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-1>;
    CompressionMethod compression_methods<1..2^8-1>;
    Extension client_hello_extension_list<0..2^16-1>;
} ClientHello;
```

Ici, le nouveau champ "client_hello_extension_list" contient une liste d'extensions. Les formats "Extension" réels sont définis au paragraphe 2.3.

Dans le cas où un client demande des fonctionnalités supplémentaires en utilisant le hello étendu de client, et où cette fonctionnalité n'est pas fournie par le serveur, le client PEUT interrompre la prise de contact.

Noter que le paragraphe 7.4.1.2 de [TLS]permet d'ajouter des informations supplémentaires au message hello du client. Et donc, l'utilisation du hello étendu de client défini ci-dessus ne devrait pas "casser" les serveurs TLS existants.

Un serveur qui prend en charge le mécanisme des extensions DOIT n'accepter les messages hello de client que dans le format original ou le format ClientHello étendu et (comme pour tous les autres messages) DOIT vérifier que la quantité de données dans le message correspond précisément à un de ces formats. Si elle ne le fait pas, il DOIT alors envoyer une alerte fatale "decode_error". Cela outrepasse la "note de compatibilité de transmission" de [TLS].

2.2 Hello étendu de serveur

Le format étendu de message hello de serveur PEUT être envoyé à la place du message hello de serveur lorsque le client a demandé la fonctionnalité étendue via le message hello étendu de client spécifié au paragraphe 2.1. Le format de message hello étendu de serveur est :

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    Extension server_hello_extension_list<0..2^16-1>;
} ServerHello;
```

Ici, le nouveau champ "server_hello_extension_list" contient une liste d'extensions. Le format "Extension" réel est défini au paragraphe 2.3.

Noter que le message hello étendu de serveur est envoyé seulement en réponse à un message hello étendu de client. Cela empêche la possibilité qu'un message hello étendu de serveur puisse "casser" les clients TLS existants.

2.3 Extensions de hello

Le format d'extension pour les hellos étendus de client et de serveur est :

```
struct {
    ExtensionType extension_type;opaque extension_data<0..2^16-1>;
} Extension;
```

Ici :
- "extension_type" identifie le type d'extension particulier.
- "extension_data" contient les informations spécifiques d'un type d'extension particulier.

Les types d'extension définis dans le présent document sont :

```
enum {
    server_name(0), max_fragment_length(1),
    client_certificate_url(2), trusted_ca_keys(3),
    truncated_hmac(4), status_request(5), (65535)
} ExtensionType;
```

La liste des types d'extension définis est conservée par l'IANA. La liste actuelle se trouve à : http://www.iana.org/assignments/tls-extensiontype-values. Voir aux Sections 5 et 8 des informations complémentaires sur la façon d'ajouter de nouvelles valeurs.

Noter que pour tous les types d'extension (y compris ceux qui seront définis à l'avenir), le type d'extension NE DOIT PAS apparaître dans le hello étendu de serveur tant que le même type d'extension n'apparaît pas dans le hello de client correspondant. Et donc les clients DOIVENT interrompre la prise de contact si ils reçoivent dans le hello étendu de serveur un type d'extension qu'ils n'ont pas demandé dans le hello (étendu) de client qui lui est associé.

Néanmoins, les extensions "orientées serveur" pourront être fournies à l'avenir dans ce cadre. Une telle extension (disons, de type x) exigerait que le client envoie d'abord un type d'extension de x dans le hello (étendu) de client avec une extension_data vide pour indiquer qu'il accepte le type d'extension. Dans ce cas, le client offre la capacité de comprendre le type d'extension, et le serveur accepte l'offre du client.

Noter aussi que quand plusieurs extensions de différents types sont présentes dans le hello étendu de client ou le hello étendu de serveur, les extensions peuvent apparaître dans n'importe quel ordre. Il NE DOIT PAS y avoir plus d'une extension de chaque type.

Finalement, noter qu'un hello étendu de client peut être envoyé aussi bien au démarrage d'une nouvelle session que lors d'une demande de reprise de session. Bien sûr, un client qui demande la reprise d'une session ne sait en général pas si le serveur va accepter cette demande, et donc, il DEVRAIT envoyer un hello étendu de client s'il le ferait normalement pour une nouvelle session. En général, la spécification de chaque type d'extension doit inclure une discussion de l'effet de l'extension à la fois pendant de nouvelles sessions et durant des reprises de session.

2.4 Extensions au protocole de prise de contact

Le présent document suggère l'utilisation de deux nouveaux messages de prise de contact, "CertificateURL" et "CertificateStatus". Ces messages sont décrits respectivement aux paragraphes 3.3 et3.6. La structure du nouveau message de prise de contact devient donc :

```
enum {
    hello_request(0), client_hello(1), server_hello(2),
    certificate(11), server_key_exchange (12),
    certificate_request(13), server_hello_done(14),
    certificate_verify(15), client_key_exchange(16),
    finished(20), certificate_url(21), certificate_status(22),
    (255)
} HandshakeType;

struct {
    HandshakeType msg_type;          /* type de prise de contact */
    uint24 length;                   /* octet dans le message */
    select (HandshakeType) {
        case hello_request:          HelloRequest;
        case client_hello:           ClientHello;
        case server_hello:           ServerHello;
        case certificate:            Certificate;
        case server_key_exchange:    ServerKeyExchange;
        case certificate_request:    CertificateRequest;
        case server_hello_done:      ServerHelloDone;
        case certificate_verify:     CertificateVerify;
        case client_key_exchange:    ClientKeyExchange;
        case finished:               Finished;
        case certificate_url:        CertificateURL;
        case certificate_status:     CertificateStatus;
    } body;
} Handshake;
```

3 Extensions spécifiques

La présente section décrit les extensions spécifiques de TLS qui sont spécifiées dans le présent document.

Noter que tout message associé à ces extensions et qui est envoyé pendant la prise de contact TLS DOIT être inclus dans le calcul de hachage impliqué dans les messages "Finished".

Noter aussi que toutes les extensions définies dans cette section ne sont pertinentes que lors de l'initiation de session. Lorsqu'un client inclut un ou plusieurs des types d'extension définis dans un hello étendu de client d'une demande de reprise de session :

- si la demande de reprise est refusée, l'utilisation des extensions est négocié comme d'habitude ;
- si au contraire, la vieille session est reprise, le serveur DOIT alors ignorer les extensions et envoyer un hello de serveur ne contenant aucun des types d'extension. Dans ce cas, les fonctionnalités de ces extensions négociées durant l'initiation de session originale sont appliquées à la session reprise.

