Groupe de travail Réseau
Request for Comments : 2847
Catégorie : En cours de normalisation

M. Eisler, Zambeel
juin 2000
Traduction Claude Brière de L'Isle

LIPKEY – mécanisme de faible infrastructure de clé publique utilisant SPKM

Statut de ce mémoire
Le présent document spécifie un protocole Internet en cours de normalisation pour la communauté de l'Internet, et appelle à des discussions et des suggestions d'amélioration. Prière de se référer à l'édition en cours des "Normes officielles des protocoles de l'Internet" (STD 1) pour connaître l'état de normalisation et le statut de ce protocole. La distribution du présent mémoire n'est soumise à aucune restriction.

Résumé
Ce mémoire décrit une méthode par laquelle on peut utiliser GSS-API [RFC2078] pour fournir un canal sûr entre un client et un serveur, authentifiant le client avec un mot de passe, et un serveur avec un certificat de clé publique. À ce titre, il est analogue à l'usage commun de faible infrastructure du protocole de sécurité de la couche transport (TLS, *Transport Layer Security*) de la [RFC2246].

La méthode s'appuie sur le mécanisme de simple clé publique (SPKM, *Simple Public Key Mechanism*) [RFC2025], et est spécifié dans un mécanisme GSS-API séparé (LIPKEY) mis en couche par dessus SPKM.

Table des Matières

1. Introduction....1
2. Exigences de LIPKEY pour SPKM....2
  2.1 Type de mécanisme....2
  2.2 Type de nom....3
  2.3 Algorithmes....3
  2.4 Jetons d'établissement de contexte....5
  2.5 Qualité de protection....6
3. Comment LIPKEY utilise SPKM....6
  3.1 Jetons....6
  3.2 Initiateur....6
  3.3 Cible....8
4. Description de LIPKEY....9
  4.1 Type de mécanisme....9
  4.2 Types de nom....9
  4.3 Formats de jeton....9
  4.4 Qualité de protection....10
5. Considérations sur la sécurité....10
  5.1 Gestion des mots de passe....10
  5.2 Autorités de certification....11
  5.3 Faiblesses de HMAC-MD5 et de MD5....11
  5.4 Sécurité de cast5CBC....11
Références....11
Remerciements....12
Adresse de l'auteur....12
Déclaration complète de droits de reproduction....13

1. Introduction

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

Le présent mémoire décrit un nouveau mécanisme de sécurité sous la GSS-API appelé mécanisme de faible infrastructure de clé publique (LIPKEY, *Low Infrastructure Public Key*). GSS-API fournit le moyen pour qu'un protocole d'application mette en œuvre l'authentification, la protection de l'intégrité, et de la confidentialité. TLS est un autre moyen. Alors que TLS est de nombreuses façons plus simple à incorporer pour une application que GSS-API, il y a des situations où GSS-API peut mieux convenir. C'est certainement le cas avec des protocoles d'application qui fonctionnent sur des protocoles sans connexion. C'est aussi le cas avec des protocoles d'application tels que ONC RPC [RFC1831], [RFC2203], qui ont leur propre architecture de sécurité, et ne se mêlent pas aisément avec un protocole comme TLS qui est mis en œuvre comme une couche qui encapsule le protocole d'application de couche supérieure. GSS-API permet au protocole d'application d'encapsuler autant du protocole d'application qu'il est nécessaire.

En dépit de la souplesse de GSS-API, la comparaison est défavorable avec TLS à l'égard de la perception de la quantité d'infrastructure exigée pour le déployer. Les mécanismes mieux connus de GSS-API, Kerberos V5 [RFC1964] et SPKM exigent pour leur établissement une grande quantité d'infrastructure. Comparer cela au scénario typique de déploiement de TLS, qui consiste en un client sans certificat de clé publique qui accède à un serveur avec un certificat de clé publique. Le client :

* obtient le certificat du serveur,
* vérifie qu'il a été signé par une autorité de certification (CA) de confiance,
* génère une clé symétrique aléatoire de session,
* chiffre la clé de session avec la clé publique du serveur, et
* envoie la clé de session chiffrée au serveur.

À ce point, le client et le serveur ont un canal sûr. Le client peut alors fournir un nom d'utilisateur et un mot de passe au serveur pour authentifier le client. Par exemple, lorsque TLS est utilisé avec le protocole http, une fois qu'il y a un canal sûr, le serveur http va présenter au client une page html qui invite à fournir un nom d'utilisateur et un mot de passe. Ces informations sont ensuite chiffrées avec la clé de session et envoyées au serveur. Le serveur authentifie ensuite le client.

Noter que le client n'est pas obligé d'avoir un certificat pour lui-même s'identifier et s'authentifier auprès du serveur. En plus d'une mise en œuvre de TLS, l'infrastructure de sécurité requise comporte un certificat de clé publique et une base de données de mots de passe sur le serveur, et une liste d'autorités de certification (CA) de confiance et leurs clés publiques sur le client. La plupart des systèmes d'exploitation sur lesquels le serveur http va fonctionner ont déjà une base de données de mots de passe native, de sorte que l'infrastructure supplémentaire nette est un certificat de serveur et une liste de CA. D'où le terme de "modèle de sécurité à faible infrastructure" pour identifier ce scénario typique de déploiement de TLS.

En utilisant l'authentification unilatérale, et en utilisant un mécanisme ressemblant au type de mécanisme SPKM-1, SPKM peut offrir de nombreux aspects du modèle de sécurité à faible infrastructure décrit précédemment. Une application qui utilise GSS-API est certainement libre d'utiliser le sous-programme GSS_Wrap() de GSS-API pour chiffrer le nom et le mot de passe d'un usager et les envoyer au serveur, pour qu'il les déchiffre et les vérifie.

