Groupe de travail Réseau
Request for Comments : 2025
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle

C. Adams
Bell-Northern Research
octobre 1996

Mécanisme simple de GSS-API à clé publique (SPKM)

Statut du présent mémoire
Le présent document spécifie un protocole Internet en cours de normalisation pour la communauté de l'Internet. Il appelle à la discussion et à des suggestions pour son amélioration. Prière de se référer à l'édition actuelle 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
La présente spécification définit les protocoles, procédures, et conventions qu'emploieront les homologues qui mettent en œuvre l'interface de programme d'application de service générique de sécurité (GSS-API, *Generic Security Service Application Program Interface*) (spécifié dans les RFC1508 et 1509) quand on utilise le mécanisme de clé publique simple.

Fondements
Bien que le mécanisme GSS-API de Kerberos version 5 [RFC1964] soit maintenant bien établi dans de nombreux environnements, il est important dans certaines applications d'avoir un mécanisme GSS-API qui se fonde sur une infrastructure de clé publique, plutôt que de clé symétrique. Le mécanisme décrit dans le présent document a été proposé pour satisfaire ce besoin et fournir les caractéristiques suivantes :

1) SPKM permet l'authentification aussi bien unilatérale que mutuelle sans utiliser d'horodatages sécurisés. Cela permet à des environnements qui n'ont pas accès à une heure sécurisée d'avoir néanmoins accès à une authentification sûre.

2) SPKM utilise des identifiants d'algorithme pour spécifier divers algorithmes à utiliser par les homologues communicants. Cela permet un maximum de souplesse dans divers environnements, de futures améliorations, et d'autres algorithmes de remplacement.

3) SPKM permet l'option d'une véritable signature numérique, fondée sur un algorithme asymétrique dans les opérations gss_sign() et gss_seal() (maintenant appelées gss_getMIC() et gss_wrap() dans la [RFC2078]) plutôt qu'une somme de contrôle d'intégrité fondée sur un MAC calculé avec un algorithme symétrique (par exemple, DES). Pour certains environnements, la disponibilité de vraies signatures numériques prenant en charge la non répudiation est une nécessité.

4) Les formats et procédures de données SPKM sont conçus pour être en pratique aussi similaires que possible de ceux des mécanismes de Kerberos. Ceci est fait pour faciliter la mise en œuvre dans les environnements où Kerberos a déjà été mis en œuvre.

Pour les raisons ci-dessus, on pense que SPKM offrira souplesse et fonctionnalité, sans complexité ou redondance indue.

Gestion de clés
La gestion de clé employée dans SPKM est destinée à être aussi compatible que possible aussi bien avec [X.509] qu'avec PEM [RFC1422], car ceux-ci représentent de grandes communautés d'intérêt et montrent une relative maturité dans la normalisation.

Remerciements
Beaucoup du matériel de ce document se fonde sur le mécanisme GSS-API de Kerberos version 5 [RFC1964], et est destiné à être aussi compatible que possible avec lui. Le présent document doit aussi beaucoup à Warwick Ford et Paul Van Oorschot, de Bell-Northern Research, pour de nombreuses discussions fructueuses, à Kelvin Desplanque pour des précisions relatives à la mise en œuvre, à John Linn, de OpenVision Technologies, pour ses commentaires précieux, et à Bancroft Scott de OSS pour son assistance sur l'ASN.1.

Table des Matières

1. Généralités

Le but de l'interface de programme d'application de service générique de sécurité (GSS-API, *Generic Security Service Application Program Interface*) est déclaré dans le résumé de la [RFC1508] comme suit :
"Cette définition de l'interface de programme d'application de service générique de sécurité (GSS-API) fournit des services de sécurité aux appelants d'une façon générique, pris en charge par une gamme de mécanismes et technologies sous-jacentes et permettant donc la portabilité au niveau de la source d'applications dans différents environnements. La présente spécification définit les services et primitives de GSS-API à un niveau indépendant du mécanisme sous-jacent et de l'environnement du langage de programmation, et elle sera complétée par d'autres spécifications en rapport :
- des documents qui définissent des liens de paramètre spécifiques pour des environnements de langage particulier ;
- des documents qui définissent des formats de jeton, des protocoles, et des procédures à mettre en œuvre afin de réaliser les services GSS-API par dessus des mécanismes de sécurité particuliers."

SPKM est une instance du dernier type de document et est donc appelé un "mécanisme de GSS-API". Ce mécanisme assure l'authentification, l'établissement des clés, l'intégrité des données, et la confidentialité des données dans un environnement d'application distribuée en ligne en utilisant une infrastructure de clé publique. Parce qu'il se conforme à l'interface définie par la [RFC1508], SPKM peut être utilisé comme remplacement impromptu par toute application qui fait usage de services de sécurité à travers des appels de GSS-API (par exemple, toute application qui utilise déjà la GSS-API Kerberos pour sa sécurité). L'utilisation d'une infrastructure de clé publique permet d'employer des signatures numériques prenant en charge la non répudiation pour des échanges de messages, et présente d'autres avantages tels que l'adaptabilité à de grandes populations d'utilisateurs.

Les jetons définis dans SPKM sont destinés à être utilisés par des programmes d'application conformément au "paradigme de fonctionnement" de la GSS API (voir les détails dans la [RFC1508]):

Le paradigme de fonctionnement dans lequel fonctionne GSS-API est le suivant. Un appelant GSS-API normal est lui-même un protocole de communications (ou c'est un programme d'application qui utilise un protocole de communications) qui appelle sur la GSS-API afin de protéger ses communications par des services de sécurité d'authentification, de protection d'intégrité, et/ou de confidentialité. Un appelant GSS-API accepte des jetons qui lui sont fournis par sa mise en œuvre locale de GSS-API (c'est-à-dire, son mécanisme de GSS-API) et transfère les jetons à un homologue sur un système distant ; cet homologue passe les jetons reçus à sa mise en œuvre locale de GSS-API pour traitement.

Le présent document définit deux mécanismes distincts de GSS-API, SPKM-1 et SPKM-2, dont la principale différence est que SPKM-2 exige la présence d'horodatages sécurisés pour les besoins de la détection de répétitions durant l'établissement de contexte, alors que SPKM-1 ne l'exige pas. Cela permet une plus grande souplesse pour les applications car la disponibilité d'horodatages sécurisés ne peut pas toujours être garantie dans certains environnements.

2. Algorithmes

Un certain nombre de types d'algorithmes sont employés dans SPKM. Chaque type, avec son objet et un ensemble d'exemples spécifiques, est décrit dans cette section. Pour assurer au moins un niveau minimum d'interopérabilité parmi les diverses mises en œuvre de SPKM, un des algorithmes d'intégrité est spécifié comme OBLIGATOIRE ; tous les exemples restants (et tous les autres algorithmes) peuvent être facultativement pris en charge par une mise en œuvre SPKM (noter qu'un mécanisme conforme à GSS n'a pas besoin de prendre en charge la confidentialité). Rendre obligatoire un algorithme de confidentialité peut empêcher l'exportabilité de la mise en œuvre du mécanisme ; le présent document spécifie donc certains algorithmes comme RECOMMANDÉS (c'est-à-dire que l'interopérabilité sera améliorée si ces algorithmes sont inclus dans toutes les mises en œuvre de SPKM pour lesquelles l'exportabilité n'est pas un problème).

2.1 Algorithme d'intégrité (I-ALG)

Objet :
Cet algorithme est utilisé pour s'assurer qu'un message n'a été altéré d'aucune façon après avoir été construit par l'envoyeur légitime. Selon l'algorithme utilisé, l'application de cet algorithme peut aussi assurer l'authenticité et prendre en charge la non répudiation pour le message.

Exemples :

```
IDENTIFIANT D'OBJET md5WithRSAEncryption ::= {
     iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) pkcs-1(1) 4                -- importé de [PKCS1]
}
```

Cet algorithme (OBLIGATOIRE) assure l'intégrité et l'authenticité des données et prend en charge la non répudiation en calculant une signature RSA sur le hachage MD5 de ces données. Ceci est essentiellement équivalent à md5WithRSA {1 3 14 3 2 3}, qui est défini par l'OIW (l'atelier de travail des développeurs en environnement de systèmes ouverts).

Noter que comme ceci est le seul algorithme d'intégrité/authenticité spécifié comme obligatoire pour l'instant, pour des raisons d'interopérabilité, il est aussi stipulé que md5WithRSA est l'algorithme utilisé pour signer tous les jetons d'établissement de contexte qui sont signés plutôt que générés avec un MAC – voir les détails au paragraphe 3.1.1. Dans de futures versions de ce document, des algorithmes autres ou supplémentaires pourraient être spécifiés comme obligatoires de sorte que cette stipulation sur les jetons d'établissement de contexte pourrait être retirée.

```
IDENTIFIANT D'OBJET  DES-MAC ::= {
       iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 10
-- porte la longueur en bits du MAC comme paramètre ENTIER, restreint aux multiples de huit de 16 à 64
```

Cet algorithme (RECOMMANDÉ) assure l'intégrité en calculant un MAC DES (comme spécifié par [FIPS-113]) sur ces données.

```
IDENTIFIANT D'OBJET md5-DES-CBC ::= {
       iso(1) identified-organization(3) dod(6) internet(1) security(5) integrity(3) md5-DES-CBC(1)
}
```

Cet algorithme assure l'intégrité des données par le chiffrement, en utilisant le CBC DES, le hachage de "brouillage" MD5 de ces données (voir au paragraphe 3.2.2.1 la définition et l'objet du brouillage). Cela va normalement être plus rapide en pratique que de calculer un MAC DES sauf si les données d'entrée sont extrêmement courtes (par exemple, quelques octets). Noter que sans le brouillage, la force de ce mécanisme d'intégrité est (au plus) égale à la force de DES contre une attaque de texte en clair connu.

```
IDENTIFIANT D'OBJET sum64-DES-CBC ::= {
       iso(1) identified-organization(3) dod(6) internet(1) security(5) integrity(3) sum64-DES-CBC(2)
}
```

Cet algorithme assure l'intégrité des données par chiffrement, en utilisant CBC DES, l'enchaînement des données brouillées et la somme de tous les blocs de données d'entrée (la somme calculée en utilisant l'addition modulo 2**64 - 1). Donc, dans cet algorithme, le chiffrement est une exigence pour que l'intégrité soit assurée.

Pour des commentaires concernant la sécurité de cet algorithme d'intégrité, voir [Juen84], [Davi89].

2.2 Algorithme de confidentialité (C-ALG):

Objet : Cet algorithme symétrique est utilisé pour générer les données chiffrées pour gss_seal() / gss_wrap().

Exemple :

IDENTIFIANT D'OBJET DES-CBC ::= {
iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 7
-- porte IV (CHAINE D'OCTETS) comme paramètre ; ce paramètre (facultatif) est inutilisé dans SPKM à cause de l'utilisation du brouillage.

Cet algorithme est RECOMMANDÉ.

2.3 Algorithme d'établissement de clé (K-ALG):

Objet :
Cet algorithme est utilisé pour établir une clé symétrique à utiliser par l'initiateur et par la cible sur le contexte établi. Les clés utilisées pour C-ALG et tous les I-ALG chiffrés (par exemple, DES-MAC) sont déduites de cette clé de contexte. Comme on le verra au paragraphe 3.1, l'établissement de clés est au sein de l'échange d'authentification X.509 et donc la clé symétrique partagée résultante est authentifiée.

Exemples :

IDENTIFIANT D'OBJET RSAEncryption ::= {
iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1 -- importé de [PKCS1] et de la [RFC1423].
}

Dans cet algorithme (OBLIGATOIRE) la clé de contexte est générée par l'initiateur, chiffrée avec la clé publique RSA de la cible, et envoyée à la cible. La cible n'a pas besoin de répondre à l'initiateur pour que la clé soit établie.

IDENTIFIANT D'OBJET id-rsa-key-transport ::= {
iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 22 -- importé de [X9.44]
}

Similaire à RSAEncryption, mais les informations d'authentification de source sont aussi chiffrées avec la clé publique RSA de la cible.

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

Dans cet algorithme, la clé de contexte est générée conjointement par l'initiateur et la cible en utilisant l'algorithme Diffie-Hellman d'établissement de clé. La cible doit donc répondre à l'initiateur pour que la clé soit établie (donc ce K-ALG ne peut pas être utilisé avec l'authentification unilatérale dans SPKM-2 (voir au paragraphe 3.1)).

