Groupe de travail Réseau
Request for Comments : 2744
RFC rendue obsolèe : 1509
Catégorie : En cours de normalisation

J. Wray
Iris Associates
janvier 2000
Traduction Claude Brière de L'Isle

API de service générique de sécurité version 2 : liaisons C

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

RésuméLe présent document spécifie les liaisons en langage C pour la version 2, mise à jour 1 de l'interface de programme d'application de service générique de sécurité (GSS-API, *Generic Security Service Application Program Interface*) qui est décrite à un niveau conceptuel indépendant du langage dans la [RFC2743]. Il rend obsolète la RFC1509, faisant des changements incrémentaires spécifiques en réponse à l'expérience de mise en œuvre et aux demandes de liaison. Il est donc prévu que le présent mémoire ou une de ses versions ultérieures devienne la base pour les progrès de la spécification de GSS-API sur la voie de la normalisation.

L'interface de programme d'application de service générique de sécurité fournit des services de sécurité à ses appelants, et elle est destinée à être mise en œuvre par dessus divers mécanismes de chiffrement sous-jacents. Les appelants, GSS-API seront normalement les protocoles d'application dans lesquels des améliorations de sécurité sont intégrées par l'invocation de services fournis par la GSS-API. GSS-API permet à une application appelante d'authentifier l'identité d'un principal associé à une application homologue, pour déléguer des droits à un homologue, et appliquer des services de sécurité tels que la confidentialité et l'intégrité message par message.

Table des Matières

1. Introduction

L'interface de programmation d'application de service générique de sécurité [RFC2743] fournit des services de sécurité aux applications appelantes. Elle permet à l'application communicante d'authentifier l'usager associé à une autre application, de déléguer des droits à une autre application, et d'appliquer des services de sécurité tels que la confidentialité et l'intégrité message par message.

Il y a quatre niveaux d'utilisation de GSS-API :

a) L'application acquiert un ensemble d'accréditifs avec lesquels elle peut prouver son identité aux autres processus. Les accréditifs de l'application se portent garant de son identité globale, qui peut être ou non en rapport avec tout nom d'utilisateur local sous lequel elle pourrait fonctionner.

b) Une paire d'applications communicantes établissent un contexte de sécurité conjoint en utilisant leurs accréditifs. Le contexte de sécurité est une paire de structures de données GSS-API qui contient des informations d'état partagées, qui sont nécessaires afin de pouvoir fournir des services de sécurité par message. Des exemples d'états qui pourraient être partagés entre des applications au titre d'un contexte de sécurité sont les clés de chiffrement, et les numéros de séquence de message. Au titre de l'établissement d'un contexte de sécurité, l'initiateur de contexte est authentifié auprès du

répondant, et peut exiger que le répondant soit authentifié à son tour. L'initiateur peut facultativement donner au répondant le droit d'initier d'autres contextes de sécurité, agissant comme un agent ou délégué de l'initiateur. Ce transfert de droits est appelé délégation, et est réalisé en créant un ensemble d'accréditifs, similaire à ceux utilisés par l'application initiatrice, mais qui peuvent être utilisés par le répondant.

Pour établir et maintenir les informations partagées qui constituent le contexte de sécurité, certains appels GSS-API vont retourner une structure de données de jeton, type de données opaque qui peut contenir des données protégées cryptographiquement. L'appelant d'un tel sous-programme GSS-API est responsable du transfert du jeton à l'application homologue, en l'encapsulant si nécessaire dans un protocole d'application à application. À réception d'un tel jeton, l'application homologue devrait le passer à un sous-programme GSS-API correspondant qui va décoder le jeton et extraire les informations, mettant à jour en conséquence les informations d'état du contexte de sécurité.

c) Les services par message sont invoqués pour s'appliquer, soit à l'intégrité et l'authentification de l'origine des données, soit à la confidentialité, l'intégrité et l'authentification de l'origine des données, aux données d'application, qui sont traitées par GSS-API comme des chaînes d'octets arbitraires. Une application qui transmet un message qu'elle souhaite protéger va appeler le sous-programme GSS-API approprié (gss_get_mic ou gss_wrap) pour appliquer la protection, en spécifiant le contexte de sécurité approprié, et envoyer le jeton résultant à l'application receveuse. Le receveur va passer le jeton reçu (et, dans le cas de données protégées par gss_get_mic, le message de données qui l'accompagne) au sous-programme de décodage correspondant (gss_verify_mic ou gss_unwrap) pour supprimer la protection et valider les données.

d) À l'achèvement d'une session de communications (qui peut s'étendre sur plusieurs connexions de transport) chaque application appelle un sous-programme GSS-API pour supprimer le contexte de sécurité. Plusieurs contextes peuvent aussi être utilisés (soit successivement, soit simultanément) au sein d'une seule association de communications, au choix des applications.

2. Sous-programmes GSS-API

Cette section fait la liste des sous-programmes qui constituent GSS-API, et offre une brève description de l'objet de chaque sous-programme. Une description détaillée de chaque sous-programme figure en ordre alphabétique à la Section 5.

Tableau 2-1 : Sous-programmes de gestion d'accréditifs de GSS-API

Sous-programme	Paragraphe	Fonction
gss_acquire_cred	5.2	Suppose une identité globale ; obtient un lien d'accréditif GSS-API pour les accréditifs préexistants.
gss_add_cred	5.3	Construit les accréditifs de façon incrémentaire.
gss_inquire_cred	5.21	Obtient des informations sur un accréditif.
gss_inquire_cred_by_mech	5.22	Obtient des informations par mécanisme sur un accréditif.
gss_release_cred	5.27	Élimine un lien d'accréditif.

Tableau 2-2 : Sous-programmes GSS-API de niveau contexte

Sous-programme	Paragraphe	Fonction
gss_init_sec_context	5.19	Initie un contexte de sécurité avec une application homologue.
gss_accept_sec_context	5.1	Accepte un contexte de sécurité initié par une application homologue.
gss_delete_sec_context	5.9	Élimine un contexte de sécurité.
gss_process_context_token	5.25	Traite un jeton sur un contexte de sécurité d'une application homologue.
gss_context_time	5.7	Détermine combien de temps un contexte va rester valide.
gss_inquire_context	5.20	Obtient des informations sur un contexte de sécurité.
gss_wrap_size_limit	5.34	Détermine la limite de taille de jeton pour gss_wrap sur un contexte.
gss_export_sec_context	5.14	Transfère un contexte de sécurité à un autre processus.
gss_import_sec_context	5.17	Importe un contexte transféré.

Tableau 2-3 : Sous-programmes GSS-API par message

Sous-programme	Paragraphe	Fonction
gss_get_mic	5.15	Calcule un code d'intégrité de message cryptographique (MIC) pour un message ; service d'intégrité.
gss_verify_mic	5.32	Vérifie un MIC par rapport à un message ; vérifie l'intégrité d'un message reçu.
gss_wrap	5.33	Lie un MIC à un message, et peut chiffrer le contenu du message ; confidentialité.
gss_unwrap	5.31	Vérifie un message avec le MIC lié, et déchiffre si nécessaire le contenu du message.

Tableau 2-4 : Sous-programmes GSS-API de manipulation de nom

Sous-programme	Paragraphe	Fonction
gss_import_name	5.16	Convertit un nom de chaîne contiguë en forme interne.
gss_display_name	5.10	Convertit un nom de forme interne en texte.
gss_compare_name	5.6	Compare deux noms de forme interne.
gss_release_name	5.28	Élimine un nom de forme interne.
gss_inquire_names_for_mech	5.24	Énumère les types de nom acceptés par le mécanisme spécifié.
gss_inquire_mechs_for_name	5.23	Énumère les mécanismes qui acceptent le type de nom spécifié.
gss_canonicalize_name	5.5	Convertit un nom interne en nom de mécanisme (MN).
gss_export_name	5.13	Convertit un MN en forme export.
gss_duplicate_name	5.12	Crée une copie d'un nom interne.

Tableau 2-5 : Sous-programmes GSS-API divers

Sous-programme	Paragraphe	Fonction
gss_add_oid_set_member	5.4	Ajoute un identifiant d'objet à un ensemble.
gss_display_status	5.11	Convertit un code d'état GSS-API en texte.
gss_indicate_mechs	5.18	Détermine les mécanismes d'authentification sous-jacents disponibles.
gss_release_buffer	5.26	Élimine une mémoire tampon.
gss_release_oid_set	5.29	Élimine un ensemble d'identifiants d'objet.
gss_create_empty_oid_set	5.8	Crée un ensemble ne contenant pas d'identifiant d'objet.
gss_test_oid_set_member	5.30	Détermine si un identifiant d'objet est membre d'un ensemble.

Les mises en œuvre GSS-API individuelles peuvent augmenter ces sous-programmes en fournissant des sous-programmes spécifiques de mécanisme supplémentaires si la fonctionnalité requise n'est pas disponible à partir des formes génériques. Les applications sont invitées à utiliser les sous-programmes génériques chaque fois que possible dans l'intérêt de la portabilité.