Les applications ont souvent des protocoles d'application associés, et il peut n'y avoir aucune disposition dans le protocole pour spécifier un mot de passe. La mise en couche d'un fin mécanisme de GSS-API sur un mécanisme ressemblant à SPKM-1 peut atténuer ce problème. Ce peut être une approche utile pour éviter de modifier des applications qui sont déjà liées à GSS-API, en supposant que les applications ne sont pas liées statiquement à des mécanismes GSS-API spécifiques. Le reste du mémoire définit le mécanisme de clé publique à faible infrastructure (LIPKEY, *Low Infrastructure Public Key*).

2. Exigences de LIPKEY pour SPKM

SPKM-1 avec authentification unilatérale est proche du modèle de faible infrastructure désirée décrit plus haut. Cette section décrit des changements supplémentaires de la façon dont SPKM-1 opère afin de réaliser le modèle de faible infrastructure. Ces changements incluent des changements mineurs de sémantique. Alors qu'il serait possible de mettre en œuvre ces changements de sémantique au sein d'une mise en œuvre de SPKM-1 (incluant d'utiliser le même identifiant d'objet (OID, *Object Identifier*) de type de mécanisme que SPKM-1) l'ensemble de changements contraint l'interprétation de la RFC2025 à un point tel que la compatibilité serait en danger. Un nouveau type de mécanisme, appelé SPKM-3, est justifié. LIPKEY exige que la mise en œuvre de SPKM prenne en charge SPKM-3. SPKM-3 est équivalent à SPKM-1, excepté comme décrit dans le reste de cette section.

2.1 Type de mécanisme

SPKM-3 a un OID de type de mécanisme différent de celui de SPKM-1.

IDENTIFIANT D'OBJET spkm-3 ::= {iso(1)identified-organization(3)dod(6)internet(1)security(5) mechanisms(5) spkm(1)spkm-3(3)}

2.2 Type de nom

La [RFC2025] ne définit pas de type de nom exigé de SPKM. LIPKEY exige que la mise en œuvre de SPKM-3 prenne en charge tous les types de noms indépendants du mécanisme, dans la [RFC2078].

2.3 Algorithmes

2.3.1 Algorithmes obligatoires

La [RFC2025] définit divers algorithmes pour la protection de l'intégrité, de la confidentialité, l'établissement de clés, et la déduction de sous clés. Sauf pour md5WithRSAEncryption, les algorithmes, d'établissement de clé EXIGÉ (K-ALG), d'intégrité (I-ALG) et de fonctions unidirectionnelles pour la déduction de sous clés (O-ALG) mentionnés dans la [RFC2025] continuent d'être EXIGÉS.

SPKM est conçu pour être extensible à l'égard des nouveaux algorithmes. Afin que LIPKEY fonctionne correctement et en toute sécurité, les algorithmes suivants DOIVENT être mis en œuvre dans SPKM-3:

* **Algorithmes d'intégrité (I-ALG)**

NULL-MAC
Parce qu'il se peut que l'initiateur n'ait pas de certificat pour lui-même, ni pour la cible, il ne lui est pas possible de calculer une valeur d'intégrité dans le REQ-TOKEN *(jeton de demande)* de l'initiateur qui est envoyé à la cible. On définit donc, en syntaxe ASN.1 [CCITT], un I-ALG nul qui retourne une chaîne de bits de longueur zéro sans considération de l'entrée qui lui est passée :

IDENTIFIANT D'OBJET NULL-MAC ::= {iso(1) identified-organization(3) dod(6) internet(1) security(5) integrity(3) NULL-MAC(3)}

id-dsa-with-sha1
C'est l'algorithme de signature el que défini au paragraphe 7.2.2 de la [RFC2459]. L'OID ASN.1 qui y est utilisé pour identifier cet algorithme de signature est :

IDENTIFIANT D'OBJET id-dsa-with-sha1 ::= {iso(1) member-body(2) us(840) x9-57(10040)x9cm(4) 3}

Noter que la [RFC3280] rend obsolète la [RFC2459]. Elle ne change cependant pas la définition de id-dsa-with-sha1.

HMAC-MD5
Une conséquence de ce que l'initiateur SPKM-3 n' a pas de certificat est qu'il ne peut pas utiliser un algorithme de signature numérique comme md5WithRSAEncryption, id-dsa-with-sha1, ou sha1WithRSAEncryption une fois que le contexte est établi. Au lieu de cela, un algorithme de code d'authentification de message (MAC, *message authentication code*) est nécessaire. DES-MAC est spécifié comme recommandé dans la [RFC2025]. Comme il a été prouvé que la sécurité de DES à 56 bits n'est pas adéquate [EFF], SPKM-3 a besoin d'un MAC plus fort. Donc, SPKM-3 DOIT prendre en charge l'algorithme HMAC-MD5 [RFC2104], avec cet OID :

IDENTIFIANT D'OBJET HMAC-MD5 ::= {iso(1) org(3) dod(6) internet(1) security(5) mechanisms(5) ipsec(8) isakmpOakley(1)1}

La référence de l'OID d'algorithme de HMAC-MD5 est [IANA]. La référence pour l'algorithme HMAC-MD5 est la [RFC2104].

L'algorithme HMAC-SHA1 n'est pas un MAC d'I-ALG obligatoire de SPKM-3 parce que SHA-1 a environ la moitié de la vitesse de MD5 [Young]. Un MAC fondé sur un algorithme de chiffrement comme cast5CBC, DES EDE3, ou RC4 n'est pas obligatoire parce que MD5 est 31 % plus rapide que le plus rapide des trois algorithmes de chiffrement [Young].

* **Algorithmes de confidentialité (C-ALG).**

La [RFC2025] n'a pas d'algorithme de confidentialité obligatoire, et a à la place un algorithme DES de 56 bits RECOMMANDÉ. Comme l'initiateur LIPKEY a besoin d'envoyer un mot de passe à la cible, et comme il a été prouvé que le DES à 56 bits est inadéquat [EFF], LIPKEY a besoin d'un chiffrement plus fort. Donc, SPKM-3 DOIT prendre en charge l'algorithme suivant :

```
IDENTIFIANT D'OBJET cast5CBC ::= {iso(1) memberBody(2) usa(840) nt(113533) nsn(7)algorithms(66) 10}
Paramètres ::= SEQUENCE {
    iv CHAINE D'OCTETS DEFAULT 0,                -- Vecteur d'initialisation
    keyLength ENTIER                             -- Longueur de clé, en bits
}
```

La référence pour l'OID et la description de l'algorithme cast5CBC est la [RFC2144]. Dans les paramètres, la longueur de clé DOIT être réglée à 128 bits.

