Groupe de travail Réseau
**Request for Comments : 4178**
RFC rendue obsolète : 2478
Catégorie : Sur la voie de la normalisation
Traduction Claude Brière de L'Isle

L. Zhu, Microsoft Corporation
P. Leach, Microsoft Corporation
K. Jaganathan, Microsoft Corporation
W. Ingersoll, Sun Microsystems
octobre 2005

# Mécanisme simple et protégé de négociation d'interface de programme d'application de service de sécurité générique (GSS-API)



**Résumé**
Le présent document spécifie un mécanisme de négociation pour l'interface de programme d'application de service général de sécurité (GSS-API, *General Security Service Application Program Interface*) qui est décrit dans la RFC 2743. Les homologues GSS-API peuvent utiliser ce mécanisme de négociation pour choisir dans un ensemble commun de mécanismes de sécurité. Si des services d'intégrité par message sont disponibles sur le contexte de mécanisme établi, la négociation est alors protégée contre un attaquant qui force la sélection d'un mécanisme non désiré par les homologues.

Ce mécanisme remplace la RFC 2478 afin de corriger les défauts de cette spécification et décrire comment interopérer avec des mises en œuvre de cette spécification qui sont couramment déployées dans l'Internet.

## Table des Matières



## 1. Introduction

GSS-API [RFC2743] fournit une interface générique qui peut être mise en couches par dessus différents mécanismes de sécurité tels que, si les homologues communicants acquièrent des accréditifs GSS-API pour le même mécanisme de

sécurité, un contexte de sécurité peut alors être établi entre eux (selon la politique). Cependant, GSS-API ne prescrit pas la méthode par laquelle les homologues GSS-API peuvent établir si ils ont un mécanisme de sécurité commun.

Le mécanisme de négociation simple et protégée de GSS-API (SPNEGO, *Simple et Protected GSS-API Negotiation*) défini ici est un pseudo mécanisme de sécurité qui permet aux homologues GSS-API de déterminer dans la bande si leurs accréditifs prennent en charge un ensemble commun d'un ou plusieurs mécanismes de sécurité GSS-API ; si c'est le cas, ils invoquent l'établissement normal de contexte de sécurité pour un mécanisme de sécurité choisi commun. C'est très utile pour les applications qui dépendent de mises en œuvre de GSS-API et partagent plusieurs mécanismes avec leurs homologues.

La négociation de mécanisme SPNEGO se fonde sur le modèle suivant : l'initiateur propose une liste de mécanismes de sécurité, en ordre de préférence décroissante (choix favori en premier) l'acceptant (aussi appelé la cible) accepte le mécanisme de sécurité préféré de l'initiateur (le premier de la liste) ou choisit un des mécanismes disponibles dans la liste offerte ; si aucun n'est acceptable, l'acceptant rejette les valeurs proposées. La cible informe alors l'initiateur de son choix.

Une fois qu'un mécanisme de sécurité commun est choisi, des options spécifiques de mécanisme PEUVENT être négociées au titre de l'établissement du contexte du mécanisme choisi. Ces négociations (si il en est) sont internes au mécanisme et opaques au protocole SPNEGO. À ce titre, elles sortent du domaine d'application du présent document.

Si des services d'intégrité par message [RFC2743] sont disponibles sur le mécanisme de contexte de sécurité établi, la négociation est alors protégée pour assurer que la liste de mécanismes n'a pas été modifiée. Dans les cas où un attaquant pourrait avoir influencé matériellement la négociation, les homologues échangent des jetons de code d'intégrité de message (MIC, *message integrity code*) pour confirmer que la liste de mécanismes n'a pas été modifiée. Si aucune action d'un attaquant n'aurait pu avoir matériellement modifié le résultat de la négociation, l'échange de jetons de MIC est facultatif (voir la Section 5). Permettre que les jetons de MIC soient facultatifs dans ce cas assure l'interopérabilité avec les mises en œuvre existantes tout en protégeant quand même la négociation. Cette interopérabilité est au prix d'une complexité accrue.

SPNEGO s'appuie sur les concepts développés dans la spécification GSS-API [RFC2743]. Les données de négociation sont encapsulées dans des jetons de niveau contexte. Donc, les appelants de la GSS-API n'ont pas besoin de connaître l'existence des jetons de la négociation, mais seulement le nouveau pseudo mécanisme de sécurité. Un échec dans la phase de négociation cause le retour d'un code d'état majeur : GSS_S_BAD_MECH.

## 2. Conventions utilisées dans le document

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

## 3. Protocole de négociation

Lorsque le contexte de mécanisme établi fournit la protection de l'intégrité, la négociation de mécanisme peut être protégée. Quand on acquiert des jetons de mécanisme de sécurité négociés, les services de sécurité par message sont toujours exigés par le mécanisme SPNEGO.

Quand le contexte de mécanisme établi prend en charge les services de sécurité par message, SPNEGO garantit que le mécanisme choisi est mutuellement préféré.

Cette section décrit le processus de négociation de ce protocole.