3. Types de données et conventions d'invocation

Les conventions suivantes sont utilisées par les liens de langage C de GSS-API :

3.1 Types d'entier

GSS-API utilise les type de données d'entiers suivants :

OM_uint32 entier de 32 bits non signé

Ce type de données portable est utilisé par les définitions de sous-programmes GSS-API lorsque le compte de bits minimum garanti est important. Les mises en œuvre GSS-API individuelles vont inclure les définitions de typedef appropriées pour transposer ce type en un type de données incorporé. Si la plateforme accepte le fichier d'en-tête X/Open xom.h, la définition OM_uint32 qui y est contenue devrait être utilisée ; le fichier d'en-tête GSS-API de l'Appendice A contient une logique qui va détecter l'inclusion de xom.h, et ne tentera pas de redéclarer OM_uint32. Si le fichier d'en-tête X/Open est non disponible sur la plateforme, la mise en œuvre GSS-API devrait utiliser le plus petit type d'entier naturel non signé qui donne au moins 32 bits de précision.

3.2 Chaîne et données similaires

Beaucoup de sous-programmes GSS-API prennent des arguments et retournent des valeurs qui décrivent des chaînes d'octets contiguës. Toutes ces données sont passées de GSS-API à l'appelant en utilisant le type de données gss_buffer_t. Ce type de données est un pointeur sur un descripteur de mémoire tampon, qui consiste en un champ de longueur qui contient le nombre d'octets total de l'élément de données, et un champ de valeur qui contient un pointeur sur l'élément de données réel.

```
typedef struct      gss_buffer_desc_struct {
  size_t                longueur ;
  void                  *valeur ;
} gss_buffer_desc,      *gss_buffer_t ;
```

La mémorisation pour les données retournées à l'application par un sous-programme GSS-API qui utilise les conventions gss_buffer_t est allouée par le sous-programme GSS-API. L'application peut libérer cette mémorisation en invoquant le sous-programme gss_release_buffer. L'allocation de l'objet gss_buffer_desc est toujours de la responsabilité de l'application ; les objets gss_buffer_desc non utilisés peuvent être initialisés à la valeur GSS_C_EMPTY_BUFFER.

3.2.1 Types de données opaques

Certains éléments de données de plusieurs mots sont considérés comme des types de données opaques pour GSS-API, parce que leur structure interne n'a pas de signification pour GSS-API ou pour l'appelant. Des exemples de tels types de données opaques sont le paramètre input_token *(jeton d'entrée)* de gss_init_sec_context (qui est opaque pour l'appelant) et le paramètre input_message *(message d'entrée)* de gss_wrap (qui est opaque pour GSS-API). Les données opaques sont passées entre GSS-API et l'application qui utilisé le type de données gss_buffer_t.

3.2.2 Chaînes de caractères

Certains éléments de données de plusieurs mots peuvent être vus comme de simples chaînes de caractères ISO Latin-1. Des exemples en sont les chaînes imprimables passées à gss_import_name via le paramètre input_name_buffer. Certains sous-programmes GSS-API retournent aussi des chaînes de caractères. Toutes ces chaînes de caractères sont passées entre l'application et la mise en œuvre GSS-API en utilisant le type de données gss_buffer_t, qui est un pointeur sur l'objet gss_buffer_desc.

Lorsque un objet gss_buffer_desc décrit une chaîne imprimable, le champ de longueur de la gss_buffer_desc ne devrait compter que les caractères imprimables dans la chaîne. En particulier, un caractère NUL en queue NE DEVRAIT PAS être inclus dans le compte de longueur, pas plus que la mise en œuvre GSS-API ou l'application ne devrait supposer la présence d'un NUL non compté en queue.

3.3 Identifiant d'objets

Certaines procédures GSS-API prennent des paramètres du type gss_OID, ou Identifiant d'objet. C'est un type qui contient des valeurs définies par l'ISO structurées en arborescence, et qui est utilisé par les appelants GSS-API pour choisir un mécanisme de sécurité sous-jacent et pour spécifier des espaces de noms. Une valeur de type gss_OID a la structure suivante :

```
typedef struct      gss_OID_desc_struct {
  OM_uint32           longueur ;
  void                *éléments ;
} gss_OID_desc,     *gss_OID ;
```

Le champ éléments de cette structure pointe sur le premier octet d'une chaîne d'octets contenant le codage ASN.1 BER de la portion valeur du codage normal de TLV en BER du gss_OID. Le champ de longueur contient le nombre d'octets dans cette valeur. Par exemple, la valeur de gss_OID correspondant à {iso(1) identified-organization(3) icd-ecma(12) member-company(2) dec(1011) cryptoAlgorithms(7) DASS(5)}, qui signifie le mécanisme d'authentification DASS X.509, a un champ de longueur 7 et un champ éléments qui pointe sur sept octets contenant les valeurs octales suivantes : 53,14,2,207,163,7,5. Les mises en œuvre GSS-API devraient fournir des valeurs constantes de gss_OID pour permettre aux applications de demander tout mécanisme accepté, bien que les applications soient encouragées au nom de la portabilité à accepter le mécanisme par défaut. Les valeurs de gss_OID devraient aussi être fournies de façon à permettre aux applications de spécifier des types de noms particuliers (voir le paragraphe 3.10). Les applications devraient traiter les valeurs de gss_OID_desc retournées par les sous-programmes GSS-API en lecture seule. En particulier, l'application ne devrait pas tenter de les désallouer avec free(). Le type de données gss_OID_desc est équivalent au type de données X/Open OM_object_identifier [XOM].

3.4 Ensembles d'identifiants d'objet

Certaines procédures GSS-API prennent des paramètres du type gss_OID_set. Ce type représente un ou plusieurs identifiants d'objet (paragraphe 2.3). Un objet gss_OID_set a la structure suivante :

```
typedef struct      gss_OID_set_desc_struct {
  size_t              compte ;
  gss_OID             éléments ;
} gss_OID_set_desc, *gss_OID_set ;
```

Le champ compte contient le nombre d'OID dans l'ensemble. Le champ éléments est un pointeur sur une rangée d'objets gss_OID_desc, dont chacun décrit un seul OID. Les valeurs de gss_OID_set sont utilisées pour désigner les mécanismes disponibles acceptés par la GSS-API, pour demander l'utilisation de mécanismes spécifiques, et pour indiquer quels mécanismes accepte un certain accréditif.

Tous les ensembles d'OID retournés à l'application par GSS-API sont des objets dynamiques (le gss_OID_set_desc, la rangée "éléments" de l'ensemble, et la rangée "éléments" de chaque OID membre sont tous alloués de façon dynamique) et cette mémorisation doit être désallouée par l'application en utilisant le sous-programme gss_release_oid_set().

3.5 Accréditifs

Un lien d'accréditif est un élément de données atomique opaque à l'appelant qui identifie une structure de données d'accréditif GSS-API. Il est représenté par le type gss_cred_id_t opaque à l'appelant, qui devrait être mis en œuvre comme un pointeur ou type arithmétique. Si une mise en œuvre de pointeur est choisie, il faut veiller à s'assurer que deux valeurs de gss_cred_id_t peuvent être comparées avec l'opérateur ==.

Les accréditifs GSS-API peuvent contenir des données d'authentification du principal spécifiques du mécanisme pour plusieurs mécanismes. Un accréditif GSS-API se compose d'un ensemble d'éléments d'accréditif, dont chacun est applicable à un seul mécanisme. Un accréditif peut contenir au plus un élément d'accréditif pour chaque mécanisme pris en charge. Un élément d'accréditif identifie les données nécessaires pour qu'un seul mécanisme authentifie un seul principal, et contient conceptuellement deux références d'accréditif qui décrivent les données réelles d'authentification spécifiques du mécanisme, une à utiliser par GSS-API pour initier les contextes, et une à utiliser pour accepter les contextes. Pour les mécanismes qui ne font pas la distinction entre accréditifs d'accepteur et d'initiateur, les deux références vont pointer sur les mêmes données d'authentification spécifiques du mécanisme sous-jacent.

Les accréditifs décrivent un ensemble de principaux spécifiques du mécanisme, et donnent à leur détenteur la capacité d'agir comme n'importe lequel de ces principaux. Toutes les identités de principal affirmées par un seul accréditif GSS-API devraient appartenir à la même entité, bien que la mise en application de cette propriété soit une affaire spécifique de la mise en œuvre. GSS-API ne rend pas les accréditifs réels disponibles aux applications ; on utilise plutôt un lien d'accréditif pour identifier un accréditif particulier, détenu en interne par GSS-API. La combinaison du lien d'accréditif GSS-API et du mécanisme identifie le principal dont l'identité sera certifiée par l'accréditif lorsque utilisé avec ce mécanisme.

Les sous-programmes gss_init_sec_context et gss_accept_sec_context permettent de spécifier la valeur GSS_C_NO_CREDENTIAL comme leur paramètre de lien d'accréditif. Ce lien d'accréditif particulier indique le désir de l'application d'agir comme un principal par défaut. Bien que les mises en œuvre GSS-API individuelles soient libres de déterminer un tel comportement par défaut comme approprié pour le mécanisme, le comportement par défaut suivant de ces sous-programmes est recommandé pour la portabilité :

gss_init_sec_context

1) Si il y a un seul principal capable d'initier des contextes de sécurité pour le mécanisme choisi au nom duquel l'application est autorisée à agir, ce principal devra alors être utilisé ; autrement,
2) si la plateforme entretient un concept d'identité réseau par défaut pour le mécanisme choisi, et si l'application est autorisée à agir au nom de cette identité pour les besoins d'initiation des contextes de sécurité, le principal correspondant à cette identité devra alors être utilisé ; autrement,
3) si la plateforme entretient un concept d'identité locale par défaut, et fournit un moyen de transposer les identités locales en identités réseau pour le mécanisme choisi, et si l'application est autorisée à agir au nom de l'image d'identité réseau de l'identité locale par défaut pour les besoins de l'initiation des contextes de sécurité en utilisant le mécanisme choisi, alors le principal correspondant à cette identité devra être utilisé ; autrement,
4) on devrait utiliser une identité par défaut configurable par l'utilisateur.

gss_accept_sec_context

1) Si il y a seulement une identité de principal autorisé capable d'accepter des contextes de sécurité pour le mécanisme choisi, ce principal devra alors être utilisé ; autrement,
2) si le mécanisme peut déterminer l'identité du principal cible en examinant le jeton d'établissement de contexte, et si l'application acceptante est autorisée à agir comme ce principal pour les besoins de l'acceptation des contextes de sécurité en utilisant le mécanisme choisi, cette identité de principal devra alors être utilisée ; autrement,
3) si le mécanisme prend en charge l'acceptation de contexte par tout principal, et si l'authentification mutuelle n'était pas demandée, tout principal dont l'application est autorisée à accepter des contextes de sécurité en utilisant le mécanisme choisi peut être utilisé ; autrement,
4) on devrait utiliser une identité par défaut configurable par l'utilisateur.