2.4 Fonction unidirectionnelle (O-ALG) pour l'algorithme de déduction de sous-clé

Objet :
Ayant établi une clé de contexte en utilisant le K-ALG négocié, l'initiateur et la cible doivent tous deux être capables de déduire un ensemble de sous-clés pour les divers C-ALG et I-ALG chiffrés pris en charge sur le contexte. Disons que la liste (ordonnée) des C-ALG acceptés est numérotée de façon consécutive, de sorte que le premier algorithme (celui "par défaut") est numéroté "0", le suivant est numéroté "1", et ainsi de suite. Que la numérotation de la liste (ordonnée) des I-ALG acceptés est identique. Et finalement, disons que la clé de contexte est une chaîne binaire de longueur arbitraire "M", soumise à la contrainte suivante : $L \leq M \leq U$ (où la limite inférieure "L" est la longueur binaire de la plus longue clé nécessaire pour tout C-ALG ou I-ALG chiffré accepté, et la limite supérieure "U" est la plus grande taille en bits qui tient dans les paramètres K-ALG).

Par exemple, si DES et deux clés triple DES sont les algorithmes de confidentialité négociés et si DES-MAC est l'algorithme d'intégrité chiffré négocié (noter que les signatures numériques n'utilisent pas de clé de contexte) alors la clé de contexte doit avoir au moins 112 bits. Si RSAEncryption à 512 bits est le K-ALG utilisé, le générateur peut alors générer

au hasard une clé de contexte de toute longueur jusqu'à 424 bits (la plus grande entrée RSA admissible spécifiée dans [PKCS-1]) -- la cible peut déterminer la longueur qui a été choisie en retirant les octets de bourrage durant l'opération de déchiffrage RSA. D'un autre côté, si dhKeyAgreement est le K-ALG utilisé, alors la clé de contexte est le résultat du calcul Diffie-Hellman (à l'exception de l'octet de poids fort qui est éliminé pour des raisons de sécurité) de sorte que sa longueur est celle du module Diffie-Hellman, p, moins 8 bits.

L'algorithme de déduction pour une sous-clé de k bits est spécifié comme suit :

k_bits_de_droite (OWF(clé_de_contexte || x || n || s || clé_de_contexte))

où
- "x" est le caractère ASCII "C" (0x43) si la sous-clé est pour un algorithme de confidentialité ou le caractère ASCII "I" (0x49) si la sous-clé est pour un algorithme d'intégrité chiffré ;
- "n" est le numéro de l'algorithme dans la liste acceptée appropriée pour le contexte (le caractère ASCII "0" (0x30), "1" (0x31), et ainsi de suite) ;
- "s" est l'état *(stage)* du traitement – toujours le caractère ASCII "0" (0x30), sauf si "k" est supérieur à la taille du résultat de OWF, auquel cas la OWF est calculée de façon répétitive avec augmentation des valeurs ASCII de "l'état" (chaque résultat d'OWF étant enchaîné à la fin des précédents résultats d'OWF) jusqu'à ce que "k" bits aient été générés ;
- "||" est l'opération d'enchaînement ;
- "OWF" est la fonction unidirectionnelle (OWF, *One-Way Function*) appropriée.

Exemples :
IDENTIFIANT D'OBJET MD5 ::= {
iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 5
}

Cet algorithme est OBLIGATOIRE

IDENTIFIANT D'OBJET SHA ::= {
iso(1) identified-organization(3) oiw(14) secsig(3) algorithm(2) 18
}

On reconnaît que les fonctions de hachage existantes peuvent ne pas satisfaire à toutes les propriétés requises des OWF. C'est la raison pour laquelle est permise la négociation de l'OWF O-ALG durant le processus d'établissement de contexte (voir au paragraphe 2.5) car de cette façon de futures améliorations de la conception des OWF peuvent facilement être préparées. Par exemple, dans certains environnements, une technique d'OWF préférée pourrait être un algorithme de chiffrement qui chiffre les entrées spécifiées ci-dessus en utilisant la clé_de_contexte comme clé de chiffrement.

2.5 Négociation

Durant l'établissement du contexte en SPKM, l'initiateur offre à la cible un ensemble d'algorithmes de confidentialité possibles et un ensemble d'algorithmes d'intégrité possibles (noter que le terme "algorithmes d'intégrité" inclut des algorithmes de signature numérique). Les algorithmes de confidentialité choisis par la cible deviennent ceux qui peuvent être utilisés pour C-ALG sur le contexte établi, et les algorithmes d'intégrité choisis par la cible deviennent ceux qui peuvent être utilisés pour I-ALG sur le contexte établi (la cible "choisit" les algorithmes en retournant, dans le même ordre relatif, le sous ensemble de chaque liste offerte qu'il prend en charge). Noter que tout C-ALG et I-ALG peut être utilisé pour tout message sur le contexte et que le premier algorithme de confidentialité et le premier algorithme d'intégrité dans les ensembles acceptés deviennent les algorithmes par défaut pour ce contexte.

Les algorithmes acceptés de confidentialité et d'intégrité pour un contexte spécifique définissent les valeurs valides du paramètre Qualité de protection (QOP) utilisé dans les appels gss_getMIC() et gss_wrap() – voir les détails au paragraphe 5.2. Si aucune réponse n'est attendue de la cible (authentification unilatérale dans SPKM-2) alors les algorithmes offerts par l'initiateur sont ceux qui peuvent être utilisés sur le contexte (si c'est inacceptable pour la cible, un jeton de suppression doit alors être envoyé à l'initiateur de sorte que le contexte ne soit jamais établi).

De plus, dans le premier jeton d'établissement de contexte, l'initiateur offre un ensemble de K-ALG possibles, avec la clé (ou une demi clé) correspondant au premier algorithme dans l'ensemble (son algorithme préféré). Si ce K-ALG est inacceptable pour la cible, alors celle-ci doit en choisir un parmi les autres K-ALG de l'ensemble et envoyer ce choix avec la clé (ou la demi clé) correspondant à ce choix dans sa réponse (autrement, un jeton de suppression doit être envoyé de sorte que le contexte ne soit jamais établi). Si nécessaire (c'est-à-dire, si la cible choisit un K-ALG à deux passes comme le

dhKeyAgreement) l'initiateur va envoyer sa demi clé dans une réponse à la cible.

Finalement, dans le premier jeton d'établissement de contexte, l'initiateur offre un ensemble d'O-ALG possibles (un seul O-ALG si aucune réponse n'est attendue). Le O-ALG (seul) choisi par la cible devient l'OWF d'algorithme de déduction de sous clé à utiliser sur le contexte.

Dans les futures versions de SPKM, d'autres algorithmes peuvent être spécifiés pour un ou tous les I-ALG, C-ALG, K-ALG, et O-ALG.

3. Formats de jetons

Cette section expose les caractéristiques de SPKM visibles par le protocole ; elle définit les éléments de protocole pour l'interopérabilité et est indépendante des liens de langage selon la [RFC1509].

Le mécanisme de GSS-API SPKM sera identifié par un identifiant d'objet représentant "SPKM-1" ou "SPKM-2", ayant la valeur {spkm spkm-1(1)} ou {spkm spkm-2(2)}, où spkm a la valeur {iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) spkm(1)}. SPKM-1 utilise des nombres au hasard pour la détection de répétition durant l'établissement de contexte et SPKM-2 utilise des horodatages (noter que pour les deux mécanismes, les numéros de séquence sont utilisés pour fournir la détection de répétition et de hors séquence durant le contexte, si cela a été demandé par l'application).

Les jetons transférés entre les homologues GSS-API (pour les besoins de la gestion du contexte de sécurité et de la protection par message) sont définis.

3.1 Jetons d'établissement de contexte

Trois classes de jetons sont définies dans cette section : les jetons "initiateur", émis par des appels à gss_init_sec_context() et consommés par des appels à gss_accept_sec_context() ; des jetons "cible", émis par des appels à gss_accept_sec_context() et consommés par des appels à gss_init_sec_context(); et les jetons "erreur", potentiellement émis par des appels à gss_init_sec_context() ou gss_accept_sec_context(), et potentiellement consommés pas des appels à gss_init_sec_context() ou gss_accept_sec_context().

Selon la RFC1508, Appendice B, le jeton initial d'établissement de contexte sera inclus dans une trame comme suit :

```
InitialContextToken ::= [APPLICATION 0] SÉQUENCE IMPLICITE {
      thisMech              MechType,
                              -- MechType est un IDENTIFIANT D'OBJET représentant "SPKM-1" ou "SPKM-2"
      innerContextToken     TOUT DÉFINI PAR thisMech
}                             -- contenu spécifique du mécanisme
```

Lorsque thisMech est SPKM-1 ou SPKM-2, innerContextToken est défini comme suit :

```
SPKMInnerContextToken ::= CHOIX {
  req       [0] SPKM-REQ,
  rep-ti    [1] SPKM-REP-TI,
  rep-it    [2] SPKM-REP-IT,
  error     [3] SPKM-ERROR,
  mic       [4] SPKM-MIC,
  wrap      [5] SPKM-WRAP,
  del       [6] SPKM-DEL
}
```

Le tramage GSS-API ci-dessus devra être appliqué à tous les jetons émis par le mécanisme GSS-API SPKM, y compris SPKM-REP-TI (la réponse de la cible à l'initiateur) SPKM-REP-IT (la réponse de l'initiateur à la cible) SPKM-ERROR, suppression de contexte, et les jetons par message, et non juste le jeton initial dans un échange d'établissement de contexte. Bien que non exigé par la RFC1508, ceci permet aux mises en œuvre d'effectuer une vérification d'erreur améliorée. Les valeurs d'étiquettes fournies dans SPKMInnerContextToken ("[0]" à "[6]") spécifient un identifiant de jeton pour chaque jeton ; des informations similaires sont contenues dans le champ tok-id de chaque jeton. Bien qu'elles semblent redondantes, la valeur de l'étiquette et le tok-id effectuent en fait des tâches différentes : l'étiquette assure que InitialContextToken peut être correctement décodé ; tok-id assure, entre autres choses, que les données associées aux jetons par message sont cryptographiquement reliées au type de jeton voulu. Chaque innerContextToken comporte aussi un champ

context-id ; voir à la Section 6 un exposé sur les informations de token-id et de context-id et leur utilisation dans la fonction de soutien de SPKM).

Le champ innerContextToken des jetons d'établissement de contexte pour le mécanisme GSS-API de SPKM contiendra un des messages suivants : SPKM-REQ, SPKM-REP-TI, SPKM-REP-IT, et SPKM-ERROR. De plus, tous les innerContextTokens sont codés en utilisant le BER ASN.1 (restreint, dans l'intérêt de la simplicité de l'analyse, au sous-ensemble DER défini dans [X.509], clause 8.7).

Les jetons d'établissement de contexte SPKM sont définis conformément à la section 10 de [X.509] et sont compatibles avec [9798]. SPKM-1 (nombres aléatoires) utilise le paragraphe 10.3, "Authentification bidirectionnelle", lors de l'authentification unilatérale de la cible auprès de l'initiateur et utilise le paragraphe 10.4, "Authentification à trois passages", lorsque l'authentification mutuelle est demandée par l'initiateur. SPKM-2 (horodatages) utilise le paragraphe 10.2, "Authentification unidirectionnelle", lorsqu'il effectue l'authentification unilatérale de l'initiateur auprès de la cible et utilise le paragraphe 10.3, "Authentification bidirectionnelle", lorsque l'authentification mutuelle est demandée par l'initiateur.

Ce qu'implique le paragraphe précédent est que pour l'authentification SPKM-2 unilatérale aucune négociation de K-ALG ne peut être faite (la cible accepte le K-ALG et la clé de contexte donnée par l'initiateur, ou n'accepte pas le contexte). Pour l'authentification SPKM-2 mutuelle ou SPKM-1 unilatérale, une certaine négociation est possible, mais la cible peut seulement choisir parmi les K-ALG à un passage offerts par l'initiateur (ou refuser le contexte). Autrement, l'initiateur peut demander que la cible génère et transmette la clé de contexte. Pour l'authentification SPKM-1 mutuelle, la cible peut choisir tout K-ALG à un ou deux passages offert par l'initiateur et, là encore, il peut lui être demandé de générer et transmettre la clé de contexte.

Il est envisagé que l'utilisation typique de SPKM-1 ou SPKM-2 implique une authentification mutuelle. Bien que l'authentification unilatérale soit disponible pour les deux mécanismes, son utilisation n'est pas généralement recommandée.

3.1.1 Jetons d'établissement de contexte - Initiateur (premier jeton)

Afin d'accomplir un établissement de contexte, il peut être nécessaire qu'aussi bien l'initiateur que la cible aient accès au ou aux certificats de clé publique de l'autre partie. Dans certains environnements, l'initiateur peut choisir d'acquérir tous les certificats et d'envoyer ceux qui sont pertinents à la cible dans le premier jeton. Dans d'autres environnements, l'initiateur peut demander que la cible envoie les données de certificat dans son jeton de réponse, ou chaque côté peut individuellement obtenir les données de certificat dont il a besoin. Dans tous les cas, cependant, la mise en œuvre SPKM doit avoir la capacité d'obtenir des certificats qui correspondent à un nom fourni. Le mécanisme réel à utiliser pour réaliser cela est une question de mise en œuvre locale et sort donc du domaine d'application de la présente spécification.