### 3.1 Description de la négociation

Le premier jeton de négociation envoyé par l'initiateur contient une liste ordonnée de mécanismes en ordre décroissant de préférence (le mécanisme favori en premier) et facultativement le jeton initial de mécanisme pour le mécanisme préféré de l'initiateur (c'est-à-dire, le premier de la liste). (Noter que la liste NE DOIT PAS contenir ce mécanisme SPNEGO lui-même ou tout mécanisme pour lequel le client n'a pas d'accréditifs appropriés.)

La cible traite alors le jeton provenant de l'initiateur. Il va en résulter qu'un des quatre états possibles (comme défini au paragraphe 4.2.2) sera retourné dans le message de réponse : accept-completed, accept-incomplete, reject, ou request-mic. Un état "reject" termine la négociation ; un état "accept-completed" indique que le mécanisme choisi par l'initiateur était

acceptable pour la cible, et que le jeton de mécanisme de sécurité incorporé dans le premier message de négociation était suffisant pour achever l'authentification ; un état "accept-incomplete" indique qu'un autre échange de messages est nécessaire mais que l'échange de jeton de MIC (comme décrit à la Section 5) est FACULTATIF ; un état "request-mic" (cet état ne peut être présent que dans le premier message de réponse de la cible) indique que l'échange de jeton de MIC est EXIGÉ si des services de sécurité par message sont disponibles.

Sauf si l'ordre de préférence est spécifié par l'application, la politique par laquelle la cible choisit un mécanisme est une affaire locale, spécifique de la mise en œuvre. En l'absence d'un ordre de préférence spécifié par l'application ou par une autre politique, la cible DEVRA choisir le premier mécanisme dans la liste proposée par l'initiateur pour lequel il a des accréditifs valides.

En cas de négociation réussie, le mécanisme de sécurité dans le premier message de réponse représente la valeur convenable pour la cible qui a été choisie sur la liste offerte par l'initiateur.

En cas d'échec de négociation, l'état "reject" est retourné, et la génération d'un jeton de négociation de niveau contexte est FACULTATIVE.

Une fois qu'un mécanisme a été choisi, les jetons d'établissement de contexte spécifiques du mécanisme choisi sont portés au sein des jetons de négociation.

Enfin, des jetons de MIC peuvent être échangés pour s'assurer de l'authenticité de la liste de mécanismes reçue par la cible.

Pour éviter des conflits avec l'utilisation de jetons de MIC par SPNEGO, des contextes partiellement établis NE DOIVENT PAS être utilisés pour les appels par message. Pour garantir cela, le prot_ready_state [RFC2743] DOIT être réglé à faux sur le retour de GSS_Init_sec_context() et GSS_Accept_sec_context(), même si le mécanisme sous-jacent a retourné vrai.

Noter qu'afin d'éviter un aller-retour supplémentaire, le premier jeton d'établissement de contexte du mécanisme préféré de l'initiateur DEVRAIT être incorporé dans le message de négociation initiale (comme défini au paragraphe 4.2). (Ce jeton de mécanisme est appelé le jeton de mécanisme optimiste dans ce document.) De plus, l'utilisation du jeton de mécanisme optimiste permet à l'initiateur de récupérer d'erreurs non fatales rencontrées en essayant de produire le premier jeton de mécanisme avant qu'un mécanisme puisse être choisi. Dans les cas où le mécanisme préféré de l'initiateur n'a pas de chances d'être choisi par l'acceptant à cause du coût significatif de sa génération, les mises en œuvre PEUVENT omettre le jeton de mécanisme optimiste.

### 3.2 Procédure de négociation

La forme de base de la procédure suppose que des services de sécurité par message sont disponibles sur le contexte de mécanisme établi, et elle est résumé comme suit :

a) L'initiateur GSS-API invoque GSS_Init_sec_context() normalement, mais demande que SPNEGO soit utilisé. SPNEGO peut être demandé explicitement ou accepté comme mécanisme par défaut.

b) La mise en œuvre de l'initiateur GSS-API génère un jeton de négociation contenant une liste d'un ou plusieurs mécanismes de sécurité disponibles sur la base des accréditifs utilisés pour cet établissement de contexte, et facultativement sur le jeton de mécanisme initial pour le premier mécanisme de la liste.

c) L'application de l'initiateur GSS-API envoie le jeton à l'application de la cible. L'application GSS-API de la cible passe le jeton en invoquant GSS_Accept_sec_context(). L'acceptant va faire une des choses suivantes :
   I) Si aucun des mécanismes proposés n'est acceptable, la négociation DEVRA être terminée. GSS_Accept_sec_context indique GSS_S_BAD_MECH. L'acceptant PEUT produire un jeton de négociation contenant un état "reject".
   II) Si le mécanisme préféré de l'initiateur n'est pas accepté par la cible ou si ce mécanisme est accepté mais n'est pas le préféré de l'acceptant (c'est-à-dire, si l'échange de jeton de MIC comme décrit à la Section 5 est exigé) GSS_Accept_sec_context() indique GSS_S_CONTINUE_NEEDED. L'acceptant DOIT produire un jeton de négociation contenant un état "request-mic".
   III) Autrement, si au moins un jeton de négociation supplémentaire provenant de l'initiateur est nécessaire pour établir ce contexte, GSS_Accept_sec_context() indique GSS_S_CONTINUE_NEEDED et produit un jeton de négociation contenant un état "accept-incomplete".
   IV) Autrement, aucun jeton de négociation supplémentaire provenant de l'initiateur n'est nécessaire pour établir ce contexte, GSS_Accept_sec_context() indique GSS_S_COMPLETE et produit un jeton de négociation contenant un état "accept_complete".