L'objet des règles ci-dessus est de permettre que les contextes de sécurité soient établis par aussi bien l'initiateur que l'accepteur en utilisant le comportement par défaut chaque fois que possible. Les applications qui demandent le comportement par défaut seront probablement plus portables à travers les mécanismes et plateformes que ceux qui utilisent gss_acquire_cred pour demander une identité spécifique.

3.6 Contextes

Le type de données gss_ctx_id_t contient une valeur atomique opaque à l'appelant qui identifie une extrémité d'un contexte de sécurité GSS-API. Il devrait être mis en œuvre comme un pointeur ou un type arithmétique. Si un type pointeur est choisi, il faut veiller à s'assurer que deux valeurs de gss_ctx_id_t peuvent être comparées avec l'opérateur ==.

Le contexte de sécurité détient des informations d'état sur chaque extrémité d'une communication entre homologues, incluant les informations d'état cryptographiques.

3.7 Jetons d'authentification

Un jeton est un type opaque à l'appelant que GSS-API utilise pour maintenir la synchronisation entre les structures de données de contexte à chaque extrémité d'un contexte de sécurité GSS-API. Le jeton est une chaîne d'octets protégée cryptographiquement, générée par le mécanisme sous-jacent à une extrémité d'un contexte de sécurité GSS-API pour être utilisé par le mécanisme homologue à l'autre extrémité. L'encapsulation (si elle est requise) et le transfert du jeton sont de la responsabilité des homologues de l'application. Un jeton est passé entre la GSS-API et l'application en utilisant les conventions gss_buffer_t.

3.8 Jetons inter processus

Certains sous-programmes GSS-API sont destinés à transférer des données entre les processus dans les programmes multi processus. Ces sous-programmes utilisent une chaîne d'octets opaque à l'appelant, générée par la GSS-API dans un procès pour utilisation par la GSS-API dans un autre processus. L'application appelante est responsable du transfert de ces jetons entre les procès d'une manière qui est spécifique du système d'exploitation. Noter que, bien que les mises en œuvre de GSS-API soient invitées à éviter de placer des informations sensibles au sein des jetons interprocès, ou de les protéger cryptographiquement, de nombreuses mises en œuvre ne seront pas capables d'éviter de placer du matériel de clés ou autres données sensibles dans les jetons. Il est de la responsabilité de l'application de s'assurer que les jetons interprocessus sont protégés dans le transit, et ne sont transférés qu'aux processus qui sont dignes de confiance. Un jeton interprocessus est passé entre la GSS-API et l'application en utilisant les conventions gss_buffer_t.

3.9 Valeurs d'état

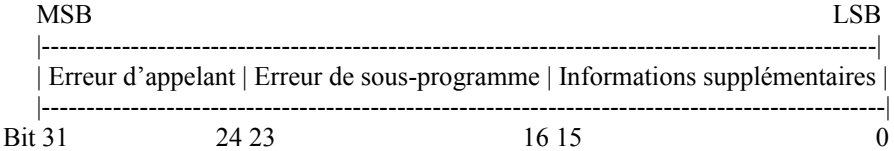
Tout sous-programme GSS-API retourne deux valeurs distinctes pour faire rapport des informations d'état à l'appelant : les codes d'état GSS et les codes d'état de mécanisme.

3.9.1 Codes d'état GSS

Les sous-programmes GSS-API retournent les codes d'état GSS comme leur valeur de fonction OM_uint32. Ces codes indiquent les erreurs qui sont indépendantes du ou des mécanismes sous-jacents utilisés pour fournir le service de sécurité. Les erreurs qui peuvent être indiquées via un code d'état GSS sont soit des erreurs génériques de sous-programme API (erreurs qui sont définies dans la spécification GSS-API) soit des erreurs de l'appelant (erreurs qui sont spécifiques de ces liens de langage).

Un code d'état GSS peut indiquer une seule erreur fatale générique d'API provenant du sous-programme et une seule erreur d'appelant. De plus, des informations d'état supplémentaires peuvent être indiquées via l'établissement de bits dans le champ d'informations supplémentaires d'un code d'état GSS.

Ces erreurs sont codées dans les 32 bits du code d'état GSS comme suit :



Donc si un sous-programme GSS-API retourne un code d'état GSS dont les 16 bits supérieurs contiennent une valeur non

zéro, l'appel échoue. Si le champ erreur d'appelant est non zéro, l'appel de l'application invoquant le sous-programme était erroné. Les erreurs d'invocation sont définies au tableau 3-1. Si le champ Erreurs de sous-programme est non zéro, le sous-programme a échoué pour une des raisons spécifiques du sous-programme énumérées ci-dessous dans le tableau 3-2. Que les 16 bits supérieurs indiquent ou non un échec ou un succès, le sous-programme communique des informations supplémentaires en établissant des bits dans le champ informations supplémentaires du code d'état. La signification des bits individuels figure ci-dessous dans le tableau 3-3.

Tableau 3-1 : Erreurs d'invocation

Nom	Valeur dans le champ	Signification
GSS_S_CALL_INACCESSIBLE_READ	1	Un paramètre d'entrée exigé n'a pas pu être lu.
GSS_S_CALL_INACCESSIBLE_WRITE	2	Un paramètre de sortie exigé n'a pas pu être écrit.
GSS_S_CALL_BAD_STRUCTURE	3	Un paramètre était mal formé.

Tableau 3-2 : Erreurs de sous-programme

Nom	Valeur dans le champ	Signification
GSS_S_BAD_MECH	1	Un mécanisme non pris en charge a été demandé.
GSS_S_BAD_NAME	2	Un nom invalide a été fourni.
GSS_S_BAD_NAMETYPE	3	Un nom fourni était d'un type non accepté.
GSS_S_BAD_BINDINGS	4	Des liens de canal incorrects ont été fournis.
GSS_S_BAD_STATUS	5	Un code d'état invalide a été fourni.
GSS_S_BAD_MIC GSS_S_BAD_SIG	6	Un jeton avait un MIC invalide.
GSS_S_NO_CRED	7	Aucun accréditif n'a été fourni, ou les accréditifs étaient indisponibles ou inaccessibles.
GSS_S_NO_CONTEXT	8	Aucun contexte n'a été établi.
GSS_S_DEFECTIVE_TOKEN	9	Un jeton était invalide.
GSS_S_DEFECTIVE_CREDENTIAL	10	Un accréditif était invalide.
GSS_S_CREDENTIALS_EXPIRED	11	Les accréditifs référencés sont périmés.
GSS_S_CONTEXT_EXPIRED	12	Le contexte est arrivé à expiration.
GSS_S_FAILURE	13	Défaillance diverse (voir le texte).
GSS_S_BAD_QOP	14	La qualité de protection demandée n'a pas pu être fournie.
GSS_S_UNAUTHORIZED	15	L'opération est interdite par la politique de sécurité locale.
GSS_S_UNAVAILABLE	16	L'opération ou option est indisponible.
GSS_S_DUPLICATE_ELEMENT	17	L'élément d'accréditif demandé existe déjà.
GSS_S_NAME_NOT_MN	18	Le nom fourni n'est pas un nom de mécanisme.

Tableau 3-3 : Bits d'état supplémentaires

Nom	n° de bit	Signification
GSS_S_CONTINUE_NEEDED	0 (LSB)	Retourné seulement par gss_init_sec_context ou gss_accept_sec_context. Le sous-programme doit être invoqué de nouveau pour achever sa fonction. Voir la description détaillée dans la documentation du sous-programme.
GSS_S_DUPLICATE_TOKEN	1	Le jeton était un dupliqué d'un jeton antérieur.
GSS_S_OLD_TOKEN	2	La période de validité du jeton est terminée.
GSS_S_UNSEQ_TOKEN	3	Un jeton ultérieur a déjà été traité.
GSS_S_GAP_TOKEN	4	Un jeton par message attendu n'a pas été reçu.

La documentation de sous-programme utilise aussi le nom GSS_S_COMPLETE, qui est une valeur de zéro, pour indiquer une absence de toute erreur d'API ou de bits d'informations supplémentaires.