La syntaxe pertinente pour SPKM-REQ est la suivante (noter que les importations d'autres documents sont données à l'Appendice A) :

```
SPKM-REQ ::= SÉQUENCE {
    requestToken        REQ-TOKEN,
    certif-data [0]     CertificationData FACULTATIF,
    auth-data [1]       AuthorizationData FACULTATIF
                            - voir la [RFC1510] pour un exposé sur auth-data.
}

CertificationData ::= SÉQUENCE {
    certificationPath [0]           CertificationPath FACULTATIF,
    certificateRevocationList [1]   CertificateList FACULTATIF
}                                   -- au moins l'un des deux devra être présent.

CertificationPath ::= SÉQUENCE {
    userKeyId [0]           CHAINE D'OCTETS FACULTATIF,
                                -- identifiant de la clé publique de l'utilisateur
    userCertif [1]          Certificate FACULTATIF,
                                -- certificat contenant la clé publique de l'utilisateur
    verifKeyId [2]          CHAINE D'OCTETS FACULTATIF,
                                -- identifiant de la clé de vérification publique de l'utilisateur
    userVerifCertif [3]     Certificate FACULTATIF,
                                -- certificat contenant la clé de vérification publique de l'utilisateur
```

```
    theCACertificates [4]   SÉQUENCE DE CertificatePair FACULTATIF
}                                   -- chemin de certification de la cible à la source
```

Avoir des champs de vérification séparés permet d'utiliser des paires de clés différentes (correspondant éventuellement à des algorithmes différents) pour le chiffrement/déchiffrement et la signature/vérification. La présence de [0] ou [1] et l'absence de [2] et [3] implique que la même paire de clés est à utiliser pour le chiffrement/déchiffrement et la vérification/signature (noter que cette pratique n'est normalement pas recommandée). La présence de [2] ou [3] implique qu'une paire de clés distincte est à utiliser pour la vérification/signature et donc [0] ou [1] doit aussi être présent. La présence de [4] implique qu'au moins un de [0], [1], [2], et [3] doit aussi être présent.

```
REQ-TOKEN ::= SÉQUENCE {
     req-contents     Req-contents,
    algId             AlgorithmIdentifier,
    req-integrity     Integrity          -- le "jeton" est Req-contents
}

Integrity ::= CHAINE BINAIRE
```

-- Si le algId correspondant spécifie un algorithme de signature, "Integrity" contient le résultat de l'application de la procédure de signature spécifiée dans algId à la chaîne d'octets codée en BER qui résulte de l'application de la procédure de hachage (aussi spécifiée dans algId) aux octets de "jeton" codés en DER. Autrement, si le algId correspondant spécifie un algorithme à MAC, "Integrity" contient le résultat de l'application de la procédure de MAC spécifiée dans algId aux octets codés en DER de "jeton" (noter que pour le MAC, algId doit être un des algorithmes d'intégrité offerts par l'initiateur avec la sous-clé appropriée déduite de la clé de contexte (voir au paragraphe 2.4) utilisée comme entrée de clé)

Il est envisagé que l'utilisation normale du champ Intégrité pour chacun des REQ-TOKEN, REP-TI-TOKEN, et REP-IT-TOKEN soit une vraie signature numérique, fournissant l'authentification unilatérale ou mutuelle ainsi que la protection contre la répétition, à la demande. Cependant, il y a des situations dans lesquelles le choix du MAC sera approprié. Un exemple est le cas dans lequel l'initiateur souhaite rester anonyme (de sorte que le premier, ou le premier et le troisième jetons seront avec MAC et que le second jeton sera signé). Un autre exemple est le cas dans lequel un contexte précédemment authentifié, établi, et mis en antémémoire est rétabli à un moment ultérieur (ici tous les jetons échangés seront protégés par le MAC).

Le principal avantage du choix du MAC est qu'il réduit la redondance de traitement pour les cas dans lesquels l'authentification n'est pas requise (par exemple, anonymat) ou où l'authentification est établie par d'autres moyens (par exemple, la capacité à former le MAC correct sur un jeton "frais" dans le rétablissement du contexte).

```
Req-contents ::= SÉQUENCE {
    tok-id          ENTIER (256),               -- devra contenir 0100 (hex).
    context-id      Random-Integer,             -- voir le paragraphe 6.3.
    pvno            CHAINE BINAIRE,             -- numéro de version du protocole
    timestamp       UTCTime FACULTATIF,         -- obligatoire pou SPKM-2.
    randSrc         Random-Integer,
    targ-name       Name,
    src-name [0]    Name FACULTATIF,            --doit être fourni sauf si l'origine est "anonyme".
    req-data        Context-Data,
    validity [1]    Validity FACULTATIF,
        -- intervalle de validité pour la clé (peut être utilisé dans le calcul de la durée de vie du contexte de sécurité).
    key-estb-set    Key-Estb-Algs,              -- spécifie l'ensemble des algorithmes d'établissement de clé.
    key-estb-req    CHAINE BINAIRE FACULTATIF,
        -- paramètre d'établissement de clé correspondant au premier K-ALG dans l'ensemble (no, utilisé si l'initiateur
           n'est pas capable de, ou ne veut pas, générer et transmettre en toute sécurité le matériel de clé à la cible). La clé
           établie doit satisfaire aux contraintes de longueur de clé spécifiées au paragraphe 2.4.
    key-src-bind    CHAINE D'OCTETS FACULTATIF
        -- Utilisé pour lier le nom de la source à la clé symétrique. Ce champ doit être présent pour le cas de
           l'authentification unilatérale SPKM-2 si le K-ALG utilisé ne fournit pas un tel lien (mais est facultatif dans tous
           les autres cas). La chaîne d'octets contient le résultat de l'application de la procédure MD5 de hachage
           obligatoire (dans I-ALG OBLIGATOIRE; voir le paragraphe 2.1) comme suit : MD5(src || context_key), où
           "src" sont les octets codés en DER de src-name, "context-key" est la clé symétrique (c'est-à-dire, la version non
           protégée de ce qui est transmis dans key-estb-req), et "||" est l'opération d'enchaînement.
}

        -- Le paramètre numéro de version du protocole (pvno) est une CHAINE BINAIRE qui utilise autant de bits que
```

nécessaire pour spécifier toutes les versions de protocole SPKM prises en charge par l'initiateur (un bit par version de protocole). Le protocole spécifié par le présent document est la version 0. Le bit 0 de pvno est donc établi si cette version est prise en charge ; de même, le bit 1 est établi si la version 1 (si elle est définie à l'avenir) est prise en charge, et ainsi de suite. Noter que pour l'authentification unilatérale utilisant SPKM-2, aucun jeton de réponse n'est attendu durant l'établissement de contexte, de sorte qu'aucune négociation de protocole ne peut avoir lieu ; dans ce cas, l'initiateur doit établir exactement un bit de pvno. La version de REQ-TOKEN doit correspondre au plus fort bit établi dans pvno. Le paramètre "validity" ci-dessus est la seule façon au sein de SPKM pour que l'initiateur transmette la durée de vie désirée du contexte à la cible. Comme il ne peut être garanti que l'initiateur et la cible soient synchronisés, l'espace de temps spécifié par "validity" est à prendre comme définitif (plutôt que l'heure réelle donnée dans ce paramètre).

```
Random-Integer ::= CHAINE BINAIRE
        -- Chaque mise en œuvre SPKM est chargée de générer un nombre aléatoire "frais" pour les besoins de
           l'établissement de contexte ; c'est-à-dire, qui n'a pas (avec une probabilité élevée) été utilisé précédemment. Il
           n'y a pas d'exigence cryptographique sur ce nombre aléatoire (c'est-à-dire, il n'a pas besoin d'être imprévisible,
           il doit seulement être frais).

Context-Data ::= SÉQUENCE {
    channelId       ChannelId FACULTATIF,     -- liens des canaux.
    seq-number      ENTIER FACULTATIF,        -- numéro de séquence.
    options         Options,
    conf-alg        Conf-Algs,                -- algorithmes de confidentialité.
    intg-alg        Intg-Algs,                -- algorithme d'intégrité
    owf-alg         OWF-Algs                  -- pour la déduction de sous-clé.
}

ChannelId ::= CHAINE D'OCTETS

Options ::= CHAINE BINAIRE {
    delegation-state            (0),
    mutual-state                (1),
    replay-det-state            (2),        -- utilisé pour la détection de répétition durant le contexte.
    sequence-state              (3),        -- utilisé pour le séquençage durant le contexte.
    conf-avail                  (4),
    integ-avail                 (5),
    target-certif-data-required (6)         -- utilisé pour demander les données de certificat de la cible.
}

Conf-Algs ::= CHOIX {
    algs [0]    SÉQUENCE DE AlgorithmIdentifier,
    null [1]    NUL     -- utilisé si conf. n'est pas disponible sur le contexte pour C-ALG (voir QOP en 5.2)
}

Intg-Algs ::= SÉQUENCE DE AlgorithmIdentifier       -- pour I-ALG (voir QOP en 5.2)

OWF-Algs ::= SÉQUENCE OF AlgorithmIdentifier
        -- Contient exactement un algorithme dans REQ-TOKEN pour SPKM-2 unilatéral, et contient au moins un
           algorithme autrement. Contient toujours exactement un algorithme dans REP-TOKEN.

Key-Estb-Algs ::= SÉQUENCE DE AlgorithmIdentifier       -- pour permettre la négociation de K-ALG.
```

Une séquence d'établissement de contexte fondée sur SPKM va effectuer l'authentification unilatérale si le bit mutual-req n'est pas établi dans l'invocation de l'application à gss_init_sec_context(). SPKM-2 accomplit cela en utilisant seulement SPKM-REQ (authentifiant par là l'initiateur auprès de la cible) tandis que SPKM-1 le fait en utilisant SPKM-REQ et SPKM-REP-TI (authentifiant par là la cible auprès de l'initiateur).

Les applications qui exigent l'authentification des deux homologues (initiateur et cible) doivent demander l'authentification mutuelle, d'où résulte un "état mutuel" établi au sein des options SPKM-REQ. En réponse à une telle demande, la cible du contexte va répondre à l'initiateur avec un jeton SPKM-REP-TI. Si le mécanisme SPKM-2 a été choisi, cela achève l'échange (fondé sur l'horodatage) d'établissement de contexte d'authentification mutuelle. Si le mécanisme SPKM-1 a été choisi et si SPKM-REP-TI est envoyé, l'initiateur va alors répondre à la cible par un jeton SPKM-REP-IT, achevant l'échange (fondé sur le nombre aléatoire) d'établissement de contexte d'authentification mutuelle.

Les autres bits du champ Options de Context-Data sont expliqués dans la [RFC1508], à l'exception de target-certif-data-required *(données du certificat de cible exigées)*, que l'initiateur règle à VRAI pour demander que la cible retourne ses données de certification dans le jeton SPKM-REP-TI. Pour l'authentification unilatérale dans SPKM-2 (dans lequel aucun jeton SPKM-REP-TI n'est construit) ce bit d'option est ignoré par l'initiateur et par la cible.