   Si le mécanisme préféré de l'initiateur est accepté, et si un jeton de mécanisme optimiste était inclus, ce jeton de

mécanisme DOIT être passé au mécanisme choisi en invoquant GSS_Accept_sec_context(). Si un jeton de mécanisme est retourné en réponse, il DOIT être inclus dans le jeton de négociation de réponse. Autrement, la cible ne va pas générer de jeton de mécanisme de réponse dans la première réponse.

d) L'application GSS-API cible retourne le jeton de négociation à l'application de l'initiateur. L'application GSS-API de l'initiateur passe le jeton en invoquant GSS_Init_sec_context(). L'initialisation du contexte de sécurité se continue alors conformément aux conventions GSS-API standard pour le mécanisme choisi, où les jetons du mécanisme choisi sont encapsulés dans un message de négociation (voir la Section 4) jusqu'à ce que GSS_S_COMPLETE soit retourné pour l'initiateur et la cible par le mécanisme de sécurité choisi.

e) Les jetons de MIC sont alors sautés ou échangés conformément à la Section 5.

Noter que les paramètres d'entrée *_req_flag pour l'établissement du contexte se rapportent au mécanisme choisi, comme le sont les paramètres de résultat *_state. C'est-à-dire que ces paramètres ne sont pas applicables au processus de négociation en soi.

À réception d'un jeton de négociation du côté cible, une mise en œuvre GSS-API qui ne prend pas en charge la négociation va indiquer l'état GSS_S_BAD_MECH bien qu'un mécanisme de sécurité de base particulier ait été demandé et n'était pas supporté.

Quand un accréditif GSS-API est acquis pour le mécanisme SPNEGO, la mise en œuvre DEVRAIT produire un élément accréditif pour le mécanisme SPNEGO qui contienne en interne des éléments d'accréditifs GSS-API pour tous les mécanismes pour lesquels le principal a des accréditifs disponibles, sauf pour les mécanismes qui ne sont pas à négocier, par mise en œuvre, site, ou politique spécifique d'application. Voir à l'Appendice B les interfaces pour exprimer la politique d'application.

## 4. Définitions de jetons

Les définitions types de cette section supposent des définitions de module ASN.1 de la forme suivante :

```
SPNEGOASNOneSpec { iso(1) identified-organization(3) dod(6) internet(1) security(5) mecanism(5) snego (2)
                        modules(4) spec2(2) }
ÉTIQUETTES DE DÉFINITIONS EXPLICITES ::= DÉBUT

-- le reste des définitions vient ici

FIN
```

Cela spécifie que le contexte d'étiquetage pour le module sera explicite et non automatique.

Le codage des messages du protocole SPNEGO devra obéir aux règles de codage distinctif (DER) de ASN.1, comme décrit dans [X690].

### 4.1 Types de mécanismes

Dans ce modèle de négociation, chaque OID représente un mécanisme GSS-API ou une de ses variantes (voir la Section 6) conformément à la [RFC2743].

```
MechType ::= IDENTIFIANT D'OBJET
-- L'OID représente chaque mécanisme de sécurité comme suggéré par la [RFC2743]

MechTypeList ::= SEQUENCE DE MechType
```

### 4.2 Jetons de négociation

La syntaxe des jetons de négociation initiaux suit la syntaxe de initialContextToken définie au paragraphe 3.1 de la [RFC2743]. Le pseudo mécanisme SPNEGO est identifié par l'identifiant d'objet iso.org.dod.internet.security.mecanism.snego (1.3.6.1.5.5.2). Les jetons suivants NE DOIVENT PAS être encapsulés dans ce tramage générique de jeton GSS-API.

Ce paragraphe spécifie la syntaxe du jeton interne pour le message initial et la syntaxe des jetons d'établissement de contexte suivants.

```
NegotiationToken ::= CHOIX {
     negTokenInit      [0] NegTokenInit,
     negTokenResp    [1] NegTokenResp
  }
```

#### 4.2.1 negTokenInit

```
NegTokenInit ::= SEQUENCE {
     mechTypes       [0] MechTypeList,
     reqFlags          [1] ContextFlags  FACULTATIF,
-- hérité de la RFC 2478 pour la rétro compatibilité, il est RECOMMANDÉ de le laisser de côté --
     mechToken       [2] CHAINE D'OCTETS  FACULTATIF,
     mechListMIC    [3] CHAINE D'OCTETS  FACULTATIF,
     ...
  }
ContextFlags ::= CHAINE BINAIRE {
     delegFlag       (0),
     mutualFlag      (1),
     replayFlag      (2),
     sequenceFlag  (3),
     anonFlag        (4),
     confFlag        (5),
     integFlag       (6)
} (TAILLE (32))
```