Un algorithme triple DES (DES EDE3) n'est pas un C-ALG SPKM-3 obligatoire parce qu'il est beaucoup plus lent que cast5CBC. Un ensemble de mesures [Young] sur un processeur Pentium Pro 200 Mhz utilisant le code SSLeay a montré que DES EDE3 effectuait jusqu'à 1 646 210 octets par seconde, en utilisant des blocs de 1024 octets. Le même cadre d'essai a donné une performance de 7 147 760 octets par seconde pour cast5CBC, et de 22 419 840 octets par seconde pour RC4. La plupart des sessions TLS négocient le chiffrement RC4. Étant donné que LIPKEY a pour cible des environnements similaires à ceux où TLS est déployé, choisir un chiffrement qui est plus de 13 fois plus lent (et plus de 13 fois plus consommateur de CPU) que RC4 amoindrirait sévèrement l'utilité de LIPKEY. Pour des raisons de performances, RC4 serait l'algorithme obligatoire préféré pour SPKM-3. Du fait de considérations de propriété intellectuelle avec RC4 [Schneier], la combinaison des performances raisonnables de cast5CBC et de ses conditions de licence libre de redevance [RFC2144] rend cast5CBC le choix optimal entre DES EDE3, RC4, et cast5CBC.

* **Algorithme d'établissement de clé (K-ALG)**

La [RFC2025] mentionne dhKeyAgreement [PKCS-3] comme un algorithme apparemment facultatif. Comme on le décrit plus loin, l'algorithme d'établissement de clé RSAEncryption n'est d'aucune utilité pour un mécanisme de sécurité à faible infrastructure comme défini par le présent mémoire. Donc, dans SPKM-3, dhKeyAgreement est un algorithme d'établissement de clé EXIGÉ :

```
IDENTIFIANT D'OBJET dhKeyAgreement  ::= {iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) pkcs-3(3) 1 }
```

* **Fonction unidirectionnelle pour algorithme de déduction de clé (O-ALG)**

La [RFC2025] mentionne MD5 comme algorithme obligatoire. Comme MD5 s'est révélé avoir des faiblesses lorsque utilisé comme hachage [Dobbertin], id-sha1 est un O-ALG EXIGÉ dans SPKM-3 :

```
IDENTIFIANT D'OBJET id-sha1 ::= { iso(1) identified-organization(3) oiw(14) secsig(3) algorithms(2) 26 }
```

La référence de l'OID d'algorithme de id-sha1 est la [RFC2437]. La référence pour l'algorithme SHA-1 correspondant à id-sha1 est [FIPS].

2.3.2 Algorithmes d'intégrité (I-ALG) RECOMMANDÉS

md5WithRSAEncryption

L'algorithme d'intégrité md5WithRSAEncryption est mentionné dans la [RFC2025] comme obligatoire. Du fait des considérations de propriété intellectuelle [RSA-IP], les mises en œuvre de SPKM-3 ne peuvent pas être obligées à le mettre en œuvre. Cependant, étant donnée la prolifération de certificats qui utilisent les clés publiques RSA, md5WithRSAEncryption est fortement RECOMMANDÉ. Autrement, les opportunités pour que LIPKEY démultiplie l'infrastructure de clé publique existante seront limitées.

sha1WithRSAEncryption

Pour des raisons similaires à celles de md5WithRSAEncryption, sha1WithRSAEncryption est un algorithme RECOMMANDÉ. L'algorithme sha1WithRSAEncryption est mentionné en plus de md5WithRSAEncryption à cause des faiblesses de l'algorithme de hachage MD5 [Dobbertin]. L'OID pour sha1WithRSAEncryption est :

```
IDENTIFIANT D'OBJET sha1WithRSAEncryption ::= { iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1)
                                                pkcs-1(1) 5 }
```

La [RFC2437] est la référence de l'OID d'algorithme et contient la description de sha1WithRSAEncryption.

2.4 Jetons d'établissement de contexte

La [RFC2025] établit un contexte avec le premier jeton d'un initiateur (REQ-TOKEN), une réponse de la cible (REP-TI-TOKEN) et finalement le second jeton d'un initiateur (REP-IT-TOKEN) pour répliquer à la réponse de la cible. Comme LIPKEY utilise SPKM-3 avec authentification unilatérale, REP-IT-TOKEN n'est pas utilisé. LIPKEY a certaines exigences sur le contenu de REQ-TOKEN et REP-TI-TOKEN, mais la syntaxe de jetons SPKM-3 n'est pas différente de celle des jetons SPKM-1 de la [RFC2025].

2.4.1 Exigences de contenu de REQ-TOKEN

2.4.1.1 algId et req-integrity

Si l'initiateur SPKM-3 ne peut pas calculer un champ req-integrity à cause de l'absence d'un certificat de la cible, il DOIT utiliser l'I-ALG NULL-MAC décrit plus haut dans le présent mémoire. Cela va produire une chaîne de longueur zéro dans le champ Intégrité.

2.4.1.2 Req-contents

Parce que la [RFC2025] exige que le K-ALG RSAEncryption soit présent, SPKM-1 doit être capable de transposer la cible (targ-name) en son certificat de clé publique, et donc SPKM peut utiliser l'algorithme RSAEncryption pour remplir le champ key-estb-req. Parce que LIPKEY suppose le déploiement d'une faible infrastructure, SPKM-3 DOIT être prêt à être incapable de transposer le champ targ-name du champ Req-contents. C'est une contradiction qui se résout en exigeant que SPKM-3 prenne en charge l'algorithme dhKeyAgreement. Noter que si une mise en œuvre de SPKM-3 essaye de transposer la cible en un certificat, et y réussit, elle est libre d'utiliser l'algorithme K-ALG RSAEncryption. Elle est aussi libre d'utiliser un algID autre que NULL-MAC dans le type REQ-TOKEN.