3.1.2 Jetons d'établissement de contexte - Cible

```
SPKM-REP-TI ::= SÉQUENCE {
      responseToken    REP-TI-TOKEN,
      certif-data      CertificationData FACULTATIF    -- inclus si l'option target-certif-data-required était réglé à VRAI
                                                          dans SPKM-REQ
}

REP-TI-TOKEN ::= SÉQUENCE {
      rep-ti-contents  Rep-ti-contents,
      algId            AlgorithmIdentifier,
      rep-ti-integ     Integrity                       -- "token" est Rep-ti-contents.
}

Rep-ti-contents ::= SÉQUENCE {
  tok-id               ENTIER (512),                   -- devra contenir 0200 (hex).
  context-id           Random-Integer,                 -- voir au paragraphe 6.3.
  pvno [0]             CHAINE BINAIRE FACULTATIF,      -- numéro de version du protocole.
  timestamp            UTCTime FACULTATIF,             -- obligatoire pour SPKM-2.
  randTarg             Random-Integer,
  src-name [1]         Name FACULTATIF,                -- doit contenir la valeur fournie dans REQ-TOKEN.
  targ-name            Name,
  randSrc              Random-Integer,
  rep-data             Context-Data,
  validity [2]         Validity  FACULTATIF,           -- intervalle de validité pour la clé (utilisé si la cible ne peut
                                                          accepter qu'une durée de vie de contexte plus courte que
                                                          celle offerte dans REQ-TOKEN).
  key-estb-id          AlgorithmIdentifier FACULTATIF, -- utilisé si la cible change l'algorithme d'établissement de clé
                                                          (doit être un membre du key-estb-set de l' initiateur).
  key-estb-str         CHAINE BINAIRE FACULTATIF       -- contient (1) la réponse au key-estb-req de l'initiateur (si il
                                                          utilisait un K-ALG à deux passes), ou (2) le key-estb-req
                                                          correspondant au K-ALG fourni dans le key-estb-id, ou (3) le key-estb-req
                                                          correspondant au premier K-ALG fourni dans le key-estb-id de
                                                          l'initiateur, si (FACULTATIF) le key-estb-req de l'initiateur n'était pas
                                                          utilisé (le key-estb-str de la cible doit être présent dans ce cas). Les clés
                                                          établies doivent satisfaire aux contraintes de longueur de clé du 2.4.
}
```

Le paramètre Numéro de version de protocole est une CHAINE BINAIRE qui utilise autant de bits que nécessaire pour spécifier une seule version de protocole SPKM offerte par l'initiateur qui est acceptée par la cible (un bit par version de protocole) ; c'est-à-dire que la cible établit exactement un bit de pvno. Si aucune des versions offertes par l'initiateur n'est acceptée par la cible, un jeton de suppression doit être retourné afin que le contexte ne soit jamais établi. Si le pvno de l'initiateur a seulement un bit établi et si la cible se trouve accepter cette version du protocole, alors cette version est utilisée sur le contexte et le paramètre pvno de REP-TOKEN peut être omis. Finalement, si l'initiateur et la cible ont une ou plusieurs versions en commun mais si la version du REQ-TOKEN reçue n'est pas prise en charge par la cible, un REP-TOKEN doit être envoyé avec le bit version désirée établi dans pvno (et des valeurs factices utilisées pour tous les champs de jetons suivants). L'initiateur peut alors répondre par un nouveau REQ-TOKEN de la version appropriée (recommençant en fait l'établissement de contexte).

3.1.3 Jetons d'établissement de contexte - Initiateur (second jeton)

La syntaxe SPKM-REP-IT pertinente est la suivante :

```
SPKM-REP-IT ::= SÉQUENCE {
    responseToken  REP-IT-TOKEN,
    algId          AlgorithmIdentifier,
    rep-it-integ   Integrity                  -- "token" est REP-IT-TOKEN
```

```
}

REP-IT-TOKEN ::= SÉQUENCE {
    tok-id          ENTIER (768),                   -- devra contenir 0300 (hex).
    context-id      Random-Integer,
    randSrc         Random-Integer,
    randTarg        Random-Integer,
    targ-name       Name,                           -- le targ-name spécifié dans REP-TI.
    src-name        Name FACULTATIF,                -- doit contenir la valeur fournie dans REQ-TOKEN.
    key-estb-rep    CHAINE BINAIRE FACULTATIF       -- contient la réponse au key-estb-str de la cible (si elle a choisi
                                                       un K-ALG à deux passes).
    }
```

3.1.4 Jeton d'erreur

La syntaxe de SPKM-ERROR est la suivante :

```
SPKM-ERROR ::= SÉQUENCE {
        error-token   ERROR-TOKEN,
        algId         AlgorithmIdentifier,
        integrity     Integrity                   -- "token" est ERROR-TOKEN

}

ERROR-TOKRN ::=   SÉQUENCE {
        tok-id        ENTIER (1024),              -- devra contenir 0400 (hex)
        context-id    Random-Integer
        }
```

Le jeton SPKM-ERROR n'est utilisé que durant le processus d'établissement de contexte. Si un jeton SPKM-REQ ou SPKM-REP-TI est reçu par erreur, la fonction de réception (gss_init_sec_context() ou gss_accept_sec_context()) va générer un jeton SPKM-ERROR à envoyer à l'homologue (si l'homologue est encore dans le processus d'établissement de contexte) et va retourner GSS_S_CONTINUE_NEEDED. Si, d'un autre côté, aucune réponse d'établissement de contexte n'est attendue de l'homologue (c'est-à-dire, si l'homologue a achevé l'établissement de contexte) la fonction va retourner le code d'état majeur approprie (par exemple, GSS_S_BAD_SIG) avec un état mineur de GSS_SPKM_S_SG_CONTEXT_ESTB_ABORT et toutes les informations pertinentes de contexte seront supprimées. Le jeton de résultat ne sera pas un jeton SPKM-ERROR mais sera plutôt un jeton SPKM-DEL qui sera traité par le gss_process_context_token() de l'homologue.

Si gss_init_sec_context() reçoit un jeton d'erreur (valide ou invalide) il val générer à nouveau SPKM-REQ comme son jeton de résultat et retourner un code d'état majeur de GSS_S_CONTINUE_NEEDED. (Noter que si le gss_accept_sec_context() de l'homologue reçoit le jeton SPKM-REQ alors qu'il attend un jeton SPKM-REP-IT, il va ignorer SPKM-REQ et retourner un jeton de résultat de longueur zéro avec un état majeur de GSS_S_CONTINUE_NEEDED.)

De même, si gss_accept_sec_context() reçoit un jeton d'erreur (valide ou invalide) il va générer à nouveau SPKM-REP-TI comme jeton de résultat et retourner un code d'état majeur de GSS_S_CONTINUE_NEEDED.

md5WithRsa est actuellement stipulé pour la signature des jetons d'établissement de contexte. Les discordances qui impliquent un module de longueur binaire peuvent se résoudre par une utilisation judicieuse du jeton SPKM-ERROR. L'initiateur du contexte signe REQ-TOKEN en utilisant le plus fort RSA qu'il accepte (par exemple, 1024 bits). Si la cible n'est pas capable de vérifier les signatures de cette longueur, il envoie une SPKM-ERROR signée avec le plus fort RSA qu'il accepte (par exemple 512).

À l'achèvement de cet échange, les deux côtés savent quelle longueur binaire de RSA est acceptée par l'autre, car la taille de la signature est égale à la taille du module. D'autres échanges peuvent être faits (en utilisant les plus petites longueurs binaires acceptées successives) jusqu'à ce qu'un accord soit trouvé ou que l'établissement de contexte soit abandonné parce qu'un accord n'est pas possible.

3.2 Jetons par message et de suppression de contexte

Trois classes de jetons sont définies dans cette section : jetons "MIC", émis par des invocations de gss_getMIC() et consommés par des invocations de gss_verifyMIC() ; jetons "Wrap", émis par l'invocation de gss_wrap() et consommés par l'invocation de gss_unwrap() ; et les jetons de suppression de contexte, émis par l'invocation de gss_init_sec_context(), gss_accept_sec_context() ou gss_delete_sec_context() et consommés par l'invocation de gss_process_context_token().

3.2.1 Jetons par message - Sign / MIC

L'utilisation de l'invocation gss_sign() / gss_getMIC() donne un jeton distinct des données d'utilisateur qui sont protégées, qui peut être utilisé pour vérifier l'intégrité de ces données lorsque elles sont reçues. Le jeton et les données peuvent être envoyés séparément par l'application d'envoi et il est de la responsabilité de l'application receveuse d'associer les données reçues au jeton reçu.

Le format du jeton SPKM-MIC est le suivant :

```
SPKM-MIC ::= SÉQUENCE {
      mic-header      Mic-Header,
      int-cksum       CHAINE BINAIRE      -- Somme de contrôle sur l'en-tête et les données, calculé conformément à
                                             l'algorithme spécifié dans le champ int-alg.
}

Mic-Header ::= SÉQUENCE {
      tok-id          ENTIER (257),       -- devra contenir 0101 (hex).
      context-id      Random-Integer,
      int-alg [0]     AlgorithmIdentifier FACULTATIF,  -- Indicateur d'algorithme d'intégrité (doit être un des
                                                          algorithmes d'intégrité accepté pour ce contexte). Si le champ
                                                          n'est pas présent, c'est l'indicateur par défaut.
      snd-seq [1]     SeqNum FACULTATIF -- champ Numéro de séquence.
}

SeqNum ::= SÉQUENCE {
      num             ENTIER,             -- le numéro de séquence lui-même.
      dir-ind         BOOLÉEN             -- indicateur de direction.
}
```

3.2.1.1 Somme de contrôle

La procédure de calcul de la somme de contrôle (commune à tous les algorithmes -- noter que pour SPKM le terme de "somme de contrôle" inclut des signatures numériques aussi bien que des hachages et des MAC) : les sommes de contrôle sont calculées sur le champ Données, logiquement précédé par les octets de l'en-tête du jeton de texte en clair (mic-header). Le résultat lie les données à l'en-tête entier de texte en clair, de façon à minimiser la possibilité d'un collage malveillant.

Par exemple, si le int-alg spécifie l'algorithme md5WithRSA, la somme de contrôle est alors formée par le calcul d'un hachage MD5 [RFC1321] sur les données de texte en clair (précédées de l'en-tête) et en calculant ensuite une signature RSA [PKCS1] sur les 16 octets du résultat MD5. La signature est calculée en utilisant la clé privée RSA restituée de la structure d'accréditifs et le résultat (dont la longueur est impliquée par le paramètre "modulus" dans la clé privée) est mémorisé dans le champ int-cksum.

Si le int-alg spécifie un algorithme de hachage chiffré (par exemple, DES-MAC ou md5-DES-CBC) la clé à utiliser est alors la sous-clé appropriée déduite de la clé de contexte (voir au paragraphe 2.4). Là encore, le résultat (dont la longueur est impliquée par int-alg) est mémorisée dans le champ int-cksum.

3.2.1.2 Numéro de séquence

On suppose que les couches de transport sous-jacentes (quelle que soit la pile de protocole utilisée par l'application) va fournir une fiabilité de communications adéquate (c'est-à-dire, pas de pertes malveillantes, de réarrangements, etc., des paquets de données traités correctement). Donc, les numéros de séquence sont utilisés dans SPKM pour les seules raisons de sécurité, par opposition à la fiabilité (c'est-à-dire, pour éviter des pertes malveillantes, la répétition, ou le réarrangement des jetons SPKM) – il est donc recommandé que les applications demandent le séquençage et la détection des répétitions sur tous les contextes. Noter que les numéros de séquence sont utilisés de telle sorte qu'il n'y ait pas d'exigence

d'horodatages sécurisés dans les jetons de message. Le numéro de séquence initial de l'initiateur pour le contexte en cours peut être donné explicitement dans le champ Context-Data de la SPKM-REQ et le numéro de séquence initial de la cible peut être donné explicitement dans le champ Context-Data de SPKM-REP-TI ; si l'un d'eux n'est pas donné, la valeur par défaut de 00 est alors à utiliser.

Champ Numéro de séquence : il est formé à partir des quatre octets du numéro de séquence de l'envoyeur et d'un indicateur de direction booléen (FAUX – l'envoyeur est l'initiateur du contexte, VRAI – l'envoyeur est celui qui accepte le contexte). Après la construction d'un jeton gss_sign/getMIC() ou gss_seal/wrap(), le numéro de séquence de l'envoyeur est incrémenté de 1.

3.2.1.3 Traitement du numéro de séquence

Le receveur du jeton va vérifier le champ Numéro de séquence en comparant le numéro de séquence au numéro de séquence attendu et l'indicateur de direction avec l'indicateur de direction attendu. Si le numéro de séquence dans le jeton est supérieur au numéro attendu, le numéro de séquence attendu est ajusté et GSS_S_GAP_TOKEN est retourné. Si le numéro de séquence du jeton est inférieur au numéro attendu, le numéro de séquence attendu n'est alors pas ajusté et GSS_S_DUPLICATE_TOKEN, GSS_S_UNSEQ_TOKEN, ou GSS_S_OLD_TOKEN est retourné, selon ce qui est approprié. Si l'indicateur de direction est faux, le numéro de séquence attendu n'est alors pas ajusté et GSS_S_UNSEQ_TOKEN est retourné.

Comme le numéro de séquence est utilisé au titre des entrées de la somme de contrôle d'intégrité, les numéros de séquence n'ont pas besoin d'être chiffrés, et les tentatives de collage d'une somme de contrôle et d'un numéro de séquence provenant de messages différents seront détectées. L'indicateur de direction va détecter les jetons qui ont été reflétés dans une intention malveillante.

3.2.2 Jetons par message - Seal / Wrap

L'utilisation de l'invocation gss_seal() / gss_wrap() donne un jeton qui encapsule les entrées de données d'usager (facultativement chiffrées) avec les quantités associées de vérification d'intégrité. Le jeton émis par gss_seal() / gss_wrap() consiste en un en-tête d'intégrité suivi par une portion de corps qui contient soit les données de texte en clair (si conf-alg = NUL) ou les données chiffrées (en utilisant la sous-clé appropriée spécifiée au paragraphe 2.4 pour un des C-ALG acceptés pour ce contexte).