C'est la syntaxe pour le jeton interne du message de négociation initiale.

mechTypes : Ce champ contient un ou plusieurs mécanismes de sécurité disponibles pour l'initiateur, en ordre décroissant de préférence (choix favori en premier).

reqFlags : Ce champ, si il est présent, contient les options de service qui sont demandées pour établir le contexte (le paramètre req_flags de GSS_Init_sec_context()). Ce champ est hérité de la RFC 2478 et n'est pas protégé en intégrité. Pour les mises en œuvre de cette spécification, l'initiateur DEVRAIT omettre ce champ reqFlags et l'acceptant DOIT ignorer ce champ reqFlags.

La contrainte de taille du type ASN.1 ContextFlags ne s'applique qu'au type abstrait. Les DER ASN.1 exigent que tous les zéros en queue soient coupés au codage d'un type de chaîne binaire dont la définition abstraite inclut des bits désignés. Les mises en œuvre ne devraient pas s'attendre à recevoir exactement 32 bits dans un codage de ContextFlags.

mechToken : Ce champ, si présent, contient le jeton de mécanisme optimiste.

mechlistMIC : Ce champ, si présent, contient un jeton de MIC pour la liste de mécanismes dans le message de négociation initiale. Ce jeton de MIC est calculé conformément à la Section 5.

#### 4.2.2 negTokenResp

```
NegTokenResp ::= SEQUENCE {
   negState            [0] ENUMERATED {
     accept-completed    (0),
     accept-incomplete   (1),
     reject                (2),
     request-mic           (3)
   }                       FACULTATIF,                    -- EXIGÉ dans la première réponse de la cible
   supportedMech   [1] MechType   FACULTATIF,       -- présent seulement dans la première réponse de la cible
   responseToken    [2] CHAINE D'OCTETS  FACULTATIF,
   mechListMIC      [3] CHAINE D'OCTETS  FACULTATIF,
   ...
}
```

C'est la syntaxe pour tous les messages de négociation suivants.

negState : Ce champ, si présent, contient l'état de la négociation. Ce peut être :
accept-completed : aucun autre message de négociation n'est attendu de la part de l'homologue, et le contexte de sécurité est établi pour l'envoyeur.
accept-incomplete : au moins un message de négociation supplémentaire de la part de l'homologue est nécessaire pour établir le contexte de sécurité.
reject : l'envoyeur termine la négociation.
request-mic : l'envoyeur indique que l'échange des jetons de MIC, comme décrit à la Section 5, sera EXIGÉ si des services de sécurité par message sont disponibles sur le contexte de mécanisme à établir. Cette valeur DEVRA n'être présente que dans la première réponse de la cible.

Ce champ est EXIGÉ dans la première réponse de la cible, et est FACULTATIF à partir de là. Quand negState est absent, l'état réel devrait être déduit de l'état du contexte de mécanisme négocié.

supportedMech : Ce champ DEVRA n'être présent que dans la première réponse de la cible. Il DOIT être un des mécanismes offerts par l'initiateur.

ResponseToken : Ce champ, si présent, contient les jetons spécifiques du mécanisme choisi.

mechlistMIC : Ce champ, si présent, contient un jeton de MIC pour la liste de mécanismes dans le message de négociation initiale. Ce jeton de MIC est calculé conformément à la Section 5.

## 5. Traitement de mechListMIC

Si le mécanisme choisi par la négociation ne prend pas en charge la protection de l'intégrité, aucun jeton mechlistMIC n'est utilisé.

Autrement, si le mécanisme accepté est le mécanisme préféré de l'initiateur et de l'acceptant, l'échange de jeton de MIC, comme décrit plus loin, est alors FACULTATIF. Un mécanisme est le préféré de l'acceptant si il n'y a pas d'autre mécanisme que l'acceptant aurait préféré au mécanisme accepté si il avait été présent dans la liste de mécanismes.

Dans tous les autres cas, les jetons de MIC DOIVENT être échangés après l'établissement complet du contexte de mécanisme.

a) Le jeton mechlistMIC (ou simplement le jeton de MIC) est calculé sur la liste de mécanismes dans le message de négociation initiale en invoquant GSS_GetMIC() comme suit : l'entrée context_handle est le contexte de mécanisme établi, l'entrée qop_req est 0, et le message d'entrée est le codage en DER de la valeur du type MechTypeList, qui est contenue dans le champ "mechTypes" du NegTokenInit. Le message d'entrée N'EST PAS le codage en DER du type "[0] MechTypeList".