2.4.1.2.1 Options

Une mise en œuvre de SPKM-3 DOIT établir le bit target-certif-data-required à 1 si le seul K-ALG dans le champ key-estb-set de Req-contents est dhKeyAgreement. Cela va normalement se produire si la mise en œuvre de SPKM-3 ne peut pas résoudre le nom de la cible en certificat.

2.4.1.2.2 Conf-Algs

Si la mise en œuvre de SPKM-3 prend en charge un algorithme plus faible que cast5CBC, cast5CBC DOIT être mentionné avant l'algorithme plus faible pour encourager la cible à négocier le plus fort algorithme.

2.4.1.2.3 Intg-Algs

Parce que l'initiateur sera anonyme (au niveau SPKM-3) et n'aura pas de certificat pour lui-même, l'initiateur ne peut pas utiliser un algorithme d'intégrité qui prend en charge la non répudiation ; il doit utiliser un algorithme de MAC. Si la mise en œuvre de SPKM-3 prend en charge un algorithme plus faible que HMAC-MD5, HMAC-MD5 DOIT être mentionné avant l'algorithme plus faible pour encourager la cible à négocier l'algorithme plus fort.

2.4.2 Exigences de contenu de REP-TI-TOKEN

Avec les exigences décrites précédemment sur REQ-TOKEN, le contenu du REP-TI-TOKEN de SPKM-3 peut pour sa majeure partie être déduit de la spécification de la [RFC2025]. Les exceptions sont les champs algId et rep-ti-integ.

2.4.2.1 algId

La cible SPKM-3 NE DOIT PAS utiliser un I-ALG NULL-MAC ; elle DOIT utiliser un algorithme de signature comme id-dsa-with-sha1, md5WithRSAEncryption, ou sha1WithRSAEncryption.

2.4.2.2 rep-ti-integ

Si le req-token a un algId de NULL-MAC, la cible DOIT alors calculer le rep-ti-integ sur l'enchaînement de req-contents et de rep-ti-contents.

2.5 Qualité de protection

L'initiateur et la cible SPKM-3 négocient l'ensemble d'algorithmes qu'ils prennent mutuellement en charge, en utilisant la procédure définie au paragraphe 5.2 de la [RFC2025]. Si une qualité de protection de zéro est spécifiée, l'initiateur et la cible vont alors utiliser les premiers algorithmes C-ALG (confidentialité) et I-ALG (intégrité) négociés.

SPKM sépare la qualité de protection en plusieurs champs, comme reproduit ici d'après le paragraphe 5.2 de la RFC2025 :

```
 Confidentialité                    Intégrité
 31 (MSB)                         16 15                  (LSB) 0
-------------------------------|-------------------------------
| TS(5) | U(3) | IA(4) | MA(4) | TS(5) | U(3) | IA(4) | MA(4) |
-------------------------------|-------------------------------
```

Les sous champs MA *(Message Authentication,* authentification de message*)* énumèrent les algorithmes définis par des mécanismes. Comme le présent mémoire introduit un nouveau mécanisme, SPKM-3, au sein de la famille SPKM, il est approprié d'ajouter des algorithmes aux sous champs MA des champs respectifs de Confidentialité et d'Intégrité.

L'ensemble complet des algorithmes de MA de confidentialité est donc :

0001 (1) = DES-CBC
0010 (2) = cast5CBC

Où "0001" et "0010" sont en base 2. Un homologue SPKM qui négocie une valeur d'algorithme de MA de confidentialité de "0010" DOIT utiliser une clé de 128 bits, c'est-à-dire, établir les valeurs de keyLength *(longueur de clé)* dans les paramètres cast5CBC à 128 bits.

L'ensemble complet d'algorithmes de MA d'intégrité est donc :

0001 (1) = md5WithRSAEncryption
0010 (2) = DES-MAC
0011 (3) = id-dsa-with-sha1
0100 (4) = HMAC-MD5
0101 (5) = sha1WithRSAEncryption

Où "0001" à "0101" sont en base 2.

L'ajout de la prise en charge de cast5CBC, id-dsa-with-sha1, HMAC-MD5, et sha1WithRSAEncryption de la manière ci-dessus à SPKM-1 et SPKM-2 ne réduit pas la rétro compatibilité de SPKM-1 et SPKM-2 parce que, comme on l'a noté précédemment, SPKM négocie les algorithmes. Un SPKM-1 ou SPKM-2 plus ancien qui ne reconnaît pas les valeurs de MA pour cast5CBC, id-dsa-with-sha1, HMAC-MD5, ou sha1WithRSAEncryption ne les choisira pas.

3. Comment LIPKEY utilise SPKM

3.1 Jetons

LIPKEY va invoquer SPKM-3 pour produire des jetons SPKM. Comme le mécanisme qu'utilise l'application est LIPKEY, LIPKEY va envelopper certains jetons SPKM-3 avec des préfixes LIPKEY. La définition exacte des jetons est décrite plus loin dans le présent mémoire.

3.2 Initiateur

3.2.1 GSS_Import_name

L'initiateur utilise GSS_Import_name pour importer le nom de la cible, normalement, mais pas nécessairement, en utilisant le type de nom GSS_C_NT_HOSTBASED_SERVICE. Finalement, le résultat de GSS_Import_name va s'applique à un type de mécanisme SPKM-3 parce qu'une cible LIPKEY est une cible SPKM-3.

3.2.2 GSS_Acquire_cred

L'initiateur invoque GSS_Acquire_cred. Les accréditifs qui sont acquis sont des accréditifs LIPKEY, un nom d'utilisateur et un mot de passe. Comment le nom d'utilisateur et le mot de passe sont acquis dépend de l'environnement de fonctionnement. Une application qui invoque GSS_Acquire_cred() tandis que l'utilisateur de l'application a une interface d'utilisateur graphique qui fonctionne peut déclancher l'apparition d'une fenêtre incrustée qui invite aux informations. Une application incorporée dans le système d'exploitation, telle qu'un client NFS [Sandberg] mise en œuvre comme un fichier système natif pourrait diffuser un message au terminal de l'utilisateur pour lui dire d'invoquer une commande qui invite aux informations.