Tous les symboles GSS_S_xxx équivalent à compléter les codes d'état OM_uint32, plutôt que des valeurs de champ binaire. Par exemple, la valeur réelle du symbole GSS_S_BAD_NAMETYPE (valeur 3 dans le champ erreur de sous-programme) est 3<<16. Les macros GSS_CALLING_ERROR(), GSS_ROUTINE_ERROR() et GSS_SUPPLEMENTARY_INFO() sont fournies, dont chacune prend un code d'état GSS et supprime tout sauf le champ pertinent. Par exemple, la valeur obtenue en appliquant GSS_ROUTINE_ERROR à un code d'état supprime les champs d'erreurs d'invocation et d'informations supplémentaires, laissant seulement le champ erreurs de sous-programme. Les valeurs délivrées par ces macros peuvent être directement comparées avec un symbole GSS_S_xxx du type approprié. La macro GSS_ERROR() est aussi fournie, qui lorsque appliquée à un code d'état GSS retourne une valeur non zéro si le code

d'état indique une erreur d'invocation ou de sous-programme, et une valeur zéro autrement. Toutes les macros définies par GSS-API évaluent leur ou leurs arguments exactement une fois.

Une mise en œuvre GSS-API peut choisir de signaler les erreurs d'invocation d'une manière spécifique de la plateforme plutôt que, ou en plus de, la valeur du sous-programme ; les erreurs de sous-programme et d'informations supplémentaires devraient être retournées seulement via des valeurs d'état majeur.

Le code d'état majeur GSS GSS_S_FAILURE est utilisé pour indiquer que le mécanisme sous-jacent a détecté une erreur pour laquelle aucun code d'état GSS spécifique n'est défini. Le code d'état spécifique du mécanisme fournira plus de détails sur l'erreur.

3.9.2 Codes d'état spécifiques de mécanisme

Les sous-programmes GSS-API retournent un paramètre minor_status, qui est utilisé pour indiquer des erreurs spécialisées de la part du mécanisme de sécurité sous-jacent. Ce paramètre peut contenir une seule erreur spécifique du mécanisme, indiquée par une valeur OM_uint32.

Le paramètre minor_status va toujours être établi par un sous-programme GSS-API, même si il retourne une erreur d'invocation ou une des erreurs d'API génériques indiquées ci-dessus comme fatale, bien que la plupart des autres paramètres de sortie puissent rester non établis dans de pareils cas. Cependant, les paramètres de sortie qui sont supposés retourner des pointeurs sur des mémorisations allouées par un sous-programme doivent toujours être établis par le sous-programme, même dans le cas d'une erreur, bien qu'alors le sous-programme GSS-API puisse choisir de régler la valeur du paramètre retourné à NUL pour indiquer qu'aucune mémorisation n'est en fait allouée. Tout champ de longueur associé à de tels pointeurs (comme dans une structure gss_buffer_desc) devrait aussi être réglé à zéro dans de tels cas.

3.10 Noms

Un nom est utilisé pour identifier une personne ou entité. GSS-API authentifie les relations entre un nom et l'entité qui revendique le nom.

Comme différents mécanismes d'authentification peuvent employer des espaces de noms différents pour identifier leurs principaux, la prise en charge des dénominations GSS-API est nécessairement complexe dans les environnements multi mécanismes (ou même dans certains environnements à un seul mécanisme où le mécanisme sous-jacent prend en charge plusieurs espaces de noms).

Deux représentations distinctes sont définies pour les noms :

Forme interne : c'est le format "natif" GSS-API pour les noms, représenté par le type gss_name_t spécifique de la mise en œuvre. Il est opaque pour les appelants GSS-API. Un seul objet gss_name_t peut contenir plusieurs noms provenant de différents espaces de noms, mais tous les noms devraient se référer à la même entité. Un exemple d'un tel nom interne serait le nom retourné d'un appel au sous programme gss_inquire_cred, lorsque appliqué à un accréditif contenant des éléments d'accréditif pour plusieurs mécanismes d'authentification employant des espaces de noms différents. Cet objet gss_name_t va contenir un nom distinct pour l'entité pour chaque mécanisme d'authentification.

Pour les mises en œuvre GSS-API qui prennent en charge plusieurs espaces de noms, les objets de type gss_name_t doivent contenir des informations suffisantes pour déterminer l'espace de noms auquel chaque nom primitif appartient.

Formes de chaîne d'octets contigus spécifiques du mécanisme : format capable de contenir un seul nom (d'un seul espace de noms). Les noms de chaîne contigus sont toujours accompagnés d'un identifiant d'objet qui spécifie l'espace de noms auquel appartient le nom, et leur format dépend du mécanisme d'authentification qui emploie le nom. Beaucoup de noms de chaîne contigus, mais pas tous, seront imprimables, et peuvent donc être utilisés par les applications GSS-API pour la communication avec leurs utilisateurs.

Les sous-programmes (gss_import_name et gss_display_name) sont fournis pour convertir les noms entre les représentations de chaînes contiguës et le type interne gss_name_t. gss_import_name peut accepter plusieurs syntaxes pour chaque espace de nom accepté, permettant aux utilisateurs de choisir une représentation de nom préférée. gss_display_name devrait utiliser une syntaxe imprimable choisie par la mise en œuvre pour chaque type de noms accepté.

Si une application invoque gss_display_name(), passant le nom interne résultant d'une invocation à gss_import_name(), il n'est pas garanti que le nom de chaîne contiguë résultant soit le même que le nom de chaîne importée originale. Pas plus que les identifiants d'espace de nom ne survivent nécessairement inchangés après un voyage sous la forme de nom interne. Un exemple pourrait être un mécanisme qui authentifie des nom X.500, mais fournit un algorithme de transposition des

noms DNS Internet en X.500. La mise en œuvre de gss_import_name() de ce mécanisme pourrait, lorsque on lui présente un nom DNS, générer un nom interne qui contiendrait à la fois le nom DNS d'origine et le nom X.500 équivalent. Autrement, il pourrait ne mémoriser que le nom X.500. Dans ce dernier cas, gss_display_name() génèrerait probablement un nom X.500 imprimable, plutôt que le nom DNS d'origine.

Le processus d'authentification livre un nom interne à l'accepteur du contexte. Comme ce nom a été authentifié par un seul mécanisme, il ne contient qu'un seul nom (même si le nom interne présenté par l'initiateur de contexte à gss_init_sec_context avait plusieurs composants). De tels noms sont appelés des noms de mécanisme internes, ou "MN" et les noms émis par gss_accept_sec_context() sont toujours de ce type. Comme certaines applications peuvent exiger des MN sans vouloir subir la redondance d'une opération d'authentification, une seconde fonction, gss_canonicalize_name(), est fournie pour convertir un nom interne général en un MN.

La comparaison des noms de forme interne peut être accomplie via le sous-programme gss_compare_name(), qui retourne VRAI si les deux noms comparés se réfèrent à la même entité. Cela supprime le besoin que les programmes d'application e comprennent les syntaxes des divers noms imprimables qu'une certaine mise en œuvre GSS-API peut prendre en charge. Comme GSS-API suppose que tous les noms primitifs contenus dans un certain nom interne se réfèrent à la même entité, gss_compare_name() peut retourner VRAI si les deux noms ont au moins un nom primitif en commun. Si la mise en œuvre incorpore la connaissance des relations d'équivalence entre les noms pris dans les différents espaces de noms, cette connaissance peut aussi permettre la réussite de la comparaison des noms internes ne contenant pas d'éléments primitifs qui se chevauchent.

Lorsque ils sont utilisés sur de grandes listes de contrôle d'accès *(ACL)* la redondance de l'invocation de gss_import_name() et gss_compare_name() sur chaque nom de l'ACL peut être prohibitive. Comme solution de remplacement pour la prise en charge de ce cas, GSS-API définit une forme spéciale de nom de chaîne contiguë qui peut être comparée directement (par exemple, avec memcmp()). Les noms contigus convenables pour la comparaison sont générés par le sous-programme gss_export_name(), qui exige un MN en entrée. Les noms exportés peuvent être réimportés par le sous-programme gss_import_name(), et le nom interne résultant sera aussi un MN. La constante gss_OID GSS_C_NT_EXPORT_NAME identifie le type "nom d'export", et la valeur de cette constante est donnée à l'Appendice A. Structurellement, un objet Nom exporté consiste en un en-tête contenant un OID qui identifie le mécanisme qui a authentifié le nom, et un en-queue qui contient le nom lui-même, où la syntaxe de l'en-queue est définie par la spécification du mécanisme individuel. Le format précis d'un nom d'export est défini dans la spécification GSS-API indépendante du langage [RFC2743].

Noter que les résultats obtenus de l'utilisation de gss_compare_name() vont en général être différents de ceux obtenus par l'invocation de gss_canonicalize_name() et gss_export_name(), et de comparer ensuite les noms exportés. La première série d'opérations détermine si deux noms (non authentifiés) identifient le même principal ; la seconde si un mécanisme particulier les authentifierait comme le même principal. Ces deux opérations vont en général ne donner le même résultat que pour les MN.

Le type de données gss_name_t devrait être mis en œuvre comme un type de pointeur. Pour permettre au compilateur d'aider le programmeur d'application en effectuant la vérification de type, l'utilisation de (void *) est déconseillée. Un pointeur sur un type défini par la mise en œuvre est le choix préféré.

La mémorisation est allouée par des sous-programmes qui retournent des valeurs de gss_name_t. Une procédure, gss_release_name, est fournie pour libérer la mémorisation associée à un nom de forme interne.