Le paragraphe 3.1 décrit l'extension de TLS qui permet à un client d'indiquer quel serveur il contacte. Le paragraphe 3.2 décrit l'extension qui fournit la négociation de longueur maximum de fragment. Le paragraphe 3.3 décrit l'extension qui permet les URL de certificat de client. Le paragraphe 3.4 décrit l'extension qui permet à un client d'indiquer quelles clés racines de CA il possède. Le paragraphe 3.5 décrit l'extension qui permet l'utilisation du HMAC tronqué. Le paragraphe 3.6 décrit l'extension qui prend en charge l'intégration des messages d'information d'état de certificat dans les prises de contact TLS.

3.1 Indication du nom du serveur

TLS ne fournit pas de mécanisme à un client pour dire au serveur le nom du serveur qu'il contacte. Les clients peuvent souhaiter fournir ces informations pour faciliter la sécurisation des connexions avec des serveurs qui hébergent plusieurs

serveurs 'virtuels' sur une seule adresse réseau sous-jacente.

Afin de fournir le nom du serveur, les clients PEUVENT inclure une extension de type "server_name" dans le hello (étendu) de client. Le champ "extension_data" de cette extension DEVRA contenir "ServerNameList" où :

```
struct {
    NameType name_type;
    select (name_type) {
        case host_name: HostName;
    } name;
} ServerName;

enum {
    host_name(0), (255)
} NameType;

opaque HostName<1..2^16-1>;

struct {
    ServerName server_name_list<1..2^16-1>
} ServerNameList;
```

Actuellement, les seuls noms de serveur pris en charge sont les noms d'hôte DNS ; cependant, cela n'implique aucune dépendance de TLS à DNS, et d'autres types de nom peuvent être ajoutés à l'avenir (par une RFC qui mettra à jour le présent document). TLS PEUT traiter les noms de serveur fournis comme des données opaques et passer les noms de types à l'application.

"HostName" contient le nom d'hôte DNS pleinement qualifié du serveur, comme compris par le client. Le nom d'hôte est représenté par une chaîne d'octets utilisant le codage UTF-8 [UTF8], sans point à la fin.

Si les étiquettes de nom d'hôte ne contiennent que des caractères US-ASCII, le client DOIT alors s'assurer que les étiquettes sont seulement séparées par l'octet 0x2E, représentant le caractère point U+002E (malgré l'exigence 1 du paragraphe 3.1 de [IDNA]). Si le serveur a besoin de confronter le nom d'ôte à des noms qui contiennent des caractères non US-ASCII, il DOIT effectuer l'opération de conversion décrite à la Section 4 de [IDNA], qui traite le nom d'hôte comme une "chaîne d'interrogation" (c'est-à-dire que le fanion AllowUnassigned DOIT être mis). Noter que IDNA permet que les étiquettes soient séparées par tout caractère Unicode U+002E, U+3002, U+FF0E, et U+FF61 ; donc, les serveurs DOIVENT accepter tous ces caractères comme séparateur d'étiquette. Si le serveur a seulement besoin de confronter le nom d'hôte à des noms qui contiennent exclusivement des caractères ASCII, il DOIT comparer les noms ASCII sans tenir compte de la casse.

Les adresses IPv4 et IPv6 littérales ne sont pas permises dans "HostName".

Il est RECOMMANDÉ que les clients incluent une extension de type "server_name" dans le hello de client chaque fois qu'ils localisent un serveur par un type de nom pris accepté.

Un serveur qui reçoit un hello de client contenant l'extension "server_name" PEUT utiliser les informations contenues dans l'extension pour guider sa sélection d'un certificat approprié à retourner au client, et/ou autres aspects de politique de sécurité. Dans ce cas, le serveur DEVRA inclure une extension de type "server_name" dans le hello (étendu) de serveur. Le champ "extension_data" de cette extension DEVRA être vide.

Si le serveur comprend l'extension hello de client mais ne reconnaît pas le nom du serveur, il DEVRAIT envoyer une alerte "unrecognized_name" (qui PEUT être fatale).

Si une application négocie un nom de serveur en utilisant un protocole d'application puis passe au niveau de TLS, et si une extension server_name est envoyée, l'extension DEVRAIT alors contenir le même nom que celui négocié dans le protocole d'application. Si le server_name est établi dans la prise de contact de la session TLS, le client NE DEVRAIT PAS essayer de demander un nom de serveur différent à la couche d'application.

3.2 Négociation de longueur de fragment maximale

Sans cette extension, TLS spécifie une longueur de texte en clair fixée à un maximum de 2^14 octets. Il peut être souhaitable à des clients limités de négocier une longueur maximum de fragment plus petite à cause de limitations de mémoire ou de limitations de bande passante.

Pour négocier des longueurs de fragment maximum plus petites, les clients PEUVENT inclure une extension de type "max_fragment_length" dans le hello (étendu) de client. Le champ "extension_data" de cette extension DEVRA contenir :

```
enum{
    2^9(1), 2^10(2), 2^11(3), 2^12(4), (255)
} MaxFragmentLength;
```

dont la valeur est la longueur de fragment maximum désiré. Les valeur admises pour ce champ sont : 2^9, 2^10, 2^11, et 2^12.

Les serveurs qui reçoivent un hello étendu de client qui contient une extension "max_fragment_length" PEUVENT accepter la longueur de fragment maximum demandée en incluant une extension de type "max_fragment_length" dans le hello (étendu) de serveur. Le champ "extension_data" de cette extension DEVRA contenir un "MaxFragmentLength" dont la valeur est la même que celle de la longueur de fragment maximum demandée.

Si un serveur reçoit une demande de négociation de longueur de fragment maximum pour une valeur autre que les valeurs admises, il DOIT interrompre la prise de contact avec une alerte "illegal_parameter". De même, si un client reçoit une réponse de négociation de longueur de fragment maximum qui diffère de la longueur qu'il a demandé, il DOIT aussi interrompre la prise de contact avec une alerte "illegal_parameter".