Parce que des accréditifs ne seront pas utilisés jusqu'à ce que GSS_Init_sec_context soit invoqué, la mise en œuvre de LIPKEY va avoir besoin de sauvegarder les accréditifs. Si cela pose problème la mise en œuvre peut à la place différer l'acquisition réelle du nom d'utilisateur et le mot de passe jusqu'à ce que GSS_init_sec_context soit prêt à envoyer le nom d'utilisateur et le mot de passe à la cible. Dans ce cas, l'argument output_cred_handle de GSS_Acquire_cred va simplement être une référence qui se transpose en le principal qui correspond à l'argument desired_name. Une invocation suivante de GSS_Init_sec_context prendra en considération la transposition de claimant_cred_handle en principal lorsque elle acquerra le nom d'utilisateur et le mot de passe. Par exemple, la fenêtre incrustée susmentionnée pourrait remplir la portion nom d'utilisateur du dialogue avec une valeur par défaut qui se transpose en le principal auquel se réfère le claimant_cred_handle.

3.2.3 GSS_Init_sec_context

Lorsque un programme invoque GSS_Init_sec_context sur le type de mécanisme LIPKEY, si la bride de contexte est NULL, le mécanisme LIPKEY va à son tour invoquer GSS_Init_sec_context sur un mécanisme SPKM-3 mis en œuvre conformément aux exigences décrites précédemment. Cette invocation de SPKM-3 DOIT avoir les attributs suivants :

* claimant_cred_handle est NUL
* mutual_req_flag est FAUX
* anon_req_flag est VRAI
* input_token est NUL
* mech_type est l'OID du mécanisme SPKM-3

On se souviendra que les attributs ci-dessus sont dans l'invocation GSS_Init_sec_context depuis le mécanisme LIPKEY jusqu'au mécanisme SPKM-3. Il n'y a pas de restriction particulière sur l'invocation par l'application du sous programme GSS_Init_sec_context de LIPKEY. Tous les autres arguments sont déduits des arguments du GSS_Init_sec_context de LIPKEY.

L'invocation de SPKM-3 GSS_Init_sec_context va créer une bride de contexte SPKM-3. Le reste de la description de l'invocation du GSS_Init_sec_context de LIPKEY dépend du réglage de anon_req_flag à VRAI ou FAUX par celui qui invoque le GSS_Init_sec_context LIPKEY.

3.2.3.1 L'invocateur LIPKEY spécifie anon_req_flag à VRAI

Si l'invocateur du GSS_Init_sec_context de LIPKEY règle anon_req_flag à VRAI, il DOIT retourner à l'appelant LIPKEY tous les résultats de l'invocation SPKM-3 GSS_Init_sec_context, incluant output_context_handle, output_token, et mech_type. De cette façon, LIPKEY "sort maintenant des chemins" du traitement de GSS-API entre l'application et SPKM-3, parce que rien dans les résultats retournés ne se rapporte à LIPKEY. Ceci est nécessaire, parce que les jetons de contexte LIPKEY n'ont aucune disposition pour spécifier des initiateurs anonymes. C'est parce que SPKM-3 est suffisant pour les besoins de la prise en charge d'initiateurs anonymes dans un environnement de faible infrastructure.

En clair, lorsque l'appelant LIPKEY désire une authentification anonyme, LIPKEY n'a aucune valeur ajoutée, mais il est plus simple de prendre en charge le dispositif que d'inviter l'appelant à utiliser directement SPKM-3.

Si tout va bien, il sera retourné à l'appelant de LIPKEY un major_status de GSS_S_CONTINUE_NEEDED via SPKM-3, et ainsi l'appelant de LIPKEY va envoyer le output_token à la cible. L'appelant de LIPKEY reçoit alors le jeton de réponse de la cible, et invoque directement le SPKM-3 GSS_Init_sec_context. Au retour, le major_status devrait être GSS_S_COMPLETE.

3.2.3.2 L'invocateur LIPKEY spécifie anon_req_flag à FAUX

Le mécanisme LIPKEY va avoir besoin d'allouer une bride de contexte pour lui-même, et d'enregistrer dans la bride de contexte LIPKEY la bride de contexte SPKM-3 qui a été retournée dans le paramètre output_context_handle provenant de l'invocation du sous-programme SPKM-3 GSS_Init_sec_context. Le sous-programme LIPKEY GSS_Init_sec_context va

retourner dans output_context_handle la bride de contexte LIPKEY, et dans mech_type, le type de mécanisme LIPKEY. Le output_token est comme défini plus loin dans le présent mémoire, au paragraphe intitulé "Jetons de contexte avant l'établissement du contexte SPKM-3". Tous les autres résultats retournés seront ceux que le sous-programme SPKM-3 GSS_Init_sec_context a retourné à LIPKEY. Si tout s'est bien passé, le mécanisme SPKM-3 aura retourné un major_status de GSS_S_CONTINUE_NEEDED.

L'invocateur du sous-programme LIPKEY GSS_Init_sec_context va voir un major_status de GSS_S_CONTINUE_NEEDED, et donc l'invocateur de LIPKEY va envoyer le output_token à la cible. Il va alors recevoir le jeton de réponse de la cible, et invoquer le sous-programme LIPKEY GSS_Init_sec_context pour la seconde fois. LIPKEY invoque alors le GSS_Init_sec_context SPKM-3 pour la seconde fois et au retour, le major_status devrait être GSS_S_COMPLETE.