3.11 Liens de canaux

GSS-API prend en charge l'utilisation d'étiquettes spécifiées par l'utilisateur pour identifier un certain contexte chez l'application homologue. Ces étiquettes sont destinées à être utilisées pour identifier le canal de communications particulier qui porte le contexte. Les liens de canaux sont communiqués à GSS-API en utilisant la structure suivante :

```
typedef struct      gss_channel_bindings_struct {
  OM_uint32           initiator_addrtype;
  gss_buffer_desc     initiator_address;
  OM_uint32           acceptor_addrtype;
  gss_buffer_desc     acceptor_address;
  gss_buffer_desc     application_data;
}                     *gss_channel_bindings_t;
```

Les champs initiator_addrtype et acceptor_addrtype notent le type des adresses contenues dans les mémoires tampon initiator_address et acceptor_address. Le type d'adresse devrait être un des suivants :

GSS_C_AF_UNSPEC	Type d'adresse non spécifié.
GSS_C_AF_LOCAL	Type d'adresse locale d'hôte.
GSS_C_AF_INET	Type d'adresse Internet (par exemple, IP).
GSS_C_AF_IMPLINK	Type d'adresse IMP ARPAnet.
GSS_C_AF_PUP	Type d'adresse des protocoles pup (par exemple, BSP).
GSS_C_AF_CHAOS	Type d'adresse de protocole MIT CHAOS.
GSS_C_AF_NS	Type d'adresse XEROX NS.
GSS_C_AF_NBS	Type d'adresse nbs.
GSS_C_AF_ECMA	Type d'adresse ECMA.
GSS_C_AF_DATAKIT	Type d'adresse de protocoles.
GSS_C_AF_CCITT	Protocoles du CCITT
GSS_C_AF_SNA	Type d'adresse IBM SNA.
GSS_C_AF_DECne	Type d'adresse DECnet.
GSS_C_AF_DLI	Type d'adresse d'interface de liaison de données directe.
GSS_C_AF_LAT	Type d'adresse LAT.
GSS_C_AF_HYLINK	Type d'adresse NSC Hyperchannel.
GSS_C_AF_APPLETALK	Type d'adresse AppleTalk.
GSS_C_AF_BSC	Type d'adresse 2780/3780.
GSS_C_AF_DSS	Type d'adresse de services de système réparti.
GSS_C_AF_OSI	Type d'adresse OSI TP4.
GSS_C_AF_X25	X.25.
GSS_C_AF_NULLADDR	Pas d'adresse spécifiée.

Noter que ces symboles désignent des familles d'adresses plutôt que des formats d'adressage spécifiques. Pour les familles d'adresses qui contiennent plusieurs formes d'adresses, les champs initiator_address et acceptor_address doivent contenir des informations suffisantes pour déterminer quelle forme d'adresse est utilisée. En l'absence d'autre mention, les adresses devraient être spécifiées dans l'ordre des octets du réseau (c'est-à-dire, dans l'ordre natif des octets pour la famille d'adresses).

Par conception, GSS-API enchaîne initiator_addrtype, initiator_address, acceptor_addrtype, acceptor_address et application_data pour former une chaîne d'octets. Le mécanisme calcule un MIC sur cette chaîne d'octets, et lie le MIC au jeton d'établissement de contexte émis par gss_init_sec_context. Les mêmes liens sont présentés par l'accepteur de contexte à gss_accept_sec_context, et un MIC est calculé de la même façon. Le MIC calculé est comparé à celui trouvé dans le jeton, et si les MIC diffèrent, gss_accept_sec_context va retourner une erreur GSS_S_BAD_BINDINGS, et le contexte ne sera pas établi. Certains mécanismes peuvent inclure les données de lien de canal réelles dans le jeton (plutôt que juste un MIC) ; les applications ne devraient donc pas utiliser de données confidentielles comme composants de lien de canal.

Des mécanismes individuels peuvent imposer des contraintes supplémentaires aux adresses et types d'adresse qui peuvent apparaître dans les liens de canal. Par exemple, un mécanisme peut vérifier que le champ initiator_address des liens de canal présentés à gss_init_sec_context contient l'adresse réseau correcte du système hôte. Les applications portables devraient donc s'assurer qu'elles fournissent des informations correctes pour les champs d'adresse, ou qu'elles omettent les informations d'adressage, spécifiant GSS_C_AF_NULLADDR comme le type d'adresse.

3.12 Paramètres facultatifs

Divers paramètres sont décrits comme facultatifs. Cela signifie qu'ils suivent une convention selon laquelle une valeur par défaut peut être demandée. Les conventions suivantes sont utilisées pour les paramètres omis. Ces conventions ne s'appliquent qu'aux paramètres qui sont explicitement documentés comme facultatifs.

3.12.1 Types gss_buffer_t

Ils spécifient GSS_C_NO_BUFFER comme une valeur. Pour un paramètre d'entrée, cela signifie qu'un comportement par défaut est demandé, tandis que pour un paramètre de sortie, cela indique que les informations qui seront retournées via le paramètre ne sont pas exigés par l'application.

3.12.2 Types d'entiers (entrée)

La documentation du paramètre individuel fait la liste des valeurs à utiliser pour indiquer les actions par défaut.

3.12.3 Types d'entiers (sortie)

Spécifie NUL comme la valeur pour le pointeur.

3.12.4 Types de pointeur

Spécifie NUL comme la valeur.

3.12.5 Identifiants d'objet

Spécifie GSS_C_NO_OID comme la valeur.

3.12.6 Ensembles d'identifiant d'objet

Spécifie GSS_C_NO_OID_SET comme la valeur.

3.12.7 Liens de canaux

Spécifie GSS_C_NO_CHANNEL_BINDINGS pour indiquer que les liens de canal ne sont pas à utiliser.

4. Commandes supplémentaires

Cette section expose les services facultatifs qu'un initiateur de contexte peut demander à GSS-API à l'établissement du contexte. Chacun de ces services est demandé en établissant un fanion dans le paramètre d'entrée req_flags à gss_init_sec_context.

Les services facultatifs actuellement définis sont :

Délégation – C'est le transfert (habituellement temporaire) de droits de l'initiateur à l'accepteur, ce qui permet à l'accepteur de s'authentifier comme agent de l'initiateur.

Authentification mutuelle – En plus de l'authentification de son identité par l'initiateur auprès de l'accepteur du contexte, celui-ci devrait aussi s'authentifier auprès de l'initiateur.

Détection de répétition – En plus de la fourniture des services d'intégrité de message, gss_get_mic et gss_wrap devraient inclure les informations de numérotation de message pour permettre à gss_verify_mic et gss_unwrap de détecter si un message a été dupliqué.

Détection de hors séquence – En plus de la fourniture de services d'intégrité de message, gss_get_mic et gss_wrap devraient inclure des informations de séquençage de message pour permettre à gss_verify_mic et gss_unwrap de détecter si un message a été reçu hors séquence.

Authentification anonyme – L'établissement du contexte de sécurité ne devrait pas révéler l'identité de l'initiateur à l'accepteur du contexte.

Tous les bits actuellement indéfinis dans de tels arguments de fanions devraient être ignorés par les mises en œuvre GSS-API lorsque présentés par une application, et devraient être réglés à zéro lorsque retournés à l'application par la mise en œuvre GSS-API.

Certains mécanismes peuvent ne pas prendre en charge tous les services facultatifs, et certains mécanismes peuvent ne prendre en charge que certains services en conjonction avec d'autres. gss_init_sec_context et gss_accept_sec_context informent tous deux les applications des services qui seront disponibles à partir du contexte lorsque la phase d'établissement est achevée, via le paramètre de sortie ret_flags. En général, si le mécanisme de sécurité est capable de fournir un service demandé, il devrait le faire, même si des services supplémentaires doivent être activés pour fournir le service demandé. Si le mécanisme est incapable de fournir un service demandé, il devrait continuer sans le service, laissant l'application interrompre le processus d'établissement de contexte si elle considère le service demandé comme obligatoire.

Certains mécanismes peuvent spécifier que la prise en charge de certains services est facultative, et que les mises en œuvre du mécanisme n'ont pas besoin de le fournir. Ceci est très courant pour le service de confidentialité, souvent parce qu'il y a des restrictions légales à l'utilisation du chiffrement des données, mais peut s'appliquer à tous les services. De tels mécanismes sont obligés d'envoyer au moins un jeton de l'accepteur à l'initiateur durant l'établissement de contexte quand

l'initiateur indique son désir d'utiliser un tel service, afin que la GSS-API initiatrice puisse correctement indiquer si le service est pris en charge par la GSS-API de l'accepteur.

4.1 Délégation

GSS-API permet que la délégation soit contrôlée par l'application initiatrice via un paramètre booléen à gss_init_sec_context(), le sous-programme qui établit un contexte de sécurité. Certains mécanismes n'acceptent pas la délégation, et pour ces mécanismes, les tentatives d'une application pour activer une délégation sont ignorées.

L'accepteur d'un contexte de sécurité pour lequel l'initiateur a activé la délégation va recevoir (via le paramètre delegated_cred_handle de gss_accept_sec_context) un lien d'accréditif qui contient l'identité déléguée, et ce lien d'accréditif peut être utilisé pour initier les contextes de sécurité GSS-API suivants comme un agent ou délégué de l'initiateur. Si l'identité originelle de l'initiateur est "A" et si l'identité du délégué est "B", alors, selon le mécanisme sous-jacent, l'identité incorporée par l'accréditif délégué peut être "A" ou "B agissant pour A".