Le format du jeton SPKM-WRAP est le suivant :

```
SPKM-WRAP ::= SÉQUENCE {
      wrap-header       Wrap-Header,
      wrap-body         Wrap-Body
}

Wrap-Header ::= SÉQUENCE {
      tok-id            ENTIER (513),              -- devra contenir 0201 (hex).
      context-id        Random-Integer,
      int-alg [0]       AlgorithmIdentifier FACULTATIF,   -- indicateur d'algorithme d'intégrité (doit être un de ceux
                                                             acceptés pour ce contexte). Si le champ est absent, c'est
                                                             l'indicateur par défaut.
      conf-alg [1]      Conf-Alg FACULTATIF,       -- indicateur d'algorithme de confidentialité (doit être NUL ou un de
                                                      ceux acceptés pour ce contexte). Si le champ est absent, c'est
                                                      l'indicateur par défaut. Si c'est NUL, aucun (pas de confidentialité).
      snd-seq [2]       SeqNum FACULTATIF          -- champ Numéro de séquence.
}

Wrap-Body ::= SÉQUENCE {
      int-cksum         CHAINE BINAIRE,            -- Somme de contrôle sur en-tête et données, calculé selon l'algorithme
                                                      spécifié dans le champ int-alg.
      data              CHAINE BINAIRE             -- données chiffrées ou en clair.
}

Conf-Alg ::= CHOIX {
        algId [0]       AlgorithmIdentifier,
        null [1]        NUL
}
```

3.2.2.1 Confondeur

Comme dans la [RFC1964], un confondeur aléatoire de 8 octets est ajouté aux données pour compenser le fait qu'un IV de zéro est utilisé pour le chiffrement. Le résultat est désigné sous le nom de champ de données "confondeur".

3.2.2.2 Somme de contrôle

La procédure de calcul de la somme de contrôle est commune à tous les algorithmes. Elle est calculée sur le champ des données en clair, logiquement précédées par les octets de l'en-tête du jeton de texte en clair (en-tête enveloppe, wrap-header). Comme avec gss_sign() / gss_getMIC(), le résultat lie les données à l'en-tête entier de texte en clair, afin de minimiser la possibilité d'un collage malveillant.

Les exemples de md5WithRSA et DES-MAC sont exactement comme spécifié en 3.2.1.1.

Si int-alg spécifie md5-DES-CBC et si conf-alg spécifie n'importe quoi d'autre que DES-CBC, la somme de contrôle est alors calculée conformément à 3.2.1.1 et le résultat est mémorisé dans int-cksum. Cependant, si conf-alg spécifie DES-CBC, alors le chiffrement et la protection d'intégrité sont faits comme suit. Un hachage MD5 [RFC1321] est calculé sur les données de texte en clair (précédées de l'en-tête). Cette valeur de 16 octets est ajoutée à l'enchaînement des données du "confondeur" et 1 à 8 octets de bourrage (le bourrage est comme spécifié dans la [RFC1964] pour DES-CBC). Le résultat est alors chiffré en CBC en utilisant la sous-clé DES-CBC (voir au paragraphe 2.4) et placé dans le champ "données" de Wrap-Body. Les deux blocs finaux de texte chiffré (c'est-à-dire, le hachage MD5 chiffré) sont aussi placés dans le champ int-cksum de Wrap-Body comme somme de contrôle d'intégrité.

Si int-alg spécifie sum64-DES-CBC, alors conf-alg doit spécifier DES-CBC (c'est-à-dire que la confidentialité doit être demandée par l'application invoquante ou SPKM va retourner une erreur). Le chiffrement et la protection d'intégrité sont faits en une seule passe en utilisant la sous-clé DES-CBC comme suit. La somme (modulo 2**64 - 1) de tous les blocs de données de texte en clair (précédés de l'en-tête) est calculée. Cette valeur de 8 octets est ajoutée à l'enchaînement des données du "confondeur" et de 1 à 8 octets de bourrage (le bourrage est comme spécifié dans la [RFC1964] pour DES-CBC). Comme ci-dessus, le résultat est alors chiffré en CBC et placé dans le champ "données" de Wrap-Body. Le bloc final de texte chiffré (c'est-à-dire, la somme chiffrée) est aussi placée dans le champ int-cksum de Wrap-Body comme somme de contrôle d'intégrité.

3.2.2.3 Numéro de séquence

Les numéros de séquence sont calculés et traités pour gss_wrap() exactement comme spécifié en 3.2.1.2 et 3.2.1.3.

3.2.2.4 Chiffrement des données

La procédure suivante est respectée sauf si (a) conf-alg est NUL (pas de chiffrement) ou si (b) conf-alg est DES-CBC et int-alg est md5-DES-CBC (chiffrement comme spécifié en 3.2.2.2), ou si (c) int-alg est sum64-DES-CBC (chiffrement comme spécifié en 3.2.2.2):

Les données de "confondeur" sont bourrées et chiffrées selon l'algorithme spécifié dans le champ conf-alg. Les données sont chiffrées en utilisant CBC avec un IV de zéro. La clé utilisée est la sous-clé appropriée déduite de la clé de contexte établie en utilisant l'algorithme de déduction de sous-clé décrit au paragraphe 2.4 (cela assure que la sous-clé utilisée pour le chiffrement et la sous-clé utilisée pour un algorithme d'intégrité chiffré séparé -- par exemple DES-MAC, mais pas sum64-DES-CBC -- sont différentes).

3.2.3 Jeton de suppression de contexte

Le jeton émis par gss_delete_sec_context() se fonde sur le format des jetons émis par gss_sign() / gss_getMIC().
Le format du jeton SPKM-DEL est le suivant :

```
SPKM-DEL ::= SÉQUENCE {
    del-header      Del-Header,
    int-cksum       CHAINE BINAIRE      -- Somme de contrôle d'en-tête, calculée selon l'algorithme spécifié dans le
                                           champ int-alg.
}
```

```
Del-Header ::= SÉQUENCE {
    tok-id          ENTIER (769),          -- devra contenir 0301 (hex)
    context-id      Random-Integer,
    int-alg [0]     AlgorithmIdentifier FACULTATIF,   -- Indicateur d'algorithme d'intégrité (doit être un de ceux
                                                      acceptés pour ce contexte). Si le champ est absent, l'indicateur
                                                      par défaut
    snd-seq [1]     SeqNum FACULTATIF                 -- champ Numéro de séquence.
}
```

Le champ snd-seq sera calculé comme pour les jetons émis par gss_sign() / gss_getMIC(). Le champ int-cksum sera calculé comme pour les jetons émis par gss_sign() / gss_getMIC(), sauf que le composant Données d'utilisateur des données de somme de contrôle sera une chaîne de longueur zéro.

Si un jeton de suppression valide est reçu, la mise en œuvre de SPKM va supprimer le contexte et gss_process_context_token() va retourner un état majeur de GSS_S_COMPLETE et un état mineur de GSS_SPKM_S_SG_CONTEXT_DELETED. Si par ailleurs le jeton de suppression est invalide, le contexte ne sera pas supprimé et gss_process_context_token() va retourner l'état majeur approprié (GSS_S_BAD_SIG, par exemple) et un état mineur de GSS_SPKM_S_SG_BAD_DELETE_TOKEN_RECD. L'application peut souhaiter à ce point prendre des mesures pour vérifier l'état du contexte (comme d'envoyer un message d'essai scellé/enveloppé à son homologue et d'attendre une réponse scellée/enveloppée).

4. Types de noms et identifiants d'objets

Aucune forme de nom obligatoire n'a encore été définie pour SPKM. Cette section fera l'objet d'études complémentaires.

4.1 Formes de nom facultatives

Ce paragraphe discute des formes de noms qui peuvent facultativement être prises en charge par les mises en œuvre du mécanisme GSS-API de SPKM. On reconnaît qu'il existe probablement des fonctions spécifiques des systèmes d'exploitation en dehors de GSS-API qui effectuent les traductions entre ces formes, et que les mises en œuvre de GSS-API qui prennent en charge ces formes peuvent elles-mêmes être mises en couches par dessus des fonctions spécifiques de système d'exploitation. L'inclusion de cette prise en charge au sein des mises en œuvre de GSS-API est à la décision des applications.

4.1.1 Forme de nom d'utilisateur

Cette forme de nom devra être représentée par l'identifiant d'objet {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) user_name(1)}. Le nom symbolique recommandé pour ce type est "GSS_SPKM_NT_USER_NAME".

Ce type de nom est utilisé pour indiquer un usager désigné sur un système local. Son interprétation est spécifique du système d'exploitation. Cette forme de nom est construite comme :

username

4.1.2 Forme d'UID de machine

Cette forme de nom devra être représentée par l'identifiant d'objet {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) machine_uid_name(2)}. Le nom symbolique recommandé pour ce type est "GSS_SPKM_NT_MACHINE_UID_NAME".

Ce type de nom est utilisé pour indiquer un identifiant d'usager numérique correspondant à un usager sur un système local. Son interprétation est spécifique du système d'exploitation. Le gss_buffer_desc représentant un nom de ce type devrait contenir un uid_t à signification locale, représenté dans l'ordre des octets de l'hôte. L'opération gss_import_name() résout cet uid en un nom d'utilisateur *(username)*, qui est ensuite traité comme la forme de nom d'utilisateur.

4.1.3 Forme d'UID de chaîne

Cette forme de nom devra être représentée par l'identifiant d'objet {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) string_uid_name(3)}. Le nom symbolique recommandé pour ce type est

"GSS_SPKM_NT_STRING_UID_NAME".

Ce type de nom est utilisé pour indiquer une chaîne de chiffres représentant l'identifiant numérique d'utilisateur d'un usager sur un système local. Son interprétation est spécifique du système d'exploitation. Ce type de nom est similaire à la forme d'UID de machine, sauf que la mémoire tampon contient une chaîne qui représente le uid_t.

5. Définition des paramètres

Cette section définit les valeurs des paramètres utilisés par le mécanisme de GSS-API de SPKM. Elle définit les éléments d'interface pour la prise en charge de la portabilité.

5.1 Codes d'états mineurs

Ce paragraphe recommande les noms symboliques courants pour les valeurs d'état mineur à retourner par le mécanisme de GSS-API de SPKM. L'utilisation de ces définitions permettra à des mises en œuvre indépendantes d'améliorer la portabilité d'application entre différentes mises en œuvre du mécanisme défini dans cette spécification. (En tous cas, les mises en œuvre de gss_display_status() permettront aux appelants de convertir les indicateurs de minor_status en représentations de texte.) Chaque mise en œuvre doit rendre disponible, par des fichiers inclus ou d'autres moyens, une facilité de traduction de ces noms symboliques en la valeur concrète qu'utilise une mise en œuvre particulière de GSS-API pour représenter les valeurs de d'état mineur spécifiées dans ce paragraphe. On reconnaît que cette liste peut grossir au fil du temps et que le besoin de codes d'état mineur supplémentaires spécifiques de mises en œuvre particulières peut apparaître.