Une fois qu'une longueur de fragment maximum autre que 2^14 a été négociée avec succès, le client et le serveur DOIVENT immédiatement commencer la fragmentation des messages (y compris les messages de prise de contact), pour s'assurer qu'aucun fragment plus grand que la longueur négociée n'est envoyé. Noter que TLS exige déjà des clients et des serveurs qu'ils prennent en charge la fragmentation des messages de prise de contact.

La longueur négociée s'applique pour la durée de la session y compris les reprises de session.

La longueur négociée limite les entrées que la couche d'enregistrement peut traiter sans fragmentation (c'est-à-dire que valeur maximum de TLSPlaintext.length ; voir [TLS], paragraphe 6.2.1). Noter que le résultat de la couche d'enregistrement peut être plus grand. Par exemple, si la longueur négociée est 2^9=512, pour les suites de chiffrement alors définies (celles définies dans [TLS], [KERB], et [AESSUITES]), et lorsque la compression nulle est utilisée, le résultat de la couche d'enregistrement peut être au plus de 793 octets : 5 octets d'en-têtes, 512 octets de données d'application, 256 octets de bourrage, et 20 octets de MAC. Cela signifie que dans ce cas, un homologue de couche d'enregistrement TLS qui reçoit un message TLS de couche d'enregistrement plus grand que 793 octets peut éliminer le message et envoyer une alerte "record_overflow", sans déchiffrer le message.

3.3 URL de certificat de client

Sans cette extension, TLS spécifie que lors de l'authentification de client, les certificats de client sont envoyés par les clients aux serveurs durant la prise de contacte TLS. Il peut être souhaitable pour les clients limités d'envoyer les URL de certificat à la place des certificats, de façon qu'ils n'aient pas besoin de mémoriser leurs certificats et puissent donc épargner leur mémoire.

Pour négocier l'envoi des URL de certificat à un serveur, les clients PEUVENT inclure une extension du type "client_certificate_url" dans le hello (étendu) de client. Le champ "extension_data" de cette extension DEVRA être vide.

(Noter qu'il est nécessaire de négocier l'utilisation des URL de certificat de client afin d'éviter de "casser" les serveurs TLS existants.)

Les serveurs qui reçoivent un hello de client étendu qui contient une extension "client_certificate_url" PEUT indiquer qu'il veut accepter les URL de certificat en incluant une extension du type "client_certificate_url" dans le hello (étendu) de serveur. Le champ "extension_data" de cette extension DEVRA être vide.

Après l'achèvement réussi de la négociation de l'utilisation des URL de certificat de client (en échangeant des hellos qui incluent les extensions "client_certificate_url") les clients PEUVENT envoyer un message "CertificateURL" à la place du message "Certificate" :

```
enum {
    individual_certs(0), pkipath(1), (255)
} CertChainType;

enum {
    false(0), true(1)
```

```
} Boolean;

struct {
    CertChainType type;
    URLAndOptionalHash url_et_hash_list<1..2^16-1>;
} CertificateURL;

struct {
    opaque url<1..2^16-1>;
    Boolean hash_present;
    select (hash_present) {
        case false: struct {};
        case true: SHA1Hash;
    } hash;
} URLAndOptionalHash;

opaque SHA1Hash[20];
```

Ici, "url_et_hash_list" contient une séquence d'URL et de hachages facultatifs.

Quand on utilise les certificats X.509, il y a deux possibilités :

- Si CertificateURL.type est "individual_certs", chaque URL se réfère à un seul certificat X.509v3 codé en DER, avec l'URL pour le certificat du client en premier.

- Si CertificateURL.type est "pkipath", la liste contient un seul URL qui se réfère à une chaîne de certificats codés en DER, qui utilise le type PkiPath décrit à la Section 8.

Pour l'utilisation de tout autre format de certificat, la spécification qui décrit l'usage de ce format dans TLS devrait définir le format de codage des certificats ou chaînes de certificat, et toutes les contraintes de leur ordonnancement.

Le hachage correspondant à chaque URL est ou n'est pas présent à la discrétion du client, ou est le hachage SHA-1 du certificat ou chaîne de certificats (dans le cas de certificats X.509, le certificat ou le PkiPath, codé en DER).

Noter que lorsque une liste d'URL est utilisée pour des certificats X.509, l'ordre des URL est le même que celui utilisé dans le message Certificate de TLS (voir le paragraphe 7.4.2 de [TLS]), mais l'opposé de l'ordre dans lequel les certificats sont codés dans PkiPath. Dans les deux cas, le certificat racine auto signé PEUT être omis de la chaîne, dans l'hypothèse que le serveur le possède déjà aux fins de validation.

Les serveurs qui reçoivent "CertificateURL" DEVRAONT essayer de restituer la chaîne de certificat du client d'après les URL puis traiter la chaîne de certificats comme d'habitude. Une copie de mémoire cache du contenu de tout URL de la chaîne PEUT être utilisé, pourvu qu'un hachage SHA-1 soit présent pour cet URL et qu'il corresponde au hachage de la copie en mémoire cache.

Les serveurs qui acceptent cette extension DOIVENT accepter le schéma d'URL http: pour les URL de certificats, et PEUVENT accepter d'autres schémas. L'utilisation d'autres schémas que "http", "https", ou "ftp" peut créer des problèmes inattendus.

Si le protocole utilisé est HTTP, le serveur HTTP peut alors être configuré pour utiliser les directives Cache-Control et Expires décrites dans [HTTP] pour spécifier si les certificats ou chaînes de certificat devraient être mises en mémoire cache et pour combien de temps.

Le serveur TLS n'est pas obligé de suivre la directive redirects de HTTP lorsqu'il restitue les certificats ou chaînes de certificat. Les URL utilisés dans cette extension DEVRAIENT donc être choisis de façon à ne pas dépendre de ces directives redirects.

Si le protocole utilisé pour restituer les certificats ou chaînes de certificats retourne une réponse formatée en MIME (comme le fait HTTP) les types de contenu MIME suivants DEVRONT alors être utilisés : lorsque un seul certificat X.509v3 est retourné, le type de contenu est "application/pkix-cert" [PKIOP], et quand une chaîne de certificats X.509v3 est retournée, le type de contenu est "application/pkix-pkipath" (voir la Section 8).

Si un hachage SHA-1 est présent pour un URL, le serveur DOIT alors vérifier que le hachage SHA-1 des contenus de l'objet restitué à partir de cet URL (après décodage de tout contenu/transfert/codage MIME) correspond au hachage donné.