Alors que l'établissement de contexte de SPKM-3 est maintenant achevé, celui de LIPKEY ne l'est pas encore, parce que l'initiateur doit envoyer à la cible le nom d'utilisateur et le mot de passe qui lui ont été passés via la claimant_cred_handle sur la première invocation du sous-programme LIPKEY GSS_Init_sec_context. LIPKEY utilise la bride de contexte SPKM-3 établie comme entrée à GSS_Wrap (avec conf_req_flag réglé à VRAI) pour chiffrer ce à quoi se réfère la claimant_cred_handle (nom d'utilisateur et mot de passe) et retourne cela comme output_token à l'appelant de LIPKEY (pourvu que le résultat conf_state de l'appel au GSS_Wrap de SPKM-3 soit VRAI) avec un major_status de GSS_S_CONTINUE_NEEDED.

L'invocateur de LIPKEY envoie son second jeton d'établissement de contexte à la cible, et attend qu'un jeton soit fourni par la le sous-programme GSS_Accept_sec_context de la cible. Le sous-programme LIPKEY GSS_Accept_sec_context de la cible invoque le sous programme SPKM-3 GSS_Unwrap sur le jeton, et valide le nom d'utilisateur et le mot de passe. La cible invoque alors le sous-programme GSS_Wrap de SPKM-3 sur un booléen indiquant si le nom d'utilisateur et le mot de passe ont été acceptés ou non, et retourne le résultat output_message provenant de GSS_Wrap comme résultat output_token pour GSS_Accept_sec_context.

L'invocateur de LIPKEY reçoit le jeton de réponse de la cible, et le passe au sous-programme LIPKEY GSS_Init_sec_context via le paramètre input_token. LIPKEY invoque alors GSS_Unwrap pour obtenir l'indication booléenne d'acceptation, et transpose ceci en un major_status de GSS_S_COMPLETE indiquant la réussite (le booléen est VRAI) et l'achèvement de l'établissement de contexte LIPKEY, ou de GSS_S_FAILURE, indiquant que l'établissement de contexte a échoué. GSS_S_CONTINUE_NEEDED ne sera pas retourné.

Noter que le paramètre mutual_req_flag est ignoré parce que l'authentification unilatérale est impossible. L'initiateur doit authentifier la cible via SPKM-3 afin de créer un canal sûr pour transmettre le nom d'utilisateur et le mot de passe. La cible doit authentifier l'initiateur lorsque elle reçoit le nom d'utilisateur et le mot de passe.

Le contexte SPKM-3 reste établi tandis que le contexte LIPKEY est établi. Si le contexte SPKM-3 expire avant que le contexte LIPKEY soit détruit, la mise en œuvre LIPKEY devrait faire expirer le contexte LIPKEY et retourner l'erreur appropriée sur la prochaine opération GSS-API.

3.2.4 Autres opérations

Pour les autres opérations, le contexte LIPKEY agit comme une passerelle vers le contexte SPKM-3. Les opérations qui affectent l'état de contexte ou l'interrogent, comme GSS_Delete_sec_context, GSS_Export_sec_context, GSS_Import_sec_context, et GSS_Inquire_context, vont exiger une passerelle vers le contexte SPKM-3 et une modification d'état au contexte LIPKEY.

3.3 Cible

3.3.1 GSS_Import_name

Comme avec l'initiateur, le nom importé sera celui de la cible.

3.3.2 GSS_Acquire_cred

La cible invoque le sous-programme LIPKEY GSS_Acquire_cred pour obtenir un accréditif pour une cible SPKM-3, via le sous-programme SPKM-3 GSS_Acquire_cred. Le desired_name est le output_name provenant de GSS_Import_name.

3.3.3 GSS_Accept_sec_context

Lorsque un programme invoque GSS_Accept_sec_context sur le type de mécanisme LIPKEY, si la bride de contexte est NULL, le mécanisme LIPKEY va à son tour invoquer GSS_Accept_sec_context sur un mécanisme SPKM-3 mis en œuvre conformément aux exigences décrites précédemment. Cette invocation de SPKM-3 n'est pas différente de ce à quoi on s'attendrait avec une invocation mise en couche à GSS_Accept_sec_context.

Si tout va bien, l'invocation de SPKM-3 GSS_Accept_sec_context va réussir avec GSS_S_COMPLETE, et l'invocation de LIPKEY GSS_Accept_sec_context va retourner le output_token à l'appelant, mais avec un major_status de GSS_S_CONTINUE_NEEDED parce que l'initiateur LIPKEY est toujours supposé envoyer le nom d'utilisateur et le mot de passe.

Une fois que le contexte SPKM-3 est dans un état GSS_S_COMPLETE, le prochain jeton que reçoit la cible va contenir le nom d'utilisateur et le mot de passe, enveloppés par le résultat d'une invocation de SPKM-3 GSS_Wrap. La cible invoque le LIPKEY GSS_Accept_sec_context, qui à son tour invoque le sous-programme SPKM-3 GSS_Unwrap. Le sous-programme LIPKEY GSS_Accept_sec_context compare alors le nom d'utilisateur et le mot de passe avec ceux de sa base de données. Si le nom d'utilisateur et le mot de passe de l'initiateur sont valides, GSS_S_COMPLETE est retourné à l'appelant. Autrement, GSS_S_FAILURE est retourné. Dans l'un et l'autre cas, un output_token - égal au résultat output_message provenant d'une invocation SPKM-3 GSS_Wrap sur une valeur booléenne - est retourné à l'appelant. La valeur booléenne est réglée à VRAI si le nom d'utilisateur et le mot de passe étaient valides, FAUX autrement. La cible n'attend pas d'autre jeton d'établissement de contexte de la part de l'appelant.

4. Description de LIPKEY

4.1 Type de mécanisme

IDENTIFIANT D'OBJET lipkey ::= {iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) lipkey(9)}

4.2 Types de nom

LIPKEY utilise seulement les types de noms indépendants du mécanisme définis dans la [RFC2078]. Tous les types de noms définis dans la [RFC2078] sont EXIGÉS.