5.1.1 Codes non spécifiques de SPKM (code d'état mineur *(MSB, Minor Status Code)*, bit 31, établi)

5.1.1.1 Codes qui se rapportent à GSS (bit 30 de code d'état mineur établi)

```
GSS_S_G_VALIDATE_FAILED     /* "erreur de validation" */
GSS_S_G_BUFFER_ALLOC        /* "les données de gss_buffer_t n'ont pas pu être allouées" */
GSS_S_G_BAD_MSG_CTX         /* "contexte de message invalide" */
GSS_S_G_WRONG_SIZE          /* "mauvaise taille de mémoire tampon" */
GSS_S_G_BAD_USAGE           /* "le type d'usage d'accréditif est inconnu" */
GSS_S_G_UNAVAIL_QOP         /* "la qualité de protection spécifiée est indisponible" */
```

5.1.1.2 Codes qui se rapportent à la mise en œuvre (bit 30 de code d'état mineur à zéro)

```
GSS_S_G_MEMORY_ALLOC        /* "l'allocation de mémoire demandée n'a pu être effectuée" */
```

5.1.2 Codes spécifiques de SPKM (code d'état mineur, bit 31, à zéro)

```
GSS_SPKM_S_SG_CONTEXT_ESTABLISHED   /* "le contexte est déjà pleinement établi" */
GSS_SPKM_S_SG_BAD_INT_ALG_TYPE      /* "le jeton a un type d'algorithme d'intégrité inconnu" */
GSS_SPKM_S_SG_BAD_CONF_ALG_TYPE     /* "le jeton a un type d'algorithme de confidentialité inconnu" */
GSS_SPKM_S_SG_BAD_KEY_ESTB_ALG_TYPE/* "le jeton a un type d'algorithme d'établissement de clé inconnu" */
GSS_SPKM_S_SG_CTX_INCOMPLETE        /* "tentative d'utilisation d'un contexte de sécurité incomplet" */
GSS_SPKM_S_SG_BAD_INT_ALG_SET       /* "pas d'algorithme d'intégrité commun dans l'ensemble offert" */
GSS_SPKM_S_SG_BAD_CONF_ALG_SET  /* "pas d'algorithme de confidentialité commun dans l'ensemble offert" */
GSS_SPKM_S_SG_BAD_KEY_ESTB_ALG_SET  /* "pas d'algorithme d'établissement de clé commun dans l'ensemble
                                       offert" */
GSS_SPKM_S_SG_NO_PVNO_IN_COMMON     /* "pas de numéro de version de protocole commun dans l'ensemble
                                       offert" */
GSS_SPKM_S_SG_INVALID_TOKEN_DATA    /* "format de données impropre : aucun jeton ne peut être codé" */
GSS_SPKM_S_SG_INVALID_TOKEN_FORMAT  /* "le format du jeton reçu est impropre : décodage impossible" */
GSS_SPKM_S_SG_CONTEXT_DELETED       /* "Contexte supprimé à la demande de l'homologue" */
GSS_SPKM_S_SG_BAD_DELETE_TOKEN_RECD /* "jeton de suppression reçu invalide -- contexte non supprimé" */
GSS_SPKM_S_SG_CONTEXT_ESTB_ABORT    /* "Erreur irrécupérable d'établissement de contexte. Contexte
                                       supprimé" */
```

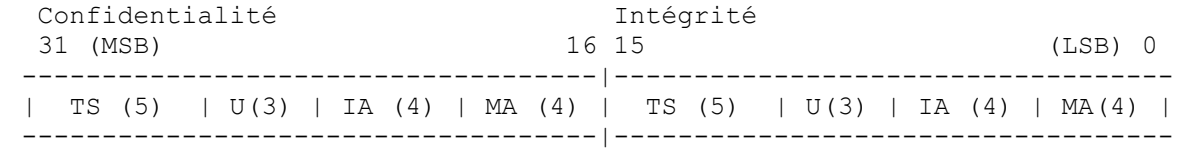
5.2 Valeurs de qualité de protection

Le paramètre Qualité de protection (QOP) est utilisé dans le mécanisme de GSS-API de SPKM comme entrée à gss_sign() et gss_seal() (gss_getMIC() et gss_wrap()) pour choisir entre les divers algorithmes de protection de la confidentialité et de vérification d'intégrité. Une fois que ces ensembles d'algorithmes ont fait l'objet d'un accord entre l'initiateur du contexte et la cible, le paramètre QOP choisit simplement parmi ces ensembles ordonnés.

Plus précisément, le jeton SPKM-REQ envoie une séquence ordonnée d'identifiants d'algorithme qui spécifient des algorithmes de vérification d'intégrité acceptés par l'initiateur et une séquence ordonnée d'identifiants d'algorithme qui spécifient des algorithmes de confidentialité pris en charge par l'initiateur. La cible retourne le sous-ensemble des identifiants d'algorithmes de vérification d'intégrité offerts qu'elle accepte et le sous-ensemble des identifiants d'algorithmes de confidentialité offerts qu'elle prend en charge dans le jeton SPKM-REP-TI (dans le même ordre relatif que celui donné par l'initiateur). Donc, l'initiateur et la cible savent tous deux quels algorithmes eux-mêmes prennent en charge et ceux qui sont acceptés par tous les deux (ces derniers sont définis comme étant ceux qui sont acceptés sur le contexte établi). Le paramètre de qualité de protection a une signification et une validité par rapport à cette connaissance. Par exemple, une application peut demander l'algorithme d'intégrité numéro 3 comme défini par la spécification du mécanisme. Si cet algorithme est pris en charge sur ce contexte, il est alors utilisé ; autrement, GSS_S_FAILURE et un code d'état mineur approprié sont retournés.

Si le jeton SPKM-REP-TI n'est pas utilisé (authentification unilatérale utilisant SPKM-2) alors les ensembles "acceptés" d'identifiants d'algorithmes sont simplement pris comme étant les ensembles de l'initiateur (si c'est inacceptable pour la cible, elle doit alors retourner un jeton d'erreur afin que le contexte ne soit pas établi). Noter que, dans l'intérêt de l'interopérabilité, l'initiateur n'est pas obligé d'offrir tous les algorithmes qu'il prend en charge ; il peut plutôt n'offrir que les algorithmes rendus obligatoires/recommandés par SPKM car ceux-ci seront probablement acceptés par la cible.

Le paramètre QOP pour SPKM est défini comme étant un entier non signé de 32 bits (un OM_uint32) avec les allocations de champs binaires suivantes :



où
TS est un identifiant de type de 5 bits (un qualificatif sémantique dont la valeur spécifie le type d'algorithme qui peut être utilisé pour protéger le jeton correspondant – voir les détails ci-dessous) ;

U est un champ de 3 bits non spécifié (disponible pour une utilisation/expansion future) ;

IA est un champ de 4 bits qui énumère les algorithmes spécifiques de la mise en œuvre ;

MA est un champ de 4 bits qui énumère les algorithmes définis par le mécanisme.

L'interprétation du paramètre QOP est la suivante (noter que la même procédure est utilisée pour les deux moitiés de confidentialité et d'intégrité du paramètre). Le champ MA est examiné en premier. Si il est différent de zéro, l'algorithme utilisé pour protéger le jeton est alors l'algorithme spécifié par le mécanisme qui correspond à cette valeur d'entier.

Si MA est zéro, alors IA est examiné. Si cette valeur de champ est différente de zéro, alors l'algorithme utilisé pour protéger le jeton est l'algorithme spécifié par la mise en œuvre qui correspond à cette valeur d'entier (si cet algorithme est disponible sur le contexte établi). Noter que l'utilisation de ce champ peut gêner la portabilité car une valeur particulière peut spécifier un algorithme dans une mise en œuvre du mécanisme et peut n'être pas prise en charge ou peut spécifier un algorithme complètement différent dans une autre mise en œuvre du mécanisme.

Finalement, si les deux champs MA et IA sont à zéro, TS est alors examiné. Une valeur de zéro pour TS spécifie l'algorithme par défaut pour le contexte établi, qui est défini comme étant le premier algorithme sur la liste des algorithmes offerts de l'initiateur (de confidentialité ou d'intégrité, selon la moitié de QOP qui est examinée) qui sont pris en charge sur le contexte. Une valeur différente de zéro pour TS correspond à un qualificatif d'algorithme particulier et choisit le premier algorithme pris en charge sur le contexte qui satisfait ce qualificatif.

Les valeurs de TS suivantes (c'est-à-dire, les qualificatifs d'algorithme) sont spécifiées ; d'autres valeurs pourront être ajoutées à l'avenir.

Pour le champ TS Confidentialité :

00001 (1) = SPKM_SYM_ALG_STRENGTH_STRONG
00010 (2) = SPKM_SYM_ALG_STRENGTH_MEDIUM
00011 (3) = SPKM_SYM_ALG_STRENGTH_WEAK

Pour le champ TS Intégrité :

00001 (1) = SPKM_INT_ALG_NON_REP_SUPPORT
00010 (2) = SPKM_INT_ALG_REPUDIABLE

Il est clair que des qualificatifs tels que fort, moyen et faible sont discutables et changeront probablement avec le temps, mais pour les besoins de cette version de la spécification, on définit ces termes comme suit. Un algorithme de confidentialité est "faible" si la longueur effective de clé du chiffrement est de 40 bits ou moins ; il est de "force moyenne" si la longueur effective de clé est strictement entre 40 et 80 bits; et il est "fort" si la longueur effective de clé est de 80 bits ou plus. (Noter que "longueur effective de clé" décrit l'effort de calcul requis pour casser un chiffrement en utilisant l'attaque de cryptanalyse la mieux connue contre le chiffrement.)

Un champ TS de cinq bits permet jusqu'à 31 qualificatifs pour chaque confidentialité et intégrité (car "0" est réservé pour "par défaut"). Le présent document en spécifie trois pour la confidentialité et deux pour l'intégrité, laissant pas mal de place pour les spécifications futures. Des suggestions de qualificatifs tels que "rapide", "vitesse moyenne", et "lent" ont été faites, mais ces termes sont difficiles à quantifier (et en tous cas, dépendent de la plateforme et du processeur) et ont donc été écartés de cette spécification initiale. L'intention est que les termes de TS soient autant qu'il est possible des qualificatifs quantitatifs, indépendants de l'environnement, des algorithmes.

L'utilisation de la structure de qualité de protection telle que définie ci-dessus est finalement destinée à être comme suit.

- Les valeurs de TS sont spécifiées au niveau de GSS-API et sont donc portables à travers les mécanismes. Les applications qui ne savent rien des algorithmes sont quand même capables de choisir la "qualité" de protection pour leurs jetons de message.
- Les valeurs de MA sont spécifiées au niveau du mécanisme et sont donc portables à travers les mises en œuvre d'un mécanisme. Par exemple, toutes les mises en œuvre du mécanisme GSS-API de Kerberos v5 GSS doivent prendre en charge :
  GSS_KRB5_INTEG_C_QOP_MD5 (valeur : 1)
  GSS_KRB5_INTEG_C_QOP_DES_MD5 (valeur : 2)
  GSS_KRB5_INTEG_C_QOP_DES_MAC (valeur : 3).
  (Noter que ces valeurs de QOP d'intégrité spécifiées pour Kerberos ne sont pas en conflit avec la structure de QOP définie ci-dessus.)
- Les valeurs de IA sont spécifiées au niveau de la mise en œuvre (dans la documentation d'utilisateur, par exemple) et sont donc normalement non portables. Une application qui connaît son propre mécanisme de mise en œuvre et le mécanisme mis en œuvre de son homologue est cependant libre d'utiliser ces valeurs car elles seront parfaitement valides et significatives sur ce contexte et entre ces homologues.

Le receveur d'un jeton doit repasser à son application appelante un paramètre de QOP avec tous les champs pertinents établis. Par exemple, si le triple DES a été spécifié par un mécanisme comme algorithme 8, un receveur d'un jeton protégé par triple DES doit alors passer à son application (QOP Confidentialité TS=1, IA=0, MA=8). De cette façon, l'application est libre de lire toute partie du QOP qu'il comprend (TS ou IA/MA).

Pour aider à la mise en œuvre et à l'interopérabilité, on fait la stipulation suivante. L'ensemble des identifiants d'algorithme d'intégrité envoyé par l'initiateur doit en contenir au moins un qui spécifie un algorithme qui calcule une signature numérique prenant en charge la non répudiation, et doit en contenir au moins un qui spécifie un autre (répudiable) algorithme d'intégrité. Le sous ensemble des identifiants d'algorithme d'intégrité retourné par la cible doit aussi en contenir au moins un qui spécifie un algorithme qui calcule une signature numérique prenant en charge la non répudiation, et au moins un qui spécifie un algorithme d'intégrité répudiable.

La raison de cette stipulation est qu'elle assure que toute mise en œuvre de SPKM va fournir un service d'intégrité qui accepte la non répudiation et un qui ne prend pas en charge la non répudiation. Une application sans connaissance des algorithmes sous-jacents peut en choisir l'un ou l'autre en passant (QOP Intégrité TS=1, IA=MA=0) ou (QOP Intégrité TS=2, IA=MA=0). Bien qu'un initiateur qui souhaite rester anonyme n'utilisera en fait jamais la signature numérique non répudiable, ce service d'intégrité doit être disponible sur le contexte afin que la cible puisse l'utiliser si elle le désire.
Finalement, conformément aux algorithmes OBLIGATOIRES et RECOMMANDÉS donnés à la Section 2, les valeurs de QOP suivantes sont spécifiées pour SPKM.

Pour les champs MA de confidentialité : 0001 (1) = DES-CBC

Pour les champs MA d'intégrité : 0001 (1) = md5WithRSA
0010 (2) = DES-MAC

6. Fonctions de soutien

Cette section décrit une fonction obligatoire pour les mises en œuvre conformes à SPKM qui peut, en fait, être précieuse dans tout mécanisme de GSS-API. Elle utilise les informations d'identifiant de jeton et d'identifiant de contexte qui sont incluses dans les jetons SPKM d'établissement de contexte, d'erreur, de suppression de contexte, et par message. La fonction est définie dans les paragraphes suivants.