Si un des objets restitués n'a pas le hachage SHA-1 correct, le serveur DOIT interrompre la prise de contact avec une alerte "bad_certificate_hash_value".

Noter que les clients peuvent choisir d'envoyer "Certificate" ou bien "CertificateURL" après avoir réussi la négociation de l'option d'envoyer les URL de certificat. L'option d'envoyer un certificat est incluse pour donner de la souplesse aux clients qui possèdent plusieurs certificats.

Si un serveur rencontre un délai déraisonnable pour obtenir des certificats sur un CertificateURL donné, il DEVRAIT provoquer une fin de temporisation et signaler une alerte d'erreur "certificate_unobtainable".

3.4 Indication d'autorité de certification de confiance

Les clients limités qui, du fait de limitations de mémoire, ne possèdent qu'un petit nombre de clés racines de CA, souhaitent indiquer aux serveurs les clés racine qu'ils possèdent, afin d'éviter des défaillances répétées de prise de contact.

Pour indiquer quelles clés racines de CA ils possèdent, les clients PEUVENT inclure une extension du type "trusted_ca_keys" dans le hello (étendu) de client. Le champ "extension_data" de cette extension DEVRA contenir "TrustedAuthorities" où :

```
struct {
    TrustedAuthority trusted_authorities_list<0..2^16-1>;
} TrustedAuthorities;

struct {
    IdentifierType identifier_type;
    select (identifier_type) {
        case pre_agreed: struct {};
        case key_sha1_hash: SHA1Hash;
        case x509_name: DistinguishedName;
        case cert_sha1_hash: SHA1Hash;
    } identifier;
} TrustedAuthority;

enum {
    pre_agreed(0), key_sha1_hash(1), x509_name(2),
    cert_sha1_hash(3), (255)
} IdentifierType;

opaque DistinguishedName<1..2^16-1>;
```

Ici "TrustedAuthorities" donne une liste des identifiants de clé racine de CA que possède le client. Chaque clé racine de CA est identifiée via :

- "pre_agreed" : aucune identité de clé racine de CA n'est fournie.
- "key_sha1_hash" : contient le hachage SHA-1 de la clé racine de CA. Pour les clés d'algorithme de signature numérique (DSA, *Digital Signature Algorithm*) et d'algorithme de signature numérique à courbe elliptique (ECDSA, *Elliptic Curve Digital Signature Algorithm*) c'est le hachage de la valeur "subjectPublicKey". Pour les clés RSA, le hachage est la représentation de la chaîne d'octets gros boutiens du module sans aucun octet à valeur zéro initiale. (Cela copie les formats de hachage de clé développés dans les autres environnements.)
- "x509_name" : contient le nom distinctif X.509 codé en DER de la CA.
- "cert_sha1_hash" : contient le hachage SHA-1 du certificat codé en DER qui contient la clé racine de la CA.

Noter que dans cette extension, les clients peuvent inclure zéro, quelques unes, ou toutes les clés racines de CA qu'ils possèdent.

Noter aussi qu'il est possible qu'un hachage de clé ou un nom distinctif seul n'identifie pas de façon univoque un producteur de certificat (par exemple, si une CA particulière a plusieurs paires de clés). Cependant, nous supposons ici que c'est le cas, suivant l'utilisation des noms distinctifs pour identifier les producteurs de certificats dans TLS.

L'option de n'inclure aucune clé racine de CA est ajoutée pour permettre au client d'indiquer la possession de certains ensembles pré définis de clés racines de CA.

Les serveurs qui reçoivent un hello de client contenant l'extension "trusted_ca_keys" PEUT utiliser les informations contenues dans l'extension pour guider leur choix d'une chaîne de certificats appropriée à retourner au client. Dans ce cas,

le serveur DEVRA inclure une extension du type "trusted_ca_keys" dans le hello (étendu) de serveur. Le champ "extension_data" de cette extension DEVRA être vide.

3.5 HMAC tronqué

Les suites de chiffrement TLS couramment définies utilisent la construction MAC HMAC aussi bien avec MD5 que SHA-1 [HMAC] pour authentifier les communications de couche d'enregistrement. Dans TLS, la totalité du résultat de la fonction de hachage est produite comme étiquette MAC. Cependant, il peut être souhaitable dans les environnements limités d'économiser la bande passante en tronquant le résultat de la fonction de hachage à 80 bits lors de la formation des étiquettes MAC.

Pour négocier l'utilisation du HMAC tronqué à 80 bits, les clients PEUVENT inclure une extension du type "truncated_hmac" dans le hello étendu de client. Le champ "extension_data" de cette extension DEVRA être vide.

Les serveurs qui reçoivent un hello étendu qui contient une extension "truncated_hmac" PEUVENT se mettre d'accord pour utiliser un HMAC tronqué en incluant une extension du type "truncated_hmac", avec "extension_data" vide, dans le hello étendu de serveur.

Noter que si on ajoute de nouvelles suites de chiffrement qui n'utilisent pas HMAC, et si la session négocie une de ces suites de chiffrement, cette extension n'aura aucun effet. Il est fortement recommandé que toute nouvelle suite de chiffrement utilisant d'autres MAC considère la taille MAC comme partie intégrante de la définition de la suite de chiffrement, en prenant en compte les considérations à la fois de sécurité et de bande passante.

Si le HMAC tronqué a été négocié avec succès durant une prise de contact TLS, et si la suite de chiffrement négociée utilise HMAC, le client et le serveur passent tous deux ce fait à la couche d'enregistrement TLS avec les autres paramètres de sécurité négociés. Ultérieurement durant la session, les clients et serveurs DOIVENT utiliser les HMAC tronqués, calculés comme spécifié dans [HMAC]. C'est-à-dire que CipherSpec.hash_size fait 10 octets, et seuls les 10 premiers octets du résultat HMAC sont transmis et vérifiés. Noter que cette extension n'affecte pas le calcul de la fonction pseudo aléatoire (PRF, *pseudo-random function*) au titre de la prise de contact ou de la déduction de clé.

La taille du HMAC tronqué négocié s'applique pour la durée de la session y compris les reprises de session.

3.6 Demande d'état de certificat

Les clients limités peuvent souhaiter utiliser un protocole d'état de certificat comme OCSP [OCSP] pour vérifier la validité des certificats de serveur, afin d'éviter la transmission des CRL et donc d'économiser la bande passante sur des réseaux à ressources limitées. Cette extension permet d'envoyer de telles informations dans la prise de contact TLS, économisant ainsi des allers-retours et des ressources.