4.3 Formats de jeton

4.3.1 Jetons de contexte

GSS-API définit les jetons de contexte comme :

```
InitialContextToken ::=                 -- indication d'option (délégation, etc.) au sein du jeton spécifique de mécanisme
    [APPLICATION 0] IMPLICIT SEQUENCE {
        thisMech MechType,
        innerContextToken ANY DEFINED BY thisMech
                                        -- une structure de contenu ASN.1 spécifique du mécanisme n'est pas exigée
    }

SubsequentContextToken ::= innerContextToken ANY
                                -- interprétation fondée sur le InitialContextToken précédent
                                -- structure ASN.1 non exigée
```

Le contenu du innerContextToken dépend de si le contexte SPKM-3 est établi ou non.

4.3.1.1 Jetons de contexte avant l'établissement de contexte SPKM-3

Dans un jeton LIPKEY InitialContextToken, thisMech sera l'identifiant d'objet pour LIPKEY. Cependant, tant que LIPKEY n'a pas établi le mécanisme SPKM-3, le innerContextToken pour InitialContextToken et pour SubsequentContextToken sera le résultat d'un GSS_Init_sec_context ou d'un GSS_Accept_sec_context SPKM-3. De sorte que le innerContextToken LIPKEY sera soit :

* un InitialContextToken, avec thisMech réglé à l'identifiant d'objet pour SPKM-3, avec innerContextToken défini comme étant un SPKMInnerContextToken, comme défini dans la [RFC2025], soit
* un SubsequentContextToken, avec innerContextToken défini comme étant SPKMInnerContextToken.

4.3.1.2 Jetons d'établissement de contexte post-SPKM-3

Une fois que le contexte SPKM-3 est établi, il y a juste un jeton envoyé de l'initiateur à la cible, et un jeton retourné à l'initiateur.

4.3.1.2.1 De l'initiateur LIPKEY

L'initiateur LIPKEY génère un jeton qui est le résultat d'un GSS_Wrap (conf_req est réglé à VRAI) d'un nom d'utilisateur et d'un mot de passe par le contexte SPKM-3. L'argument input_message de GSS_Wrap se réfère à une instance du type UserName-Password défini ci-dessous :

```
UserName-Password ::= SEQUENCE {
      user-name      CHAINE D'OCTETS,          -- chaque octet est un octet d'une chaîne UTF-8 [RFC2279]
      password       CHAINE D'OCTETS           -- chaque octet est un octet d'une chaîne UTF-8 [RFC2279]
  }
```

4.3.1.2.2 De la cible LIPKEY

La cible valide le jeton nom d'utilisateur et mot de passe provenant de l'initiateur, et génère un jeton de réponse qui est le résultat output_message d'une invocation GSS_Wrap SPKM-3 (conf_req peut être ou non réglé à VRAI) sur une indication de réussite de validation. L'argument input_message de GSS_Wrap se réfère à une instance du type Valid-UNP défini ci-dessous :

```
Valid-UNP ::= BOOLÉEN          -- Si VRAI, la paire nom d'utilisateur/mot de passe était valide.
```

4.3.2 Jetons provenant de GSS_GetMIC et de GSS_Wrap

La [RFC2078] définit le jeton émis par GSS_GetMIC et GSS_Wrap comme :

```
PerMsgToken ::=   innerMsgToken ANY
                                  -- comme émis par GSS_GetMIC et traité par GSS_VerifyMIC.
                                  -- une structure ASN.1 n'est pas exigée.

SealedMessage ::=   sealedUserData ANY
                                  -- comme émis par GSS_Wrap et traité par GSS_Unwrap.
                                  -- inclut un indicateur interne, défini par le mécanisme du chiffrement ou non.
                                  -- une structure ASN.1 n'est pas exigée.
```

Comme on peut le voir, il n'y a pas de préfixe indépendant du mécanisme dans PerMSGToken ou SealedMessage, et pas d'informations explicites spécifiques du mécanisme. Comme LIPKEY n'a pas de valeur ajouté à GSS_GetMIC et GSS_Wrap autre que de passer le message au GSS_GetMIC et GSS_Wrap SPKM-3, les jetons PerMsgToken et SealedMessage de LIPKEY sont exactement ce que produisent les sous-programmes GSS_GetMIC et GSS_Wrap de SPKM-3.

4.4 Qualité de protection

LIPKEY, étant un passeur pour GSS_Wrap et GSS_GetMIC vers SPKM-3, n'interprète ni n'altère les QOP passés aux sous-programmes susmentionnés ou reçus de leurs compléments, GSS_Unwrap, et GSS_VerifyMIC. Donc, LIPKEY prend en charge le même ensemble de QOP que SPKM-3.

5. Considérations sur la sécurité

5.1 Gestion des mots de passe

LIPKEY envoie le mot de passe en clair chiffré sur 128 bits par cast5CBC de sorte que le risque dans cette approche est dans la façon dont la cible gère le mot de passe après l'avoir fait. L'approche devrait être sûre, pourvu que la cible purge les

mémoires tampons (primaires et secondaires, comme d'un disque) qui contenaient le mot de passe, et tout hachage du mot de passe immédiatement après que le mot de passe de l'utilisateur a été validé.

5.2 Autorités de certification

L'initiateur doit avoir une liste des autorités de certification de confiance afin de vérifier la somme de contrôle (rep-ti-integ) sur le jeton de réponse de contexte de la cible SPKM-3. Si il rencontre un certificat signé par une autorité de certificat inconnue et/ou qui n'est pas de confiance, l'initiateur NE DOIT PAS accepter le certificat en silence. Si il souhaite bien accepter le certificat, il DOIT obtenir confirmation de l'utilisateur qui gère l'application qui utilise GSS-API.

5.3 Faiblesses de HMAC-MD5 et de MD5

Bien que l'algorithme de hachage MD5 ait été trouvé avoir des faiblesses [Dobbertin], ces faiblesses n'ont pas d'impact sur la sécurité de HMAC-MD5 [Dobbertin].