6.1 Appel SPKM_Parse_token

Entrées :
- o input_token CHAINE D'OCTETS

Résultats :
- o major_status ENTIER,
- o minor_status ENTIER,
- o mech_type IDENTIFIANT D'OBJET,
- o token_type ENTIER,
- o context_handle LIEN DE CONTEXTE,

Codes d'état majeur de retour :
- o GSS_S_COMPLETE indique que le jeton d'entrée a pu être analysé pour tous les champs pertinents. Les valeurs résultantes sont mémorisées respectivement dans mech_type, token_type et context_handle (avec des NUL dans tout paramètre non pertinent).
- o GSS_S_DEFECTIVE_TOKEN indique que les informations soit d'identifiant de jeton, soit d'identifiant de contexte (si on en attendait un) n'ont pas pu être analysées. Une valeur de retour non NUL dans le type de jeton indique que la dernière situation s'est produite.
- o GSS_S_NO_TYPE indique que les informations d'identifiant de jeton ont pu être analysées, mais qu'elles ne correspondent à aucun type de jeton valide.
  (Noter que ce code d'état majeur n'a pas été défini pour GSS dans la [RFC1508]. Jusqu'à ce qu'une telle définition soit donnée (si cela arrive jamais) les mises en œuvre de SPKM devraient retourner à la place GSS_S_DEFECTIVE_TOKEN avec token_type et context_handle réglé à NUL. Cela implique essentiellement que les informations d'identifiant de jeton non reconnues sont considérées comme équivalentes aux informations d'identifiant de jeton qui n'ont pas pu être analysées.)
- o GSS_S_NO_CONTEXT indique que l'identifiant de contexte a pu être analysé, mais qu'il ne correspond à aucun lien de contexte valide.
- o GSS_S_FAILURE indique que le type de mécanisme n'a pas pu être analysé (par exemple, le jeton peut être corrompu).

SPKM_Parse_token() est utilisé pour retourner à une application le type de mécanisme, le type de jeton, et le lien de contexte qui correspondent à un certain jeton d'entrée. Comme les jetons GSS-API sont destinés à être opaques à l'application appelante, cette fonction permet à l'application de déterminer les informations sur le jeton sans avoir à violer l'intention d'opacité de GSS. Le type de jeton est d'une importance primordiale, car l'application peut alors l'utiliser pour décider quelle fonction GSS invoquer afin de faire traiter le jeton.

Si tous les jetons ont le tramage suggéré dans l'Appendice B de la [RFC1508] (spécifié dans le mécanisme GSS de Kerberos v5 de la [RFC1964] et dans le présent document) toute mise en œuvre du mécanisme devrait être capable de retourner au moins le paramètre mech_type (les autres paramètres étant NUL) pour tout jeton d'entrée non corrompu. Si la mise en œuvre du mécanisme dont la fonction SPKM_Parse_token() est invoquée ne reconnaît pas le jeton, elle peut retourner token_type afin que l'application puisse invoquer ultérieurement la fonction GSS correcte. Finalement, si le mécanisme fournit un champ d'identifiant de contexte dans ses jetons (comme le fait SPKM) une mise en œuvre peut alors transposer l'identifiant de contexte en un lien de contexte et le retourner à l'application. Cela est nécessaire dans la situation où une application a plusieurs contextes ouverts simultanément, qui utilisent tous le même mécanisme. Lorsque un jeton entrant arrive, l'application peut utiliser cette fonction pour déterminer non seulement quelle fonction GSS invoquer mais aussi quel lien de contexte utiliser pour l'invocation.

Noter que cette fonction ne fait pas de traitement cryptographique pour déterminer la validité des jetons ; elle tente simplement d'analyser les champs type de mécanisme, type de jeton et identifiant de contexte de tout jeton qu'on lui donne. Donc, il est concevable, par exemple, qu'une certaine mémoire tampon de données puisse commencer par des valeurs

aléatoires qui ressemblent à un type de mécanisme valide et que SPKM_Parse_token() retourne des informations incorrectes si elles sont données par cette mémoire tampon. Bien que concevable, une telle situation est cependant improbable.

La fonction SPKM_Parse_token() est obligatoire pour les mises en œuvre conformes à SPKM, mais est facultative pour les applications. C'est-à-dire que si une application a seulement un contexte ouvert et peut deviner quelle fonction GSS invoquer (ou si elle veut mettre des codes d'erreur) elle n'a alors jamais besoin d'invoquer SPKM_Parse_token(). De plus, si cette fonction migre au niveau GSS-API, SPKM_Parse_token() sera alors déconseillé en faveur de GSS_Parse_token(), ou quel que soit le nouveau nom et la spécification de fonction. Noter finalement qu'aucun code de retour d'état mineur n'a été défini pour cette fonction pour l'instant.

6.2 Paramètre de sortie token_type

Les types de jeton suivants sont définis :

```
GSS_INIT_TOKEN          = 1
GSS_ACCEPT_TOKEN        = 2
GSS_ERROR_TOKEN         = 3
GSS_SIGN_TOKEN          = GSS_GETMIC_TOKEN      = 4
GSS_SEAL_TOKEN          = GSS_WRAP_TOKEN        = 5
GSS_DELETE_TOKEN        = 6
```

Tous les mécanismes SPKM devront être capables d'effectuer la transposition des informations d'identifiant de jeton qui sont incluses dans chaque jeton (à travers les valeurs d'étiquette dans SPKMInnerContextToken ou par le champ identifiant de jeton) en un des types de jeton ci-dessus. Les applications devraient être capables de décider, sur la base du type de jeton, quelle fonction GSS invoquer (par exemple, si le jeton est un GSS_INIT_TOKEN, l'application va alors invoquer gss_accept_sec_context(), et si le jeton est un GSS_WRAP_TOKEN alors l'application va invoquer gss_unwrap()).

6.3 Paramètre de sortie context_handle

La mise en œuvre de mécanisme SPKM est chargée d'entretenir une transposition entre la valeur d'identifiant de contexte qui est incluse dans chaque jeton et un lien de contexte, associant donc un jeton individuel à son contexte approprié. Il est clair que la valeur de context_handle peut être déterminée en local et peut, en fait, être associée à une mémoire contenant des données sensibles sur le système local, et ainsi, avoir l'identifiant de contexte en fait égal à un lien de contexte calculé ne va en général pas fonctionner. À l'inverse, avoir le lien de contexte réglé en fait égal à un identifiant de contexte calculé ne va en fait pas fonctionner non plus, en général, parce que le lien de contexte doit être retourné à l'application par le premier appel à gss_init_sec_context() ou gss_accept_sec_context(), alors que l'unicité de l'identifiant de contexte (sur tous les contextes aux deux extrémités) peut exiger que l'initiateur et la cible soient tous deux impliqués dans le calcul. Par conséquent, context_handle et context-id doivent être calculés séparément et la mise en œuvre du mécanisme doit être capable de transposer de l'un à l'autre au plus tard à l'achèvement de l'établissement de contexte.

Le calcul de l'identifiant de contexte durant l'établissement de contexte est accompli comme suit. Chaque mise en œuvre SPKM est responsable de générer un nombre aléatoire "frais" ; c'est-à-dire qui (avec une forte probabilité) n'a pas été utilisé précédemment. Noter qu'il n'y a pas d'exigences cryptographiques sur ce nombre aléatoire (c'est-à-dire, il n'a pas besoin d'être imprévisible, il doit simplement être frais). L'initiateur passe son nombre aléatoire à la cible dans le champ context-id du jeton SPKM-REQ. Si aucun autre jeton d'établissement de contexte n'est attendu (comme pour l'authentification unilatérale dans SPKM-2) cette valeur est alors prise comme context-id (si c'est inacceptable pour la cible, un jeton d'erreur doit alors être généré). Autrement, la cible génère son nombre aléatoire et l'enchaîne à la fin du nombre aléatoire de l'initiateur. Cette valeur enchaînée est alors prise comme identifiant de contexte et est utilisée dans SPKM-REP-TI et dans tous les jetons ultérieurs sur ce contexte.

Que les deux homologues aient contribué à l'identifiant de contexte assure à chaque homologue la fraîcheur et donc empêche les attaques en répétition entre les contextes (où un jeton provenant d'un vieux contexte entre deux homologues est malicieusement injecté dans un nouveau contexte entre les mêmes homologues ou des homologues différents). Une telle assurance n'est pas disponible pour la cible dans le cas d'authentification unilatérale utilisant SPKM-2, simplement parce que elle n'a pas contribué à la fraîcheur de l'identifiant de contexte calculé (elle doit à la place faire confiance à la fraîcheur du nombre aléatoire de l'initiateur, ou rejeter le contexte). Le champ key-src-bind dans SPKM-REQ est obligatoirement présent pour le cas de l'authentification unilatérale SPKM-2 précisément pour aider la cible à faire confiance à la fraîcheur de ce jeton (et sa clé de contexte proposée).

7. Considérations pour la sécurité

Les questions de sécurité sont discutées tout au long de ce mémoire.

8. Références

[Davi89] D. W. Davies et W. L. Price, "Security for Computer Networks", Seconde édition, John Wiley and Sons, New York, 1989.

[FIPS-113] National Bureau of Standards, Federal Information Processing Standard 113, "Computer Data Authentication", mai 1985.

[Juen84] R. R. Jueneman, C. H. Meyer et S. M. Matyas, "Message Authentication with Manipulation Detection Codes", dans le compte rendu du 1983 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1984, pp.33-54.

[PKCS1] "RSA Encryption Standard, Version 1.5", RSA Data Security, Inc., novembre 1993.

[PKCS3] "Diffie-Hellman Key-Agreement Standard, Version 1.4", RSA Data Security, Inc., novembre 1993.

[RFC1321] R. Rivest, "Algorithme de résumé de message MD5", avril 1992. *(Information)*

[RFC1422] S. Kent, "Amélioration de la confidentialité pour la messagerie électronique Internet : Partie II – Gestion de clés fondée sur le certificat", février 1993. *(Historique)*

[RFC1423] D. Balenson, D., "Amélioration de la confidentialité pour la messagerie électronique Internet : Partie III -- Algorithmes, modes et identifiants", février 1993. *(Historique)*

[RFC1508] J. Linn, "Interface générique de programme de service de sécurité", septembre 1993. *(Rendue obsolète par la* RFC2078, *elle-même rendue obsolète par la* RFC2743.*) (P.S.)*

[RFC1509] J. Wray, "API de service générique de sécurité : liens C", septembre 1993. *(remplacée par* RFC2744*)*

[RFC1510] J. Kohl et C. Neuman, "Service Kerberos d'authentification de réseau (v5)", septembre 1993. *(Obsolète, voir* RFC6649*)*

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

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

[9798] ISO/CEI 9798-3, "Technologies de l'information - Techniques de sécurité – Mécanismes d'authentification d'entité - Partie 3 : Authentification d'entité utilisant un algorithme de clé publique", 1993.

[X.501] ISO/CEI 9594-2, "Technologies de l'information - Interconnexion des systèmes ouverts – L'annuaire : Modèles", Recommandation UIT-T X.501, 1993.

[X.509] ISO/CEI 9594-8, "Technologies de l'information - Interconnexion des systèmes ouverts – L'annuaire : Cadre d'authentification", Recommandation UIT-T X.509, 1993.

[X9.44] ANSI, "Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry: Transport of Symmetric Algorithm Keys Using RSA", X9.44-1993.