Afin d'indiquer leur désir de recevoir des informations d'état de certificat, les clients PEUVENT inclure une extension du type "status_request" dans le hello (étendu) de client. Le champ "extension_data" de cette extension DEVRA contenir "CertificateStatusRequest" où :

```
struct {
    CertificateStatusType status_type;
    select (status_type) {
        case ocsp: OCSPStatusRequest;
    } request;
} CertificateStatusRequest;

enum { ocsp(1), (255) } CertificateStatusType;

struct {
    ResponderID responder_id_list<0..2^16-1>;
    Extensions  request_extensions;
} OCSPStatusRequest;
opaque ResponderID<1..2^16-1>;
opaque Extensions<0..2^16-1>;
```

Dans la demande OCSPStatusRequest, les "ResponderID" donnent une liste des répondants OCSP qui sont de confiance pour le client. Une séquence "responder_id_list" de longueur zéro a la signification particulière que les répondants sont implicitement connus du serveur, par exemple, par accord préalable. "Extensions" est un codage en DER des extensions de demande OCSP.

"ResponderID" et "Extensions" sont tous deux des types ASN.1 types codés en DER comme défini dans [OCSP]. "Extensions" est importé de [PKIX]. Une valeur de "request_extensions" de longueur zéro signifie qu'il n'y a pas d'extension (par opposition à une SÉQUENCE ASN.1 de longueur zéro, qui n'est pas valide pour le type "Extensions").

Dans le cas de l'extension OCSP "id-pkix-ocsp-nonce", [OCSP] n'est pas clair sur son codage ; pour être plus clair, le nom occasionnel DOIT être une CHAÎNE D'OCTETS codée en DER, qui est encapsulée comme une autre CHAÎNE D'OCTETS (noter que la conformité à cette exigence des mises en œuvre fondées sur un client OCSP existant devra être vérifiée).

Les serveurs qui reçoivent un hello de client qui contient l'extension "status_request" PEUVENT retourner une réponse d'état de certificat convenable au client avec leur certificat. Si OCSP est demandé, ils DEVRAIENT utiliser les informations contenues dans l'extension lors du choix d'un OCSP répondant et DEVRAIENT inclure request_extensions dans la demande OCSP.

Les serveurs retournent une réponse de certificat avec leur certificat en envoyant un message "CertificateStatus" immédiatement après le message "Certificate" (et avant tout message "ServerKeyExchange" ou "CertificateRequest"). Si un serveur retourne un message "CertificateStatus", le serveur DOIT alors avoir une extension du type "status_request" incluse avec "extension_data" vide dans le hello étendu de serveur.

```
struct {
    CertificateStatusType status_type;
    select (status_type) {
        case ocsp: OCSPResponse;
    } response;
} CertificateStatus;

opaque OCSPResponse<1..2^24-1>;
```

Une "ocsp_response" contient une réponse OCSP complète, codée en DER (en utilisant le type ASN.1 OCSPResponse défini dans [OCSP]). Noter qu'on ne peut envoyer qu'une seule réponse OCSP.

Le message "CertificateStatus" est convoyé en utilisant le type de message de prise de contact "certificate_status".

Noter qu'un serveur PEUT aussi choisir de ne pas envoyer un message "CertificateStatus", même si il reçoit une extension "status_request" dans le message hello de client.

Noter de plus que les serveurs NE DOIVENT PAS envoyer de messages "CertificateStatus" à moins qu'il n'aient reçu une extension "status_request" dans le message hello de client.

Les clients qui demandent une réponse OCSP et reçoivent une réponse OCSP dans un message "CertificateStatus" DOIVENT vérifier la réponse OCSP et interrompre la prise de contact si la réponse n'est pas satisfaisante.

4 Alertes d'erreur

La présente section définit de nouvelles alertes d'erreur à utiliser avec les extensions TLS définies dans le présent document.

Les nouvelles alertes d'erreur suivantes sont définies. Pour éviter de "casser" les clients et serveurs existants, ces alertes NE DOIVENT PAS être envoyées à moins que l'envoyeur n'ait reçu un message hello étendu de la part de celui avec qui elles communiquent.

- "unsupported_extension" : cette alerte est envoyée par les clients qui reçoivent un hello étendu de serveur contenant une extension qu'ils n'ont pas mis dans le hello de client correspondant (voir au paragraphe 2.3). Ce message est toujours fatal.

- "unrecognized_name" : cette alerte est envoyée par les serveurs qui reçoivent une demande d'extension server_name, mais ne reconnaissent pas le nom du serveur. Ce message PEUT être fatal.

- "certificate_unobtainable" : cette alerte est envoyée par les serveurs qui sont incapables de restituer une chaîne de certificat à partir de l'URL fourni par le client (voir au paragraphe 3.3). Ce message PEUT être fatal ; par exemple, si

l'authentification du client est requise par le serveur pour continuer la prise de contact et que le serveur est incapable de restituer la chaîne de certificat, il peut envoyer une alerte fatale.

- "bad_certificate_status_response" : cette alerte est envoyée par les clients qui reçoivent une réponse d'état de certificat invalide (voir au paragraphe 3.6). Ce message est toujours fatal.

- "bad_certificate_hash_value" : cette alerte est envoyée par les serveurs lorsque un hachage de certificat ne correspond pas au certificate_hash fourni par le client. Ce message est toujours fatal.

Ces alertes d'erreur sont convoyées en utilisant la syntaxe suivante :

```
enum {
    close_notify(0),
    unexpected_message(10),
    bad_record_mac(20),
    decryption_failed(21),
    record_overflow(22),
    decompression_failure(30),
    handshake_failure(40),
    /* 41 n'est pas défini, pour des raisons historiques */
    bad_certificate(42),
    unsupported_certificate(43),
    certificate_revoked(44),
    certificate_expired(45),
    certificate_unknown(46),
    illegal_parameter(47),
    unknown_ca(48),
    access_denied(49),
    decode_error(50),
    decrypt_error(51),
    export_restriction(60),
    protocol_version(70),
    insufficient_security(71),
    internal_error(80),
    user_canceled(90),
    no_renegotiation(100),
    unsupported_extension(110),           /* nouveau */
    certificate_unobtainable(111),        /* nouveau */
    unrecognized_name(112),               /* nouveau */
    bad_certificate_status_response(113), /* nouveau */
    bad_certificate_hash_value(114),      /* nouveau */
    (255)
} AlertDescription;
```

5 Procédure pour définir de nouvelles extensions

La liste des types d'extension, telle que définie au paragraphe 2.3, est conservée par l'Autorité d'allocation des numéros de l'Internet (IANA). Et donc, une demande doit être faite auprès de l'IANA afin d'obtenir une nouvelle valeur de type d'extension. Comme il y a de subtiles (et de pas si subtiles) interactions qui peuvent survenir dans ce protocole entre les dispositifs nouveaux et existants qui peuvent déboucher sur une réduction significative de la sécurité globale, les nouvelles valeurs ne DEVRONT être définies qu'à travers le processus de consensus de l'IETF spécifié dans [IANA].