5.4 Sécurité de cast5CBC

L'algorithme de chiffrement cast5CBC est relativement nouveau par rapport aux algorithmes établis comme le triple DES, et RC4. Néanmoins, le choix de cast5CBC comme algorithme obligatoire pour SPKM-3 est conseillé. L'algorithme cast5CBC est un algorithme de 128 bits sur lequel est fondé l'algorithme de 256 bits cast6CBC [RFC2612]. L'algorithme cast6CBC a été jugé par l'Institut américain des normes et technologies (NIST, *National Institute of Standards and Technology*) comme n'ayant pas de "trous de sécurité" majeurs ou mineurs connus et comme ayant une "forte marge de sécurité" [AES]. Le NIST a noté des vulnérabilités par rapport aux mises en œuvre de carte à mémoire, mais beaucoup d'autres algorithmes que le NIST a analysés partagent ces faiblesses, et dans tous les cas, LIPKEY est par définition non orienté carte à mémoire.

Références

[AES] Nechvatal, J., Barker, E., Dodson, D., Dworkin, M., Foti, J., Roback, E. (Non daté, mais avant 1999). "Status Report on the First Round of the Development of the Advanced Encryption Standard." http://csrc.nist.gov/encryption/aes/round1/r1report.htm

[CCITT] CCITT (1988). Recommandation X.208. "Spécification de la notation de syntaxe abstraite numéro un (ASN.1)".

[Dobbertin] Dobbertin, H. (1996). "The Status of Md5 After a Recent Attack," RSA Laboratories' CryptoBytes, Volume 2, Number 2. ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf

[EFF] Electronic Frontier Foundation, John Gilmore (Editor) (1998). "Cracking Des: Secrets of Encryption Research, Wiretap Politics & Chip Design," O'Reilly & Associates, ISBN 1565925203.

[FIPS] National Institute of Standards and Technology (1995). "Secure Hash Standard" (SHA-1). http://www.itl.nist.gov/fipspubs/fip180-1.htm

[IANA] Internet Assigned Numbers Authority (1999). "Network Management Parameters." http://www.isi.edu/in-notes/iana/assignments/smi-numbers

[PKCS-3] RSA Laboratories (1993). "PKCS #3: Diffie-Hellman Key-Agreement Standard, Version 1.4." ftp://ftp.rsa.com/pub/pkcs/ascii/pkcs-3.asc

[RFC1831] R. Srinivasan, "RPC : Spécification de la version 2 du protocole d'appel de procédure à distance", août 1995. *(P.S.) (Obsolète, voir* RFC5531)

[RFC1832] R. Srinivasan, "XDR : norme de représentation des données externes", août 1995. *(Obsolète, voir* RFC4506) *(D.S.)*

[RFC1964] J. Linn, "Mécanisme GSS-API de Kerberos version 5", juin 1996. (*MàJ par* RFC4121 et RFC6649)

[RFC2025] C. Adams, "Mécanisme simple de GSS-API à clé publique (SPKM)", octobre 1996. *(P.S.)*

[RFC2078] J. Linn, "Interface générique de programme d'application de service de sécurité, version 2", janvier 1997. *(Obsolète, voir la* RFC2743*.)*

[RFC2104] H. Krawczyk, M. Bellare et R. Canetti, "HMAC : Hachage de clés pour l'authentification de message", février 1997.

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

[RFC2144] C. Adams, "L'algorithme de chiffrement CAST-128", mai 1997. *(Information)*

[RFC2203] M. Eisler, A. Chiu, L. Ling, "Spécification du protocole RPCSEC_GSS", septembre 1997. *(P.S.)*

[RFC2246] T. Dierks et C. Allen, "Protocole TLS version 1.0", janvier 1999.

[RFC2279] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", janvier 1998. (*Obsolète, voir* RFC3629) *(D.S.)*

[RFC2437] B. Kaliski et J. Staddon, "PKCS n° 1 : Spécifications de la cryptographie RSA version 2.0", octobre 1998. *(Obsolète, voir* RFC3447*) (Information)*

[RFC2459] R. Housley, W. Ford, W. Polk et D. Solo, "Profil de certificat d'infrastructure de clé publique X.509 et de CRL pour l'Internet", janvier 1999. (*Obsolète, voir la* RFC3280) *(P.S.)*

[RFC2612] C. Adams, J. Gilchrist, "Algorithme de chiffrement CAST-256", juin 1999. *(Information)*

[RFC3280] R. Housley, W. Polk, W. Ford et D. Solo, "Profil de certificat d'infrastructure de clé publique X.509 et de liste de révocation de certificat (CRL) pour l'Internet", avril 2002. *(Obsolète, voir* RFC5280)

[RSA-IP] Toutes les déclarations reçues par le secrétariat de l'IETF sont mises en ligne à http://www.ietf.org/ipr.html . Prière de consulter sur cette page les informations d'IPR reçues sur cette technologie, et les autres.

[Sandberg] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B. (1985). "Design and Implementation of the Sun Network Filesystem", Proceedings of the 1985 Summer USENIX Technical Conference.

[Schneier] Schneier, B. (1996). "Applied Cryptography," John Wiley & Sons, Inc., ISBN 0-471-11709-9.

[Young] Young, E.A. (1997). Collected timing results from the SSLeay source code distribution.

Remerciements

L'auteur tient à remercier :
* Jack Kabat de sa patience pour expliquer les imbrications de SPKM, de ses excellentes suggestions, et de ses commentaires de relecture.
* Denis Pinkas, Carlisle Adams, John Linn, et Martin Rex, de leurs commentaires de relecture.
* Le présent mémorandum comporte des définitions ASN.1 pour les jetons GSS-API tirées de la RFC2078, dont l'auteur est John Linn.
  - Le présent mémorandum comporte des définitions ASN.1 et d'autre texte tiré de la définition de SPKM de la RFC2025, dont l'auteur est Carlisle Adams.
  -

Adresse de l'auteur

Adresser les commentaires sur ce mémorandum à : ietf-cat-wg@lists.Stanford.EDU

Mike Eisler
Zambeel
5565 Wilson Road
Colorado Springs, CO 80919
USA
téléphone : 1-719-599-9026
mél : mike@eisler.com

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par l'Internet Society.