9. Adresse de l'auteur

Carlisle Adams
Bell-Northern Research
P.O.Box 3511, Station C
Ottawa, Ontario, CANADA K1Y 4H7
téléphone : +1 613.763.9008
mél : cadams@bnr.ca

Appendice A Définition de module ASN.1

SpkmGssTokens {iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) spkm(1) spkmGssTokens(10)}

ÉTIQUETTES IMPLICITES DE DÉFINITIONS ::=
DÉBUT

-- EXPORTE TOUT --

IMPORTE

Nom
DE InformationFramework {joint-iso-ccitt(2) ds(5) module(1) informationFramework(1) 2}

Certificate, CertificateList, CertificatePair, AlgorithmIdentifier, Validity
DE AuthenticationFramework {joint-iso-ccitt(2) ds(5) module(1) authenticationFramework(7) 2} ;

-- types --

```
SPKM-REQ ::= SÉQUENCE {
    requestToken        REQ-TOKEN,
    certif-data [0]     CertificationData FACULTATIF,
    auth-data [1]       AuthorizationData FACULTATIF
}

CertificationData ::= SÉQUENCE {
    certificationPath [0]          CertificationPath FACULTATIF,
    certificateRevocationList [1]  CertificateList FACULTATIF
}                                              -- au moins un d'eux devra être présent

CertificationPath ::= SÉQUENCE {
    userKeyId [0]       CHAINE D'OCTETS FACULTATIF,
    userCertif [1]      Certificate FACULTATIF,
    verifKeyId [2]      CHAINE D'OCTETS FACULTATIF,
    userVerifCertif [3] Certificate FACULTATIF,
    theCACertificates [4] SÉQUENCE DE CertificatePair FACULTATIF
}       -- La présence de [2] ou [3] implique que [0] ou [1] doit aussi être présent. La présence de [4] implique qu'au
        moins un de [0], [1], [2], et [3] doit aussi être présent.

REQ-TOKEN ::= SÉQUENCE {
    req-contents        Req-contents,
    algId               AlgorithmIdentifier,
    req-integrity       Integrity           -- "token" est Req-contents.
}

Integrity ::= CHAINE BINAIRE
    -- Si l'identifiant d'algorithme correspondant spécifie un algorithme de signature, "Integrity" contient le résultat de
       l'application de la procédure de signature spécifiée dans algId à la chaîne d'octets codée en BER qui résulte de
       l'application de la procédure de hachage (aussi spécifiée dans algId) aux octets codés en DER de "token".
       Autrement, si le algId correspondant spécifie un algorithme de MAC, "Integrity" contient le résultat de l'application
       de la procédure de MAC spécifiée dans algId aux octets codés ee DER de "token".

Req-contents ::= SÉQUENCE {
    tok-id          ENTIER (256),           -- devra contenir 0100 (hex).
    context-id      Random-Integer,
    pvno            CHAINE BINAIRE,
    timestamp       UTCTime FACULTATIF,     -- obligatoire pour SPKM-2.
    randSrc         Random-Integer,
    targ-name       Name,
    src-name [0]    Name FACULTATIF,
    req-data        Context-Data,
```

```
        validity [1]        Validity FACULTATIF,
        key-estb-set        Key-Estb-Algs,
        key-estb-req        CHAINE BINAIRE FACULTATIF,
        key-src-bind        CHAINE D'OCTETS FACULTATIF
          -- Ce champ doit être présent pour le cas d'authentification unilatérale de SPKM-2 si le K-ALG utilisé ne fournit pas
             un tel lien (mais il est facultatif pour tous les autres cas). La chaîne d'octets contient le résultat de l'application de
             la procédure obligatoire de hachage (dans ALGORITHME OBLIGATOIRE voir le paragraphe 2.1) comme suit :
             MD5(src || context_key), où "src" sont les octets codés en DER de src-name, "context-key" est la clé symétrique
             (c'est-à-dire, la version non protégée de ce qui est transmis dans key-estb-req), et "||" l'opération d'enchaînement.
    }

    Random-Integer ::= CHAINE BINAIRE
    Context-Data ::= SÉQUENCE {
        channelId           ChannelId FACULTATIF,
        seq-number          ENTIER FACULTATIF,
        options             Options,
        conf-alg            Conf-Algs,
        intg-alg            Intg-Algs,
        owf-alg             OWF-Algs
    }

    ChannelId ::= CHAINE D'OCTETS
    Options ::= CHAINE BINAIRE {
        delegation-state            (0),
        mutual-state                (1),
        replay-det-state            (2),
        sequence-state              (3),
        conf-avail                  (4),
        integ-avail                 (5),
        target-certif-data-required (6)
    }

    Conf-Algs ::= CHOIX {
        algs [0]        SÉQUENCE DE AlgorithmIdentifier,
        null [1]        NUL
    }

    Intg-Algs ::= SÉQUENCE DE AlgorithmIdentifier

    OWF-Algs ::= SÉQUENCE DE AlgorithmIdentifier

    Key-Estb-Algs ::= SÉQUENCE DE AlgorithmIdentifier

    SPKM-REP-TI ::= SÉQUENCE {
        responseToken   REP-TI-TOKEN,
        certif-data     CertificationData FACULTATIF    -- présent si l'option target-certif-data-required était réglée à
                                                           VRAI dans SPKM-REQ.
    }

    REP-TI-TOKEN ::= SÉQUENCE {
        rep-ti-contents Rep-ti-contents,
        algId           AlgorithmIdentifier,
        rep-ti-integ    Integrity                       -- "token" est Rep-ti-contents.
    }

    Rep-ti-contents ::= SÉQUENCE {
        tok-id          ENTIER (512),                   -- devra contenir 0200 (hex)
        context-id      Random-Integer,
        pvno [0]        CHAINE BINAIRE FACULTATIF,
        timestamp       UTCTime FACULTATIF,             -- obligatoire pour SPKM-2
        randTarg        Random-Integer,
        src-name [1]    Name FACULTATIF,
        targ-name       Name,
```

```
        randSrc         Random-Integer,
        rep-data        Context-Data,
        validity [2]    Validity  FACULTATIF,
        key-estb-id     AlgorithmIdentifier FACULTATIF,
        key-estb-str    CHAINE BINAIRE FACULTATIF
}

SPKM-REP-IT ::= SÉQUENCE {
        responseToken   REP-IT-TOKEN,
        algId           AlgorithmIdentifier,
        rep-it-integ    Integrity                   -- "token" est REP-IT-TOKEN.
}

REP-IT-TOKEN ::= SÉQUENCE {
        tok-id          ENTIER (768),               -- devra contenir 0300 (hex).
        context-id      Random-Integer,
        randSrc         Random-Integer,
        randTarg        Random-Integer,
        targ-name       Name,
        src-name        Name FACULTATIF,
        key-estb-rep    CHAINE BINAIRE FACULTATIF
}

SPKM-ERROR ::= SÉQUENCE {
        errorToken      ERROR-TOKEN,
        algId           AlgorithmIdentifier,
        integrity       Integrity                   -- "token" est ERROR-TOKEN.
}

ERROR-TOKEN ::=  SÉQUENCE {
        tok-id          ENTIER (1024),              -- devra contenir 0400 (hex).
        context-id      Random-Integer
}

SPKM-MIC ::= SÉQUENCE {
        mic-header      Mic-Header,
        int-cksum       CHAINE BINAIRE
}

Mic-Header ::= SÉQUENCE {
        tok-id          ENTIER (257),               -- devra contenir 0101 (hex).
        context-id      Random-Integer,
        int-alg [0]     AlgorithmIdentifier FACULTATIF,
        snd-seq [1]     SeqNum FACULTATIF
}

SeqNum ::= SÉQUENCE {
        num             ENTIER,
        dir-ind         BOOLÉEN
}

SPKM-WRAP ::= SÉQUENCE {    wrap-header     Wrap-Header,
        wrap-body       Wrap-Body
}

Wrap-Header ::= SÉQUENCE {
        tok-id          ENTIER (513),               -- devra contenir 0201 (hex).
        context-id      Random-Integer,
        int-alg [0]     AlgorithmIdentifier FACULTATIF,
        conf-alg [1]    Conf-Alg FACULTATIF,
        snd-seq [2]     SeqNum FACULTATIF
}
```

```
Wrap-Body ::= SÉQUENCE {
      int-cksum         CHAINE BINAIRE,
      data              CHAINE BINAIRE
}

Conf-Alg ::= CHOICE {
      algId [0]         AlgorithmIdentifier,
      null [1]          NUL
}

SPKM-DEL ::= SÉQUENCE {
      del-header        Del-Header,
      int-cksum         CHAINE BINAIRE
}

Del-Header ::= SÉQUENCE {
      tok-id            ENTIER (769),                    -- devra contenir 0301 (hex).
      context-id        Random-Integer,
      int-alg [0]       AlgorithmIdentifier FACULTATIF,
      snd-seq [1]       SeqNum FACULTATIF
}

-- autre types --
-- d'après la [RFC1508] --
MechType ::= IDENTIFIANT D'OBJET

InitialContextToken ::= [APPLICATION 0] SÉQUENCE IMPLICITE {
  thisMech              MechType,
  innerContextToken     SPKMInnerContextToken
}                                                      -- lorsque thisMech est SPKM-1 ou SPKM-2.

SPKMInnerContextToken ::= CHOIX {
  req                   [0] SPKM-REQ,
  rep-ti                [1] SPKM-REP-TI,
  rep-it                [2] SPKM-REP-IT,
  error                 [3] SPKM-ERROR,
  mic                   [4] SPKM-MIC,
  wrap                  [5] SPKM-WRAP,
  del                   [6] SPKM-DEL
}

-- d'après la [RFC1510] --
AuthorizationData ::= SÉQUENCE DE SÉQUENCE {
  ad-type               ENTIER,
  ad-data               CHAINE D'OCTETS
}

-- allocation des identifiants d'objet --

IDENTIFIANT D'OBJET md5-DES-CBC ::=
   {iso(1) identified-organization(3) dod(6) internet(1) security(5) integrity(3) md5-DES-CBC(1)}

IDENTIFIANT D'OBJET sum64-DES-CBC ::=
     {iso(1) identified-organization(3) dod(6) internet(1) security(5) integrity(3) sum64-DES-CBC(2)}

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

IDENTIFIANT D'OBJET spkm-2 ::=
    {iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) spkm(1) spkm-2(2)}

FIN
```

Appendice B Types importés

Cet appendice contient, dans un souci de complétude, les types ASN.1 pertinents importés de InformationFramework (1993), AuthenticationFramework (1993), et [PKCS3].

```
AttributeType ::= IDENTIFIANT D'OBJET
AttributeValue ::= TOUT
AttributeValueAssertion ::= SÉQUENCE {AttributeType,AttributeValue}
RelativeDistinguishedName ::= ENSEMBLE DE AttributeValueAssertion
                -- noter que le module 1993 de InformationFramework utilise une syntaxe différente pour cette construction
RDNSequence ::= SÉQUENCE DE RelativeDistinguishedName
DistinguishedName ::= RDNSequence
Name ::= CHOIX {                                        -- un seul pour l'instant.
    rdnSequence     RDNSequence
}

Certificate ::= SÉQUENCE {
    certContents    CertContents,
    algID           AlgorithmIdentifier,
    sig             CHAINE BINAIRE
}       -- sig contient le résultat de l'application de la procédure de signature spécifiée dans algId à la chaîne d'octets
        codée en BER qui résulte de l'application de la procédure de hachage (aussi spécifiée dans algId) aux octets
        codés en DER de CertContents.

CertContents ::= SÉQUENCE {
    version [0]             Version PAR DÉFAUT v1,
    serialNumber            CertificateSerialNumber,
    signature               AlgorithmIdentifier,
    issuer                  Name,
    validity                Validity,
    subject                 Name,
    subjectPublicKeyInfo    SubjectPublicKeyInfo,
    issuerUID [1]           UID IMPLICITE FACULTATIF,       -- utilisé seulement en v2.
    subjectUID [2]          UID IMPLICITE FACULTATIF        -- utilisé seulement en v2.
}

Version ::= ENTIER {v1(0), v2(1)}
CertificateSerialNumber ::= ENTIER
UID ::= CHAINE BINAIRE

Validity ::= SÉQUENCE {
    notBefore       UTCTime,
    notAfter        UTCTime
}

SubjectPublicKeyInfo ::= SÉQUENCE {
    algorithm           AlgorithmIdentifier,
    subjectPublicKey    CHAINE BINAIRE
}

CertificatePair ::= SÉQUENCE {
    forward [0]     Certificate FACULTATIF,
    reverse [1]     Certificate FACULTATIF
}                                                       -- au moins un des deux doit être présent.

CertificateList ::= SÉQUENCE {
    certListContents CertListContents,
    algId           AlgorithmIdentifier,
    sig             CHAINE BINAIRE
}       -- sig contient le résultat de l'application de la procédure de signature spécifiée dans algId à la chaîne d'octets
        codée en BER qui résulte de l'application de la procédure de hachage (aussi spécifiée dans algId) aux octets
        codés en DER de CertListContents
```

```
CertListContents ::= SÉQUENCE {
      signature             AlgorithmIdentifier,
      issuer                Name,
      thisUpdate            UTCTime,
      nextUpdate            UTCTime FACULTATIF,
      revokedCertificates   SÉQUENCE DE SÉQUENCE {
          userCertificate   CertificateSerialNumber,
          revocationDate    UTCTime         } FACULTATIF
}

AlgorithmIdentifier ::= SÉQUENCE {
      algorithm         IDENTIFIANT D'OBJET,
      parameter         TOUT DÉFINI PAR algorithm FACULTATIF
 }        -- noter que le module 1993 d'AuthenticationFramework utilise une syntaxe différente pour cette construction que
             [PKCS3] (le paramètre à utiliser avec dhKeyAgreement) --

DHParameter ::= SÉQUENCE {
  prime                 ENTIER,                 -- p
  base                  ENTIER,                 -- g
  privateValueLength    ENTIER FACULTATIF
}
```