(Cela signifie que les nouvelles allocations ne peuvent être effectuée que via des RFC approuvées par l'IESG.)

Les considérations suivantes devraient être prises en compte lors de la conception de nouvelles extensions :

- Toutes les extensions définies dans le présent document suivent la convention selon laquelle à chaque extension demandée par un client et comprise par le serveur, le serveur réplique par une extension du même type.

- Certains cas où un serveur n'est pas d'accord avec une extension sont des conditions d'erreur, et d'autres sont simplement un refus de prendre en charge un dispositif particulier. En général, les alertes d'erreur devraient être utilisée

pour celles-là, et un champ dans la réponse d'extension du serveur pour celles-ci.

- Les extensions devraient autant que possible être conçues pour empêcher toute attaque qui force l'utilisation (ou la non-utilisation) d'un dispositif particulier en manipulant les messages de prise de contact. Ce principe devrait être suivi que le dispositif soit considéré ou non comme posant un problème de sécurité.

Le fait que les champs d'extension soient inclus dans les entrées des hachages du message Finished sera souvent suffisant, mais une extrême attention est nécessaire lorsque l'extension change la signification des messages envoyés dans la phase de prise de contact. Les concepteurs et développeurs devraient être conscients du fait que jusqu'à ce que la prise de contact ait été authentifiée, des attaquants actifs peuvent modifier les messages et insérer, retirer, ou remplacer des extensions.

- Il serait techniquement possible d'utiliser des extensions pour changer des aspects majeurs du concept de TLS ; par exemple, le concept de négociation de suite de chiffrement. Cela n'est pas recommandé ; il serait plus approprié de définir une nouvelle version de TLS, en particulier parce que les algorithmes de prise de contact de TLS ont des protections spécifiques contre les attaques de dégradation de version fondées sur le numéro de version. La possibilité d'une dégradation de version devrait être une considération significative dans tout changement majeur de concept.

6 Considérations pour la sécurité

Les considérations pour la sécurité du mécanisme d'extension en général et pour la conception de nouvelles extensions sont décrites dans la section précédente. Une analyse de la sécurité pour chacune des extensions définies dans le présent document est donnée ci-dessous.

En général, les développeurs devraient continuer à surveiller l'état de l'art et remédier à toute faiblesse qu'ils identifient.

Des considérations supplémentaires sur la sécurité sont décrites dans les RFC TLS 1.0 [TLS] et TLS 1.1 [TLSbis].

6.1 Sécurité de server_name

Si un seul serveur héberge plusieurs domaines, il est clairement nécessaire que les possesseurs de chaque domaine s'assurent que cela satisfait leur besoins de sécurité. À part cela, server_name ne paraît pas introduire de problèmes de sécurité significatifs.

Les mises en œuvre DOIVENT s'assurer que ne surviennent pas de débordements de mémoire tampon, quelle que soit les valeurs des champs de longueur dans le server_name.

Bien que le présent document spécifie un codage pour les noms d'hôtes internationalisés dans l'extension server_name, il ne traite aucun des problèmes de sécurité associés à l'utilisation des noms d'hôte internationalisés dans TLS (en particulier, les conséquences des noms "usurpés" qu'on ne peut pas distinguer d'un autre nom lorsqu'ils sont affichés ou imprimés). Il est recommandé que les certificats de serveur ne soient pas produits pour des noms d'hôte internationalisés à moins que des procédures ne soient mises en place pour atténuer le risque d'usurpation de noms d'hôtes.

6.2 Sécurité de max_fragment_length

La longueur de fragment maximum prend effet immédiatement, y compris pour les messages de prise de contact. Cependant, cela n'introduit aucune complication pour la sécurité qui ne soit déjà présente dans TLS, car TLS exige que les mises en œuvre soient capables de traiter les messages de prise de contact fragmentés.

Noter que, comme décrit au paragraphe 3.2, une fois qu'une suite de chiffrement non nulle a été activée, la longueur effective de fragment maximum dépend de la suite de chiffrement et de la méthode de compression, ainsi que du max_fragment_length négocié. Cela doit être pris en compte lors du dimensionnement des mémoires tampon, et de la vérification des débordements de mémoire tampon.

6.3 Sécurité de client_certificate_url

Cette extension pose deux problèmes majeurs.

Le premier problème majeur est de savoir si les clients devraient ou non inclure les hachages de certificat lorsqu'ils envoient les URL de certificat.

Lorsqu'on utilise l'authentification de client *sans* l'extension client_certificate_url, la chaîne de certificat de client est couverte par les hachages de message Finished. L'objet de l'inclusion des hachages et de leur vérification par rapport à la chaîne de certificats restituée est de s'assurer que les mêmes propriété tiennent lorsqu'on utilise cette extension, c'est-à-dire que toutes les informations de la chaîne de certificats restituée par le serveur sont ce que le client attendait.

D'un autre côté, omettre les hachages de certificats active une fonction qui est souhaitable dans certaines circonstances ; par exemple, les clients peuvent recevoir quotidiennement des certificats qui sont mémorisés dans un URL fixé et n'ont pas besoin d'être fournis au client. Les clients qui choisissent d'omettre les hachages de certificats devraient être au courant de la possibilité d'une attaque dans laquelle l'attaquant obtient un certificat valide sur la clé du client qui est différent du certificat que le client entendait fournir. Bien que TLS utilise les hachages MD5 et SHA-1 en divers autres endroits, cela n'a pas été jugé nécessaire ici. La propriété requise de SHA-1 est une résistance pré image seconde.