Pour de nombreux mécanismes qui prennent en charge la délégation, un simple booléen ne fournit pas assez de contrôle. Des exemples des aspects supplémentaires de contrôle de délégation que pourrait fournir un mécanisme à une application sont la durée de délégation, les adresses réseau à partir desquelles la délégation est valide, et les contraintes sur les tâches qui peuvent être effectuées par le délégué. De tels contrôles sont actuellement en dehors du domaine d'application de GSS-API. Les mises en œuvre de GSS-API qui acceptent les mécanismes offrant des contrôles supplémentaires devraient fournir des sous-programmes d'extension permettant d'exécuter des contrôles (peut-être en modifiant l'accréditif GSS-API de l'initiateur avant son utilisation dans l'établissement d'un contexte). Cependant, le simple contrôle de délégation fourni par GSS-API devrait toujours être capable d'outrepasser les contrôles de délégation spécifiques des autres mécanismes ; si l'application indique à GSS_Init_sec_context() que la délégation n'est pas désirée, la mise en œuvre ne doit alors pas permettre à la délégation de se produire. Ceci est une exception à la règle générale qu'un mécanisme peut activer des services même si ils ne sont pas demandés ; la délégation ne peut être fournie qu'à la demande explicite de l'application.

4.2 Authentification mutuelle

Habituellement, un accepteur de contexte va exiger qu'un initiateur de contexte s'authentifie afin que l'accepteur puisse prendre une décision de contrôle d'accès avant d'effectuer un service pour l'initiateur. Dans certains cas, l'initiateur peut aussi demander que l'accepteur s'authentifie. GSS-API permet à l'application initiatrice de demander ce service d'authentification mutuelle en établissant un fanion lors de l'invocation de gss_init_sec_context.

L'application initiatrice est informée de ce que l'accepteur du contexte s'est ou non authentifié. Noter que certains mécanismes peuvent ne pas prendre en charge l'authentification mutuelle, et d'autres mécanismes peuvent toujours effectuer l'authentification mutuelle, que l'application initiatrice la demande ou non. En particulier, l'authentification mutuelle peut être exigée par certains mécanismes afin de mettre en œuvre la détection de répétition de message ou de message hors séquence, et pour de tels mécanismes la demande de l'un ou l'autre de ces services va automatiquement activer l'authentification mutuelle.

4.3 Détection de répétition et de hors séquence

GSS-API peut fournir la détection des message déclassés une fois qu'un contexte de sécurité a été établi. La protection peut être appliquée aux messages par l'une ou l'autre application, en invoquant soit gss_get_mic, soit gss_wrap, et vérifiée par l'application homologue en invoquant gss_verify_mic ou gss_unwrap.

gss_get_mic calcule un MIC cryptographique sur un message de l'application, et retourne ce MIC dans un jeton. L'application devrait passer le jeton et le message à l'application homologue, qui les présente à gss_verify_mic.

gss_wrap calcule un MIC cryptographique d'un message de l'application, et place le MIC et le message dans un seul jeton. L'application devrait passer le jeton à l'application homologue, qui le présente à gss_unwrap pour extraire le message et vérifier le MIC.

L'une et l'autre paire de sous-programmes peuvent être capables de détecter la livraison de message hors séquence, ou la duplication de messages. Les détails de tels messages déclassés sont indiqués par des bits d'état supplémentaires dans le code d'état majeur retourné par gss_verify_mic ou gss_unwrap. Les bits supplémentaires pertinents sont :

GSS_S_DUPLICATE_TOKEN Le jeton est un dupliqué d'un qui a déjà été reçu et traité. Seuls les contextes qui prétendent fournir la détection de répétition peuvent établir ce bit.

GSS_S_OLD_TOKEN Le jeton est trop vieux pour qu'on détermine si il est ou non un dupliqué. Les contextes qui prennent en charge la détection de hors séquence mais pas la détection de répétition devraient toujours établir ce bit si GSS_S_UNSEQ_TOKEN est établi ; les contextes qui prennent en

charge la détection de répétition ne devraient établir ce bit que si le jeton est si vieux qu'on ne peut pas vérifier si il est un dupliqué.

GSS_S_UNSEQ_TOKEN Un jeton ultérieur a déjà été traité.

GSS_S_GAP_TOKEN Un jeton antérieur n'a pas encore été reçu.

Un mécanisme n'a pas besoin de tenir une liste de tous les jetons qui ont été traités afin de prendre en charge ces codes d'état. Un mécanisme normal peut conserver les informations seulement sur les "N" plus récents jetons traités, ce qui lui permet de distinguer les jetons dupliqués et manquants parmi les "N" messages les plus récents ; la réception d'un jeton plus ancien que les "N" plus récents résulterait en un état GSS_S_OLD_TOKEN.

4.4 Authentification anonyme

Dans certaines situations, une application peut souhaiter initier le processus d'authentification pour authentifier un homologue, sans révéler sa propre identité. Par exemple, considérons une application qui fournit l'accès à une base de données contenant des informations médicales, et offrant un accès sans restriction au service. Un client d'un tel service pourrait souhaiter authentifier le service (afin d'établir la confiance dans les informations qu'il en tire) mais pourrait ne pas souhaiter que le service soit capable d'obtenir l'identité du client (peut-être à cause du souci de confidentialité des interrogations spécifiques, ou peut-être simplement pour éviter d'être inscrit sur une liste de diffusion).

En utilisation normale de GSS-API, l'identité de l'initiateur est rendue disponible à l'accepteur comme résultat du processus d'établissement du contexte. Cependant, les initiateurs de contexte peuvent demander que leur identité ne soit pas révélée à l'accepteur du contexte. De nombreux mécanismes n'acceptent pas l'authentification anonyme, et pour ces mécanismes, la demande ne sera pas honorée. Un jeton d'authentification sera quand même généré, mais l'application est toujours informée si un service demandé est indisponible, et a l'option d'interrompre l'établissement de contexte si l'anonymat est considéré comme plus précieux que les autres services de sécurité qui exigeraient l'établissement d'un contexte.

En plus d'informer l'application de l'établissement anonyme d'un contexte (via les sorties ret_flags de gss_init_sec_context et gss_accept_sec_context) le résultat facultatif src_name de gss_accept_sec_context et gss_inquire_context va, pour de tels contextes, retourner un nom de forme interne réservé, défini par la mise en œuvre.

Lorsque il est présenté à gss_display_name, ce nom de forme interne réservé va résulter en un nom imprimable qui est syntaxiquement distinguable de tout nom de principal valide pris en charge par la mise en œuvre, associé à un identifiant d'objet Type de nom avec la valeur GSS_C_NT_ANONYMOUS, dont la valeur est donnée à l'Appendice A. La forme imprimable d'un nom anonyme devrait être choisie de telle sorte qu'elle implique l'anonymat, car ce nom peut apparaître, par exemple, dans les journaux d'événements. Par exemple, la chaîne "<anonyme>" peut être un bon choix, si aucun nom imprimable valide accepté par la mise en œuvre ne peut commencer par "<" et se terminer par ">".

4.5 Confidentialité

Si un contexte prend en charge le service de confidentialité, gss_wrap peut être utilisé pour chiffrer les messages d'application. Les messages sont chiffrés de façon sélective, sous le contrôle du paramètre d'entrée conf_req_flag à gss_wrap.

4.6 Transfert de contexte inter processus

GSS-API v2 fournit des sous-programmes (gss_export_sec_context et gss_import_sec_context) qui permettent à un contexte de sécurité d'être transféré entre les processus d'une seule machine. L'utilisation la plus courante de tels dispositifs est un concept de client-serveur où le serveur est mis en œuvre comme un seul procès qui accepte les contextes de sécurité entrants, qui lance ensuite des processus fils pour traiter les données sur ces contextes. Dans un tel concept, les processus fils doivent avoir accès à la structure de données du contexte de sécurité créée au sein du parent par son invocation à gss_accept_sec_context afin qu'ils puissent utiliser les services de protection par message et supprimer le contexte de sécurité lorsque la session de communication se termine.

Comme la structure de données de contexte de sécurité est supposée contenir les informations de séquençage, il n'est en général pas praticable de partager un contexte entre des processus. Donc, GSS-API fournit un appel (gss_export_sec_context) que le procès qui possède actuellement le contexte peut invoquer pour déclarer qu'il n'a pas l'intention d'utiliser le contexte ultérieurement et de créer un jeton interprocès contenant les informations nécessaires au processus adoptant pour réussir à importer le contexte. Après la réussite de l'achèvement de gss_export_sec_context, le contexte de sécurité d'origine est rendu inaccessible au processus appelant par GSS-API, et tous les liens de contexte qui se réfèrent à ce contexte ne sont plus valides. Le processus d'origine transfère le jeton interprocès au processus adopteur, qui le passe à gss_import_sec_context, et un gss_ctx_id_t frais est créé de telle sorte qu'il soit fonctionnellement identique au contexte d'origine.

Le jeton interprocès peut contenir des données sensibles provenant du contexte de sécurité d'origine (incluant des clés de chiffrement). Les applications qui utilisent des jetons interprocès pour transférer des contextes de sécurité doivent prendre les mesures appropriées pour protéger ces jetons dans le transit.

Les mises en œuvre ne sont pas obligées de prendre en charge le transfert interprocès des contextes de sécurité. La capacité de transférer un contexte de sécurité est indiquée lorsque le contexte est créé, par gss_init_sec_context ou gss_accept_sec_context qui établissent le bit GSS_C_TRANS_FLAG dans leur paramètre ret_flags.