b) Si le mécanisme choisi échange un nombre pair de jetons de mécanisme (c'est-à-dire, l'acceptant envoie le dernier jeton de mécanisme) l'acceptant fait ce qui suit quand il génère le message de négociation contenant le dernier jeton de mécanisme : si l'échange de jeton de MIC est facultatif, GSS_Accept_sec_context() indique GSS_S_COMPLETE et n'inclut pas de jeton mechlistMIC, ou indique GSS_S_CONTINUE_NEEDED et inclut un jeton mechlistMIC et un état accept-incomplete ; si l'échange de jeton de MIC est exigé, GSS_Accept_sec_context() indique SS_S_CONTINUE_NEEDED et inclut un jeton mechlistMIC. Les acceptants qui souhaitent être compatibles avec les mises en œuvre SPNEGO Windows traditionnelles, comme décrit à l'Appendice C, ne devraient pas générer un jeton mechlistMIC quand l'échange de jeton de MIC n'est pas exigé. L'initiateur traite alors le dernier jeton de mécanisme, et fait une des choses suivantes :
   I) Si un jeton mechlistMIC était inclus et est correctement vérifié, GSS_Init_sec_context() indique GSS_S_COMPLETE. Le message de négociation résultant contient un jeton mechlistMIC et un état accept_complete. L'acceptant DOIT alors vérifier ce jeton mechlistMIC.
   II) Si un jeton mechlistMIC était inclus mais incorrect, la négociation DEVRA être terminée. GSS_Init_sec_context() indique GSS_S_DEFECTIVE_TOKEN.
   III) Si aucun jeton mechlistMIC n'était inclus et si l'échange de jeton de MIC n'est pas exigé, GSS_Init_sec_context() indique GSS_S_COMPLETE sans jeton résultant.
   IV) Si aucun jeton mechlistMIC n'était inclus mais si l'échange de jeton de MIC est exigé, la négociation DEVRA être terminée. GSS_Accept_sec_context() indique GSS_S_DEFECTIVE_TOKEN.

c) Dans le cas où le mécanisme choisi échange un nombre impair de jetons de mécanisme (c'est-à-dire, l'initiateur envoie

le dernier jeton de mécanisme) l'initiateur fait ce qui suit lorsque il génère le message de négociation contenant le dernier jeton de mécanisme : si le negState était request-mic dans la première réponse de la cible, un jeton mechlistMIC DOIT être inclus ; autrement, le jeton mechlistMIC est FACULTATIF. (Noter que l'échange de jeton de MIC est exigé si un mécanisme autre que le premier choix de l'initiateur est retenu.) Dans le cas où le jeton de mécanisme optimiste est le seul jeton de mécanisme pour le mécanisme préféré de l'initiateur, le jeton mechlistMIC est FACULTATIF. Que le jeton mechlistMIC soit ou non inclus, GSS_Init_sec_context() indique GSS_S_CONTINUE_NEEDED. Les initiateurs qui souhaitent être compatibles avec les mises en œuvre SPNEGO Windows traditionnelles, comme décrit à l'Appendice C, ne devraient pas générer de jeton mechlistMIC quand l'échange de jeton de MIC n'est pas exigé. L'acceptant traite alors le dernier jeton de mécanisme et fait une des choses qui suivent :

I) Si un jeton mechlistMIC était inclus et est correctement vérifié, GSS_Accept_sec_context() indique GSS_S_COMPLETE. Le message de négociation résultant contient un jeton mechlistMIC et un état accept_complete. L'initiateur DOIT alors vérifier ce jeton mechlistMIC.

II) Si un jeton mechlistMIC était inclus mais était incorrect, la négociation DEVRA être terminée. GSS_Accept_sec_context() indique GSS_S_DEFECTIVE_TOKEN.

III) Si aucun jeton mechlistMIC n'était inclus et si l'échange de jeton mechlistMIC n'est pas exigé, GSS_Accept_sec_context() indique GSS_S_COMPLETE. Le message de négociation résultant contient un état accept_complete.

IV) Dans le cas où le jeton de mécanisme optimiste est aussi le dernier jeton de mécanisme (quand le mécanisme préféré de l'initiateur est accepté par la cible) et où la cible envoie un état request-mic mais où l'initiateur n'a pas envoyé de jeton mechlistMIC, la cible DOIT alors inclure un jeton mechlistMIC dans cette première réponse. GSS_Accept_sec_context() indique GSS_S_CONTINUE_NEEDED. L'initiateur DOIT vérifier le jeton mechlistMIC reçu et générer un jeton mechlistMIC à renvoyer à la cible. La cible DEVRA, à son tour, vérifier le jeton mechlistMIC retourné et achever la négociation.

V) Si aucun jeton mechlistMIC n'était inclus et si l'acceptant a envoyé un état request-mic dans le premier message de réponse (l'échange de jetons de MIC est exigé) la négociation DEVRA être terminée. GSS_Accept_sec_context() indique GSS_S_DEFECTIVE_TOKEN.

## 6. Extensibilité

Deux mécanismes sont fournis pour l'extensibilité. D'abord, les structures ASN.1 dans cette spécification PEUVENT être étendues par action de normalisation de l'IETF. Les mises en œuvre qui reçoivent des champs inconnus DOIVENT les ignorer.