Le second problème majeur est que la prise en charge de client_certificate_url implique que le serveur agisse comme client dans un autre protocole d'URL. Le serveur devient donc soumis à beaucoup des mêmes problèmes de sécurité que les clients du schéma d'URL, avec le problème supplémentaire que le client peut essayer d'engager le serveur à se connecter à un URL (éventuellement bizarre).

En général, ce problème signifie qu'un attaquant pourrait utiliser le serveur pour attaquer indirectement un autre hôte qui serait vulnérable à certaines failles de la sécurité. Cela introduit aussi la possibilité d'attaques de déni de service dans lesquelles un attaquant fait ne nombreuses connexions avec le serveur, chacune d'elles résultant en une tentative de connexion du serveur sur la cible de l'attaque.

Noter que le serveur peut être derrière un pare-feu ou autre, capable d'accéder à des hôtes qui ne seraient pas directement accessible à partir de l'Internet public. Cela peut exacerber les problèmes potentiels de sécurité et de déni de service décrits plus haut, aussi bien que permettre de confirmer l'existence d'hôtes internes qui resteraient cachés autrement.

Les problèmes détaillés de sécurité impliqués vont dépendre des schémas d'URL pris en charge par le serveur. Dans le cas de HTTP, le problème est similaire à ceux qui s'appliquent à un serveur mandataire HTTP en accès public. Dans le cas de HTTPS, des boucles et des impasses peuvent être créées, et cela devrait être fait. Dans le cas de FTP, il arrive des attaques qui sont similaires des attaques par rebond de FTP.

Au titre de ce problème, il est RECOMMANDÉ que l'extension client_certificate_url doive être activée spécifiquement par un administrateur du serveur, plutôt qu'activée par défaut. Il est aussi RECOMMANDÉ que les protocoles d'URI soient activés individuellement par l'administrateur, et que seul un ensemble minimal de protocoles soit activé. Les protocoles inhabituels qui offrent une sécurité limitée ou ceux dont la sécurité n'est pas bien comprise DEVRAIENT être évités.

Comme exposé dans [URI], les URL qui spécifient des accès autres que ceux par défaut peuvent causer des problèmes, comme le peuvent les très longs URL (qui sont plus vraisemblablement utiles pour exploiter les fautes de débordement de mémoire tampon).

Noter aussi que les mandataires de mémoire cache HTTP sont courants sur l'Internet, et certains mandataires ne vérifient pas correctement la dernière version d'un objet. Si une demande qui utilise HTTP (ou autre protocole à mémoire cache) passe à travers un mandataire mal configuré ou défectueux, le mandataire peut renvoyer une réponse périmée.

6.4 Sécurité de trusted_ca_keys

Il est possible que les clés racines de CA que possède un client soient considérées comme des informations confidentielles. Il en résulte que l'extension d'indication de clé racine de CA devrait être utilisée avec prudence.

L'utilisation de la solution de remplacement du hachage de certificat SHA-1 assure que chaque certificat est spécifié de façon non ambiguë. Comme pour l'extension précédente, il n'a pas été jugé nécessaire d'utiliser les deux hachages MD5 et SHA-1.

6.5 Sécurité de truncated_hmac

Il est possible que les MAC tronqués soient plus faibles que les MAC "non tronqués". Cependant, aucune faiblesse significative n'est actuellement connue ou attendus pour HMAC avec MD5 ou SHA-1, tronqué à 80 bits.

Noter que la longueur de sortie d'un MAC n'a pas besoin d'être aussi longue que celle d'une clé de chiffrement symétrique, car la falsification de valeurs de MAC ne peut pas être faite hors ligne : dans TLS, un seul soupçon de défaillance de MAC va causer la terminaison immédiate de la session TLS.

Comme l'algorithme MAC ne prend effet qu'après tous les messages de prise de contact qui affectent les paramètres d'extension aient été authentifiés par les hachages dans les messages Finished, il n'est pas possible à un attaquant actif de forcer la négociation de l'extension de HMAC tronqué alors qu'il n'aurait pas été utilisé autrement (dans la mesure où l'authentification de la prise de contact est sûre). Donc, dans le cas où on trouverait à l'avenir un problème de sécurité avec le HMAC tronqué, si le client ou le serveur pour une session donnée était mis à jour pour prendre le problème en compte, il devrait être capable d'interdire l'utilisation de cette extension.

6.6 Sécurité de status_request

Si un client demande une réponse OCSP, il doit tenir compte de ce que le serveur d'un attaquant qui utilise une clé compromise pourrait (et le ferait probablement) prétendre ne pas accepter l'extension. Dans ce cas, un client qui demande la validation de certificats OCSP DEVRAIT soit contacter directement le serveur OCSP, soit interrompre la prise de contact.

L'utilisation de l'extension de demande de nom occasionnel OCSP (id-pkix-ocsp-nonce) peut améliorer la sécurité contre les attaques qui tentent de répéter les réponses OCSP ; voir des précisions au paragraphe 4.4.1 de [OCSP].

7 Considérations pour l'internationalisation

Aucune des extensions définies ici n'utilise directement des chaîne soumises à localisation. Les noms d'hôte du système de noms de domaine (DNS) sont codés avec UTF-8. Si des extensions futures utilisent des chaînes de texte, l'internationalisation devrait être envisagée dès leur conception.

8 Considérations relatives à l'IANA

Les paragraphes 2.3 et 5 décrivent un registre des valeur de ExtensionType tenu par l'IANA. Les valeurs de ExtensionType sont allouées via le consensus IETF défini dans la RFC 2434 [IANA]. Le registre initial correspond à la définition de "ExtensionType" au paragraphe 2.3.

Le type MIME "application/pkix-pkipath" a été enregistré par l'IANA avec le gabarit suivant :
À : ietf-types@iana.org
Sujet : Enregistrement du type de support MIME application/pkix-pkipath
Nom du type de support MIME : application
Nom de sous-type MIME : pkix-pkipath
Paramètres requis : aucun
Paramètres facultatifs : version (la valeur par défaut est "1")
Considérations de codage : Ce type MIME est un codage DER du type ASN.1 PkiPath, défini comme suit :
PkiPath ::= SÉQUENCE DE Certificate
PkiPath est utilisé pour représenter un chemin de certification. Au sein de la séquence, l'ordre des certificats est tel que le sujet du premier certificat soit le producteur du second certificat, etc.
Ceci est identique à la définition publiée dans [X509-4th-TC1] ; noter qu'elle est différente de celle de [X509-4th].
Tous les certificats DOIVENT se conformer à [PKIX]. (Ceci devrait être interprété comme exigence de ne coder que les certificats conformes à PKIX qui utilisent ce type. Cela n'exige pas nécessairement que tous les certificats qui ne sont pas strictement conformes à PKIX soient rejetés par les parties consommatrices, bien que les conséquences pour la sécurité d'accepter un quelconque de tels certificats devrait être considéré avec prudence.)