4.7 Utilisation de contextes incomplets

Certains mécanismes peuvent permettre d'utiliser les services par message avant l'achèvement du processus d'établissement de contexte. Par exemple, un mécanisme peut inclure des informations suffisantes dans son jeton initial de niveau contexte pour que l'accepteur de contexte décode immédiatement les messages protégés avec gss_wrap ou gss_get_mic. Pour un tel mécanisme, l'application initiatrice n'a pas besoin d'attendre jusqu'à ce que des jetons ultérieurs de niveau contexte aient été envoyés et reçus avant d'invoquer les services de protection par message.

La capacité d'un contexte à fournir les services par message avant l'achèvement de l'établissement de contexte est indiquée par l'établissement du bit GSS_C_PROT_READY_FLAG dans le paramètre ret_flags provenant de gss_init_sec_context et gss_accept_sec_context. Les applications qui souhaitent utiliser les services de protection par message sur des contextes partiellement établis devraient vérifier ce fanion avant de tenter d'invoquer gss_wrap ou gss_get_mic.

5. Descriptions des sous-programmes GSS-API

En plus des codes explicites d'état majeur documentés ici, le code GSS_S_FAILURE peut être retourné par tout sous-programme, pour indiquer une condition d'erreur spécifique de la mise en œuvre ou du mécanisme, dont les détails sont rapportés via le paramètre d'état mineur.

5.1 gss_accept_sec_context

```
OM_uint32                       gss_accept_sec_context (
OM_uint32                       *minor_status,
gss_ctx_id_t                    *context_handle,
constante gss_cred_id_t         acceptor_cred_handle,
const. gss_buffer_t             input_token_buffer,
const. gss_channel_bindings_t   input_chan_bindings,
const. gss_name_t               *src_name,
gss_OID                         *mech_type,
gss_buffer_t                    output_token,
OM_uint32                       *ret_flags,
OM_uint32                       *time_rec,
gss_cred_id_t                   *delegated_cred_handle)
```

Objet : Permet à un contexte de sécurité initié à distance d'être établi entre l'application et un homologue distant. Le sous-programme peut retourner un jeton de sortie qui devrait être transféré à l'application homologue, et l'application homologue va le présenter à gss_init_sec_context. Si aucun jeton n'a besoin d'être envoyé, gss_accept_sec_context va indiquer cela en réglant le champ de longueur de l'argument jeton de sortie à zéro. Pour achever l'établissement de contexte, un ou plusieurs jetons de réponse peuvent être requis de la part de l'application homologue ; si il en est ainsi, gss_accept_sec_context va retourner un fanion d'état de GSS_S_CONTINUE_NEEDED, auquel cas il devrait être invoqué à nouveau lorsque le jeton de réponse est reçu de l'application homologue, en passant le jeton à gss_accept_sec_context via les paramètres de jeton de sortie.

Les applications portables devraient être construites de façon à utiliser la longueur du jeton et retourner l'état pour déterminer si un jeton doit être envoyé ou si on doit en attendre un. Donc, un appelant portable typique devrait toujours invoquer gss_accept_sec_context dans une boucle :

```
    gss_ctx_id_t context_hdl = GSS_C_NO_CONTEXT;

faire {
   receive_token_from_peer(input_token);
   maj_stat = gss_accept_sec_context(&min_stat,
```

```
                        &context_hdl,
                        cred_hdl,
                        input_token,
                        input_bindings,
                        &client_name,
                        &mech_type,
                        output_token,
                        &ret_flags,
                        &time_rec,
                        &deleg_cred);
  si (GSS_ERROR(maj_stat)) {
     report_error(maj_stat, min_stat);
   };
   si (output_token->length != 0) {
     send_token_to_peer(output_token);
     gss_release_buffer(&min_stat, output_token);
   };
   si (GSS_ERROR(maj_stat)) {
     si (context_hdl != GSS_C_NO_CONTEXT)
       gss_delete_sec_context(&min_stat,
                        &context_hdl,
                        GSS_C_NO_BUFFER);
     break;
   };
  } while (maj_stat & GSS_S_CONTINUE_NEEDED);
```

Chaque fois que le sous-programme retourne un état majeur qui comporte la valeur GSS_S_CONTINUE_NEEDED, le contexte n'est pas complètement établi et les restrictions suivantes s'appliquent aux paramètres de sortie :

La valeur retournée via le paramètre time_rec est indéfinie sauf si le paramètre ret_flags qui l'accompagne contient le bit GSS_C_PROT_READY_FLAG, indiquant que les services par message peuvent être appliqués avant un état d'achèvement réussi, la valeur retournée via le paramètre mech_type peut être indéfinie jusqu'à ce que le sous-programme retourne une valeur d'état majeur de GSS_S_COMPLETE.

Les valeurs retournées des bits GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG,GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG et GSS_C_ANON_FLAG via le paramètre ret_flags devraient contenir les valeurs dont la mise en œuvre s'attend qu'elles soient valides si l'établissement de contexte devait réussir.

Les valeurs des bits GSS_C_PROT_READY_FLAG et GSS_C_TRANS_FLAG au sein de ret_flags devraient indiquer l'état réel au moment où gss_accept_sec_context retourne, que le contexte soit ou non pleinement établi.

Bien que cela exige que les mises en œuvre GSS-API établissent le GSS_C_PROT_READY_FLAG dans le ret_flags final retourné à un appelant (c'est-à-dire, lorsque il est accompagné par un code d'état GSS_S_COMPLETE) les applications ne devraient pas compter sur ce comportement car le fanion n'a pas été défini dans la version 1 de GSS-API. Les applications devraient plutôt être prêtes à utiliser les services par message après un établissement de contexte réussi, conformément aux valeurs de GSS_C_INTEG_FLAG et GSS_C_CONF_FLAG.

Tous les autres bits au sein de l'argument ret_flags devraient être réglés à zéro. Alors que le sous-programme retourne GSS_S_CONTINUE_NEEDED, les valeurs retournées via l'argument ret_flags indiquent les services que la mise en œuvre s'attend à trouver disponibles à partir du contexte établi.

Si l'invocation initiale de gss_accept_sec_context() échoue, la mise en œuvre ne devrait pas créer un objet Contexte, et devrait laisser la valeur du paramètre context_handle réglée à GSS_C_NO_CONTEXT pour l'indiquer. Dans le cas d'un échec sur un appel suivant, il est permis à la mise en œuvre de supprimer le contexte de sécurité "à moitié construit" (auquel cas elle devrait régler le paramètre context_handle à GSS_C_NO_CONTEXT) mais le comportement préféré est de laisser inchangé le contexte de sécurité (et le paramètre context_handle) pour que l'application le supprime (en utilisant gss_delete_sec_context).

Durant l'établissement de contexte, les bits d'état pour information GSS_S_OLD_TOKEN et GSS_S_DUPLICATE_TOKEN indiquent des erreurs fatales, et les mécanismes GSS-API devraient toujours les retourner en association avec une erreur de sous-programme de GSS_S_FAILURE. Cette exigence d'appariement n'existait pas dans la version 1 de la spécification GSS-API, de sorte que les applications qui souhaitent fonctionner sous la version 1 doivent

faire un cas particulier de ces codes.

Paramètres :

context_handle gss_ctx_id_t, lire/modifier. Lien de contexte pour nouveau contexte. Fournit GSS_C_NO_CONTEXT pour le premier appel ; utilise la valeur retournée dans les appels suivants. Une fois que gss_accept_sec_context() a retourné une valeur via ce paramètre, les ressources ont été allouées au contexte correspondant, et doivent être libérées par l'application après utilisation avec un appel à gss_delete_sec_context().

acceptor_cred_handle gss_cred_id_t, lecture. Lien d'accréditif revendiqué par l'accepteur de contexte. Spécifie GSS_C_NO_CREDENTIAL pour accepter le contexte comme principal par défaut. Si GSS_C_NO_CREDENTIAL est spécifié, mais qu'aucun principal d'accepteur par défaut n'est défini, GSS_S_NO_CRED sera retourné.

input_token_buffer mémoire tampon, opaque. Jeton en lecture obtenu de l'application distante.

input_chan_bindings lien de canal, lecture, facultatif. Liens spécifiés par l'application. Permet à l'application de lier en toute sécurité les informations d'identification de canal au contexte de sécurité. Si les liens de canal ne sont pas utilisés, spécifier GSS_C_NO_CHANNEL_BINDINGS.

src_name gss_name_t, modifier, facultatif. Nom authentifié de l'initiateur de contexte. Après utilisation, ce nom devrait être désalloué en le passant à gss_release_name(). Si non exigé, spécifier NUL.

mech_type Identifiant d'objet, modifier, facultatif. Mécanisme de sécurité utilisé. La valeur d'OID retournée sera un pointeur sur une mémorisation statique, et devrait être traité comme en lecture seule par l'appelant (en particulier, il n'a pas besoin d'être libéré). Si non exigé, spécifier NUL.

output_token mémoire tampon, opaque, modifier. Jeton à passer à l'application homologue. Si le champ de longueur du jeton en mémoire tampon retourné est 0, aucun jeton n'a alors besoin d'être passé à l'application homologue. Si un champ de longueur non zéro est retourné, la mémorisation associée doit être libérée après utilisation par l'application avec un appel à gss_release_buffer().

ret_flags gabarit binaire, modifier, facultatif ; Contient divers fanions indépendants, dont chacun indique que le contexte accepte une option de service spécifique. Si non nécessaire, spécifier NUL. Les noms symboliques sont fournis pour chaque fanion, et les noms symboliques correspondant aux fanions exigés devraient être traités avec l'opérateur logique ET avec la valeur de ret_flags pour vérifier si une certaine option est acceptée par le contexte. Les fanions sont :

GSS_C_DELEG_FLAG
Vrai – Les accréditifs délégués sont disponibles via le paramètre delegated_cred_handle.
Faux - Aucun accréditif n'a été délégué.