Ensuite, les OID qui correspondent à un attribut de mécanisme désiré (c'est-à-dire, des variantes de mécanisme) peuvent être inclus dans l'ensemble des mécanismes préférés par un initiateur. L'acceptant peut choisir d'honorer cette demande en préférant des mécanismes qui ont inclus ces attributs. Des travaux futurs au sein du groupe de travail Kitten sont supposés normaliser les attributs communs que les mécanismes SPNEGO peuvent souhaiter prendre en charge. Pour l'instant, il est suffisant de dire que les initiateurs PEUVENT inclure des OID qui ne correspondent pas aux mécanismes. De tels OID PEUVENT influer sur le choix de mécanisme de l'acceptant. Comme exposé à la Section 5, si il y a des mécanismes qui, si ils sont présents dans la liste des mécanismes de l'initiateur, pourraient être préférés par l'acceptant au lieu du mécanisme préféré de l'initiateur, l'acceptant DOIT demander l'échange de jeton de MIC. Par conséquent, les acceptants DOIVENT demander l'échange de jeton de MIC si ils prennent en charge la négociation d'attributs non disponibles dans le mécanisme préféré de l'initiateur, sans considérer si l'initiateur a en fait demandé ces attributs.

## 7. Considérations sur la sécurité

Afin de produire le jeton de MIC pour la liste de mécanismes, le mécanisme doit fournir la protection de l'intégrité. Quand le mécanisme choisi ne prend pas en charge la protection de l'intégrité, la négociation est vulnérable : un attaquant actif peut le forcer à utiliser un mécanisme de sécurité qui n'est pas mutuellement préféré mais est acceptable à la cible.

Ce protocole donne les garanties suivantes quand des services d'intégrité par message sont disponibles sur le contexte de mécanisme établi, et que la liste de mécanismes a été altérée par un adversaire, de telle sorte qu'un mécanisme qui n'est pas mutuellement préféré puisse être choisi :

a) si le dernier jeton de mécanisme est envoyé par l'initiateur, les deux homologues devront échouer ;
b) si le dernier jeton de mécanisme est envoyé par l'acceptant, l'acceptant ne devra pas achever et l'initiateur, au pire, devra achever avec le choix de son mécanisme préféré.

La négociation peut ne pas se terminer si une altération a été faite mais n'a pas d'impact matériel.

La protection de la négociation dépend de la force de la protection de l'intégrité. En particulier, la force de SPNEGO n'est pas supérieure à la protection d'intégrité du plus faible mécanisme acceptable aux homologues de GSS-API.

Noter que lorsque il existe plusieurs mécanismes avec des jetons de contexte similaires mais une sémantique différente, tels que certains des jetons de contexte des mécanismes ou tous puissent être aisément altérés de sorte que les jetons de contexte d'un mécanisme puissent passer pour un autre des jetons de contexte d'un mécanisme similaire, il peut alors exister une attaque en dégradation ou des attaques similaires. Par exemple, si une certaine famille de mécanismes utilise la même syntaxe de jetons de contexte pour deux variantes ou plus et dépend de l'OID dans l'enveloppe pseudo ASN.1/DER du jeton initial, mais n'assure pas la protection d'intégrité pour cet OID, il peut alors exister une attaque contre ces mécanismes. SPNEGO ne combat généralement pas de telles attaques.

Dans tous les cas, les homologues communicants sont exposés à la menace de déni de service.

## 8. Remerciements

Les auteurs souhaitent remercier Sam Hartman, Nicolas Williams, Ken Raeburn, Martin Rex, Jeff Altman, Tom Yu, Cristian Ilac, Simon Spero, et Bill Sommerfeld de leurs commentaires et suggestions durant le développement de ce document.

Luke Howard a fourni un prototype de ce protocole en Heimdal et a résolu plusieurs problèmes de la version initiale de ce document.

Eric Baize et Denis Pinkas on écrit la spécification originale SPNEGO [RFC2478] dont une partie du texte a été conservée dans ce document.

## 9. Références

### 9.1 Références normatives

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

[RFC2743] J. Linn, "Interface générique de programme d'application de service de sécurité, version 2, mise à jour 1", janvier 2000. (*MàJ par* RFC5554)

[X690] Recommandation UIT-T X.690 (1997) | norme internationale ISO/CEI 8825-1:1998, "Règles de codage ASN.1 : Spécification des règles de codage de base (BER), règles de codage canoniques (CER) et règles de codage distinctives (DER),.

### 9.2 Références pour information

[RFC2478] E. Baize, D. Pinkas, "Mécanisme de négociation GSS-API simple et protégé", décembre 1998. (*Obs., voir* RFC4178)