Le codage en DER (par opposition à BER) DOIT être utilisé. Si ce type est envoyé sur un transport à 7 bits, le codage base64 DEVRAIT être utilisé.

Considérations de sécurité : Les considérations de sécurité de [X509-4ème] et [PKIX] (ou toutes leurs mises à jour) s'appliquent, ainsi que celles de tout protocole qui utilise ces type (par exemple, TLS).

Noter que ce type ne spécifie qu'une chaîne de certificats dont la validité ne peut être attestée que conformément à la configuration existante du consommateur d'assertions pour les autorités de certificats de confiance ; il n'est pas destiné à être utilisé pour spécifier un changement de cette configuration.

Considérations d'interopérabilité : Aucun problème spécifique d'interopérabilité n'est connu avec ce type, mais pour les recommandations se rapportant aux certificats X.509 en général, voir [PKIX].

Spécification publiée : RFC 4366 (le présent mémo), et [PKIX].

Applications qui utilisent ce type de support : TLS. Il peut aussi être utilisé par d'autres protocoles, ou pour des échanges généraux de chaînes de certificats PKIX.

Information supplémentaires :
Numéros magiques : L'ASN.1 codé en DER peut être reconnu facilement.
Une analyse complémentaire est nécessaire pour le distinguer des autres types d'ASN.1.
Extension de fichier : .pkipath
Code de type de fichier Macintosh : non spécifié

Adresse personnelle & de messagerie électronique à contacter pour des informations complémentaires :
Magnus Nystrom <magnus@rsasecurity.com>

Utilisation prévue : COMMUNE
Contrôleur des modifications : IESG <iesg@ietf.org>

9 Remerciements

Les auteurs souhaitent remercier le groupe de travail TLS et le groupe de travail WAP Sécurité. Le présent document se fonde sur les discussions menées au sein de ces groupes.

10 Références normatives

[HMAC] H Krawczyk, M. Bellare, et R. Canetti, "HMAC : Hachage de clés pour l'authentification de message", RFC 2104, février 1997.

[HTTP] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, et T. Berners-Lee, "Protocole de transfert hypertexte -- HTTP/1.1", RFC 2616, juin 1999.

[IANA] T. Narten, et H. Alvestrand, "Lignes directrices pour écrire une section de Considérations relatives à l'IANA dans les RFC", BCP 26, RFC 2434, octobre 1998.

[IDNA] P. Faltstrom, P. Hoffman, et A. Costello, "Internationalisation des noms de domaine dans les applications (IDNA)", RFC 3490, mars 2003.

[KEYWORDS] S. Bradner, "Mots clé à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, RFC 2119, mars 1997.

[OCSP] M. Myers, R. Ankney, A. Malpani, S. Galperin, et C. Adams, "Protocole d'état de certificat en ligne (OCSP) de l'infrastructure de clé publique X.509 pour l'Internet", RFC 2560, June 1999.

[PKIOP] R. Housley, et P. Hoffman, "Protocoles opérationnels d'infrastructure de clé publique X.509 pour l'Internet : FTP et HTTP", RFC 2585, mai 1999.

[PKIX] R. Housley, W. Polk, W. Ford, et D. Solo, "Profil de certificat de clé publique et de liste de révocation de clé publique X.509 pour l'Internet", RFC 3280, avril 2002.

[TLS] T. Dierks, et C. Allen, "Protocole TLS, version 1.0", RFC 2246, janvier 1999.

[TLSbis] T. Dierks, et E. Rescorla, "Protocole de sécurité de la couche Transport (TLS) version 1.1", RFC 4346, avril 2006.

[URI] T. Berners-Lee, R. Fielding, et L. Masinter, "Identifiant uniforme de ressource (URI) : Syntaxe générique", STD 66, RFC 3986, janvier 2005.

[UTF8] F. Yergeau, "UTF-8, un format de transformation de ISO 10646", STD 63, RFC 3629, novembre 2003.

[X509-4th] Recommandation UIT-T X.509 (2000) | ISO/CEI 9594-8:2001, "Systèmes d'information - Interconnexion des systèmes ouverts – L'annuaire : Cadre de travail des clés publiques et des certificats d'attribut."

[X509-4th-TC1] Recommandation UIT-T X.509 (2000) Corrigendum 1 (2001) | ISO/CEI 9594-8:2001/Cor.1:2002, Corrigendum Technique 1 à ISO/CEI 9594:8:2001.

11 Références informatives

[AESSUITES] P. Chown, "Suites de chiffrement de la norme de chiffrement évolué (AES) pour la sécurité de la couche Transport (TLS)", RFC 3268, juin 2002.

[KERB] A. Medvinsky, et M. Hur, "Ajout de suitesde chiffrement Kerberos à la sécurité de couche Transport (TLS)", RFC 2712, octobre 1999.

[MAILINGLIST] J. Mikkelsen, R. Eberhard, et J. Kistler, "Mécanisme général d'extension de prise de contact client et hébergement virtuel," ietf-tls mailing list posting, 14 août 2000.

[RFC3546] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, et T. Wright, "Extensions à la sécurité de couche Transport (TLS)", RFC 3546, juin 2003.

Adresse des auteurs

Simon Blake-Wilson	Magnus Nystrom	David Hopwood
BCI	RSA Security	Independent Consultant
mél : sblakewilson@bcisse.com	mél : magnus@rsasecurity.com	mél : david.hopwood@blueyonder.co.uk

Jan Mikkelsen	Tim Wright
Transactionware	Vodafone
mél : janm@transactionware.com	mél : timothy.wright@vodafone.com

Déclaration de copyright

Propriété intellectuelle

Remerciement

Le financement de la fonction d'édition des RFC est actuellement fourni par l'activité de soutien administratif de l'IETF (IASA, *Administrative Support Activity*).