## Appendice A. Module ASN.1 SPNEGO

```
SPNEGOASNOneSpec { iso(1) identified-organization(3) dod(6) internet(1) security(5) mecanism(5) snego (2)
modules(4) spec2(2)
  }
ÉTIQUETTES DE DÉFINITIONS EXPLICITES ::= DÉBUT

MechType ::= IDENTIFIANT D'OBJET
-- L'OID représente chaque mécanisme de sécurité comme suggéré par la [RFC2743] --

MechTypeList ::= SEQUENCE DE MechType
```

```
NegotiationToken ::= CHOIX {
   negTokenInit      [0] NegTokenInit,
   negTokenResp    [1] NegTokenResp
}

NegTokenInit ::= SEQUENCE {
   mechTypes       [0] MechTypeList,
     reqFlags        [1] ContextFlags  FACULTATIF,
-- hérité de la RFC 2478 pour la rétro compatibilité, il est RECOMMANDÉ de le laisser de côté
    mechToken       [2] CHAINE D'OCTETS  FACULTATIF,
   mechListMIC     [3] CHAINE D'OCTETS  FACULTATIF,
   ...
}

NegTokenResp ::= SEQUENCE {
   negState       [0] ENUMERATED {
      accept-completed    (0),
      accept-incomplete   (1),
      reject                (2),
      request-mic           (3)
   }                          FACULTATIF,
-- EXIGÉ dans la première réponse de la cible
      supportedMech   [1] MechType      FACULTATIF,
-- présent seulement dans la première réponse de la cible
      responseToken   [2] CHAINE D'OCTETS  FACULTATIF,
      mechListMIC     [3] CHAINE D'OCTETS  FACULTATIF,
      ...
}

ContextFlags ::= CHAINE BINAIRE {
      delegFlag       (0),
      mutualFlag      (1),
      replayFlag      (2),
      sequenceFlag    (3),
      anonFlag        (4),
      confFlag        (5),
      integFlag       (6)
} (TAILLE (32))

FIN
```

## Appendice B API de prise en charge de négociation de GSS-API

Afin de fournir à un appelant GSS-API (l'initiateur ou la cible ou les deux) la capacité de choisir dans l'ensemble de mécanismes pris en charge, un ensemble réduit de mécanismes à négocier et deux API supplémentaires sont définis :

- GSS_Get_neg_mechs() indique à l'appelant l'ensemble de mécanismes de sécurité disponibles sur le système local pour la négociation, pour lequel des accréditifs appropriés sont disponibles.
- GSS_Set_neg_mechs() spécifie l'ensemble de mécanismes de sécurité à utiliser sur le système local par l'appelant pour la négociation, pour les accréditifs concernés.

### B.1 Invocation de GSS_Set_neg_mechs

Entrées :

- cred_handle BRIDE D'ACCREDITIFS , -- NULL spécifie les accréditifs par défaut
- mech_set ENSEMBLE D'IDENTIFIANTS D'OBJET

Résultats :

- major_status ENTIER,
- minor_status ENTIER

Retourne les codes de major_status :
- o GSS_S_COMPLETE indique que l'ensemble de mécanismes de sécurité disponibles pour la négociation a été réglé à mech_set.
- o GSS_S_FAILURE indique que l'opération demandée n'a pas pu être effectuée pour des raisons non spécifiées au niveau de GSS-API.

Cela permet aux appelants de spécifier l'ensemble de mécanismes de sécurité qui peut être négocié avec les accréditifs identifiés par cred_handle. Cet appel est destiné à prendre en charge des appelants spécialisés qui ont besoin de restreindre l'ensemble des mécanismes de sécurité négociables à partir de l'ensemble de tous les mécanismes de sécurité disponibles à l'appelant (sur la base des accréditifs disponibles). Noter que si plus d'un mécanisme est spécifié dans mech_set, l'ordre dans lequel ces mécanismes est spécifié implique une préférence relative.

### B.2 Invocation de GSS_Get_neg_mechs

Entrée :
- o cred_handle BRIDE D'ACCREDITIFS , -- NULL spécifie les accréditifs par défaut

Résultats :
- o major_status ENTIER,
- o minor_status ENTIER,
- o mech_set ENSEMBLE D'IDENTIFIANTS D'OBJET

Retourne les codes de major_status :
- o GSS_S_COMPLETE indique que l'ensemble des mécanismes de sécurité disponibles à la négociation a été retourné dans mech_set.
- o GSS_S_FAILURE indique que l'opération demandée n'a pas pu être effectuée pour des raisons non spécifiées au niveau GSS-API.

Cela permet aux appelants de déterminer l'ensemble de mécanismes de sécurité disponibles à la négociation avec l'accréditif identifié par cred_handle. Cet appel est destiné à prendre en charge les appelants spécialisés qui ont besoin de réduire l'ensemble des mécanismes de sécurité négociables à partir de l'ensemble des mécanismes de sécurité pris en charge disponibles à l'appelant (sur la base des accréditifs disponibles).

Note : La fonction GSS_Indicate_mechs() indique l'ensemble complet des types de mécanismes disponibles sur le système local. Comme cet appel n'a pas de paramètre d'entrée, l'ensemble retourné n'est pas nécessairement disponible pour tous les accréditifs.

## Appendice C. Changements par rapport à la RFC 2478

Les mises en œuvre SPNEGO de Microsoft Windows 2000/Windows XP/Windows Server 2003 ont le comportement suivant : aucun mechlistMIC n'est produit et mechlistMIC n'est pas traité si il en est fourni un ; si l'initiateur envoie le dernier jeton de mécanisme, l'acceptant va renvoyer un jeton de négociation avec un état accept_complete et pas de jeton mechlistMIC. De plus, un OID incorrect (1.2.840.48018.1.2.2) peut être utilisé pour identifier le mécanisme Kerberos version 5 de GSS-API .

Les changements suivants ont été faits pour être compatibles avec ces mises en œuvre traditionnelles.
- * NegTokenTarg est changé en negTokenResp et est le format de message pour tous les jetons de négociation suivants.
- * NegTokenInit est le message pour le message de négociation initiale, et seulement de message.
- * mechTypes dans negTokenInit n'est pas facultatif.
- * Si le mécanisme choisi est aussi le préféré pour les deux homologues, il est sûr d'omettre les jetons de MIC.

Si au moins un des deux homologues met en œuvre le pseudo mécanisme mis à jour par le présent document, la négociation est protégée.

Les changements suivants règlent des problèmes de la RFC 2478.
- * reqFlags n'est pas protégé, donc,il ne devrait pas impacter la négociation.
- * le codage en DER est exigé.
- * l'entrée GSS_GetMIC() est précisée.
- * les services d'intégrité par message sont exigés pour le mécanisme négocié.
- * deux jetons de MIC sont échangés, un dans chaque direction.

Une mise en œuvre qui se conforme à la présente spécification ne va pas interopérer avec une stricte mise en œuvre de la RFC 2748. Même si la nouvelle mise en œuvre envoie toujours un jeton mechlistMIC, elle va quand même échouer à interopérer. Si c'est un serveur, il va échouer parce qu'il demande un jeton mechlistMIC en utilisant une option que les plus anciennes mises en œuvre ne prennent pas en charge. Les clients vont tendre à échouer aussi.

Comme solution de remplacement à l'approche choisie dans cette spécification, on aurait pu documenter un comportement correct qui soit pleinement rétro compatible avec la RFC 2478 et inclure un appendice sur la façon d'interopérer avec les mises en œuvre incorrectes existantes de la RFC 2478.

En pratique les développeurs de SPNEGO au sein de l'IETF ont privilégié l'interopérabilité avec les mises en œuvre de Microsoft. On n'a pas pu choisir de conserver des garanties raisonnables de sécurité, de conserver l'interopérabilité avec les mises en œuvre de Microsoft, et conserver l'interopérabilité avec les mises en œuvre correctes de la RFC 2478.

Le groupe de travail ne connaissait aucune mise en œuvre déployée dans l'Internet de la RFC 2478. Même si il y a de telles mises en œuvre, il est peu probable qu'elles interopèrent parce que il y a une faute critique dans la description du codage des listes de mécanismes dans la RFC 2478.

Avec l'approche retenue dans la présente spécification, la sécurité est assurée tout le temps entre les nouvelles mises en œuvre tout en conservant l'interopérabilité ave les mises en œuvre déployées au sein de la communauté de l'IETF. Le groupe travail pense que cela justifie de rompre la compatibilité avec une mise en œuvre correcte de la RFC 2478.

## Appendice D. Exemple de calcul de mechListMIC

Voici un exemple pour illustrer comment le champ mechListMIC devrait être calculé.

La partie initiale du codage en DER de NegTokenInit est construite comme suit (le "nn" est une longueur de codage, éventuellement de plus de un octet) :

30 – octet identifiant pour la SEQUENCE construite (NegTokenInit)
nn -- longueur

-- les octets de contenu de la SEQUENCE commencent par le codage en DER de "[0] MechTypeList":
A0 – octet identifiant pour constructed [0]
nn -- longueur

-- le contenu de constructed [0] est le codage en DER die MechTypeList (qui est une SEQUENCE):
30 -- octet identifiant pour constructed SEQUENCE
nn -- longueur

-- les octets de contenu de la SEQUENCE commencent par le codage en DER de IDENTIFIANT D'OBJET :
06 -- octet identifiant pour la primitive IDENTIFIANT D'OBJET
09 --longueur
2A 86 48 86 F7 12 01 02 02 -- Kerberos V5 -- {1 2 840 113554 1 2 2}

Si une mechlistMIC a besoin d'être générée (conformément aux règles de la Section 5) elle est calculée en utilisant le codage en DER du type de données MechTypeList à partie du jeton NegTokenInit de l'initiateur comme entrée à la fonction GSS_GetMIC(). Dans ce cas, la MIC serait calculée sur les octets suivants :
codage DER de MechTypeList:
30 nn 06 09 2A 86 48 86 F7 12 01 02 02 ...

Noter que l'octet identifiant et le ou les octets de longueur pour constructed [0] (A0 nn) ne sont pas inclus dans le calcul de la MIC.

## Adresse des auteurs

Larry Zhu
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US
mél : lzhu@microsoft.com

Paul Leach
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US
mél : paulle@microsoft.com

Karthik Jaganathan
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US
mél : karthikj@microsoft.com

Wyllys Ingersoll
Sun Microsystems
1775 Wiehle Avenue,
Reston, VA 20190
US
mél : wyllys.ingersoll@sun.com

## Déclaration complète de droits de reproduction



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