GSS_C_MUTUAL_FLAG
Vrai – L'homologue distant a demandé l'authentification mutuelle.
Faux - L'homologue distant n'a pas demandé l'authentification mutuelle.

GSS_C_REPLAY_FLAG
Vrai – La répétition du message protégé sera détectée.
Faux – Les messages répétés ne seront pas détectés.

GSS_C_SEQUENCE_FLAG
Vrai – Les messages protégés hors séquence seront détectés.
Faux - Les messages protégés hors séquence ne seront pas détectés.

GSS_C_CONF_FLAG
Vrai – Le service de confidentialité peut être invoqué en appelant le sous-programme gss_wrap.
Faux – Pas de service de confidentialité (via gss_wrap) disponible. gss_wrap fournira seulement les services d'encapsulation de message, d'authentification d'origine et d'intégrité des données.

GSS_C_INTEG_FLAG
Vrai – Le service d'intégrité peut être invoqué en appelant les sous-programmes gss_get_mic ou gss_wrap.
Faux – Le service d'intégrité par message est indisponible.

GSS_C_ANON_FLAG
Vrai – L'initiateur ne souhaite pas être authentifié ; le paramètre src_name (si demandé) contient un nom interne anonyme.
Faux – L'initiateur a été authentifié normalement.

GSS_C_PROT_READY_FLAG
Vrai – Les services de protection (comme spécifié par les états de GSS_C_CONF_FLAG et GSS_C_INTEG_FLAG) sont disponibles si la valeur d'état majeur qui les accompagne en retour est soit GSS_S_COMPLETE soit GSS_S_CONTINUE_NEEDED.
Faux – Les services de protection (comme spécifié par les états de GSS_C_CONF_FLAG et GSS_C_INTEG_FLAG) ne sont disponibles que si la valeur d'état majeur qui les accompagne en retour est GSS_S_COMPLETE.
GSS_C_TRANS_FLAG
Vrai – Le contexte de sécurité résultant peut être transféré aux autres processus via un appel à gss_export_sec_context().
Faux – Le contexte de sécurité n'est pas transférable. Tous les autres bits devraient être réglés à zéro.

time_rec Entier, modifier, facultatif ;. Nombre de secondes pendant lesquelles le contexte va rester valide. Spécifier NUL si il n'est pas exigé.

delegated_cred_handle gss_cred_id_t, modifier, facultatif. Lien d'accréditif pour les accréditifs reçus de l'initiateur de contexte. N'est valide que si deleg_flag est vrai dans ret_flags, auquel cas un lien d'accréditif explicite (c'est-à-dire, non GSS_C_NO_CREDENTIAL) sera retourné ; si deleg_flag est faux, gss_accept_context() va régler ce paramètre à GSS_C_NO_CREDENTIAL. Si un lien d'accréditif est retourné, les ressources associées doivent être libérées par l'application après utilisation avec l'invocation de gss_release_cred(). Spécifier NUL si non exigé.

minor_status Entier, modifier ; code d'état spécifique du mécanisme :
GSS_S_CONTINUE_NEEDED Indique qu'un jeton provenant de l'application homologue est requis pour achever le contexte, et que gss_accept_sec_context doit être invoqué à nouveau avec ce jeton.
GSS_S_DEFECTIVE_TOKEN Indique que les vérifications de cohérence effectuées sur le jeton d'entrée ont échoué.
GSS_S_DEFECTIVE_CREDENTIAL Indique que les vérifications de cohérence effectuées sur l'accréditif ont échoué.
GSS_S_NO_CRED Les accréditifs fournis n'étaient pas valides pour l'acceptation du contexte, ou le lien d'accréditif ne faisait référence à aucun accréditif.
GSS_S_CREDENTIALS_EXPIRED Les accréditifs référencés sont périmés.
GSS_S_BAD_BINDINGS Le jeton d'entrée contient des liens de canal différents de ceux spécifiés via le paramètre input_chan_bindings.
GSS_S_NO_CONTEXT Indique que le lien de contexte fourni ne se référait pas à un contexte valide.
GSS_S_BAD_SIG Le jeton d'entrée contient un MIC invalide.
GSS_S_OLD_TOKEN Le jeton d'entrée était trop vieux C'est une erreur fatale durant l'établissement de contexte.
GSS_S_DUPLICATE_TOKEN Le jeton d'entrée est valide, mais c'est un duplicata d'un jeton déjà traité. C'est une erreur fatale durant l'établissement de contexte.
GSS_S_BAD_MECH Le jeton reçu spécifiait un mécanisme qui n'est pas accepté par la mise en œuvre ou l'accréditif fourni.

5.2 gss_acquire_cred

```
OM_uint32        gss_acquire_cred (
  OM_uint32            *minor_status,
  const gss_name_t     desired_name,
  OM_uint32            time_req,
  const gss_OID_set    desired_mechs,
  gss_cred_usage_t     cred_usage,
  gss_cred_id_t        *output_cred_handle,
  gss_OID_set          *actual_mechs,
  OM_uint32            *time_rec)
```

Objet : Permet à une application d'acquérir un lien par nom avec un accréditif préexistant. Les mises en œuvre GSS-API doivent imposer une politique locale de contrôle d'accès aux appelants de ce sous-programme pour empêcher des appelants non autorisés d'acquérir des accréditifs auxquels ils n'ont pas droit. Ce sous-programme n'est pas destiné à fournir une fonction de "connexion au réseau", à ce titre, une fonction impliquerait la création de nouveaux accréditifs plutôt que de simplement acquérir un lien sur des accréditifs existants. De telles fonctions, si elles sont requises, devraient être définies dans des extensions spécifiques de la mise en œuvre à l'API.

Si desired_name est GSS_C_NO_NAME, l'appel est interprété comme demande de lien d'accréditif qui va invoquer le comportement par défaut lorsque passé à gss_init_sec_context() (si cred_usage est GSS_C_INITIATE ou GSS_C_BOTH) ou à gss_accept_sec_context() (si cred_usage est GSS_C_ACCEPT ou GSS_C_BOTH).

Les mécanismes devraient honorer le paramètre desired_mechs, et retourner un accréditif dont l'utilisation ne convienne qu'avec le mécanisme demandé. Une exception à cela est le cas où un élément d'accréditif sous-jacent peut être partagé par plusieurs mécanismes ; dans ce cas, il est permis à une mise en œuvre d'indiquer tous les mécanismes avec lesquels l'élément d'accréditif peut être utilisé. Si desired_mechs est un ensemble vide, le comportement est indéfini.

Ce sous-programme est censé être utilisé principalement par les accepteurs de contexte, car les mises en œuvre vont vraisemblablement fournir des façons spécifiques du mécanisme pour obtenir des accréditifs d'initiateur GSS-API du processus de connexion du système. Certaines mises en œuvre peuvent donc ne pas prendre en charge l'acquisition des accréditifs GSS_C_INITIATE ou GSS_C_BOTH via gss_acquire_cred pour tout nom autre que GSS_C_NO_NAME, ou un nom produit en appliquant soit gss_inquire_cred à un accréditif valide, soit gss_inquire_context à un contexte actif.

Si l'acquisition d'accréditif consomme trop de temps pour un mécanisme, celui-ci peut choisir de retarder l'acquisition réelle jusqu'à ce que l'accréditif soit exigé (par exemple, par gss_init_sec_context ou gss_accept_sec_context). De telles décisions de mise en œuvre spécifiques du mécanisme devraient être invisibles à l'application appelante ; donc, une invocation de gss_inquire_cred suivant immédiatement l'invocation de gss_acquire_cred doit retourner des données d'accréditif valides, et peut donc subir la redondance d'une acquisition différée d'accréditif.

Paramètres :

desired_name gss_name_t, lecture.
Nom du principal dont l'accréditif devrait être acquis.

time_req Entier, lecture, facultatif.
Nombre de secondes pendant lequel les accréditifs devraient rester valides. Spécifie GSS_C_INDEFINITE pour demander que les accréditifs aient la durée de vie maximum permise.

desired_mechs Ensemble d'identifiants d'objet, lecture, facultatif.
Ensemble de mécanismes de sécurité sous-jacents qui peuvent être utilisés. GSS_C_NO_OID_SET peut être utilisé pour obtenir une valeur par défaut spécifique de la mise en œuvre.

cred_usage gss_cred_usage_t, lecture.
GSS_C_BOTH – Les accréditifs peuvent être utilisés pour initier ou accepter les contextes de sécurité.
GSS_C_INITIATE – Les accréditifs seront seulement utilisés pour initier les contextes de sécurité.
GSS_C_ACCEPT – Les accréditifs seront seulement utilisés pour accepter les contextes de sécurité.

output_cred_handle gss_cred_id_t, modifier.
Le lien d'accréditif retourné. Les ressources associées à ce lien d'accréditif doivent être libérées par l'application après usage avec une invocation de gss_release_cred().

actual_mechs Ensemble d'identifiants d'objet, modifier, facultatif.
L'ensemble de mécanismes pour lesquels l'accréditif est valide. Les mémorisations associées à l'ensemble d'OID retournés doit être libéré par l'application après usage avec une invocation de gss_release_oid_set(). Spécifier NUL si non exigé.

time_rec Entier, modifier, facultatif.
Nombre réel de secondes pendant lequel les accréditifs retournés vont rester valides. Si la mise en œuvre n'accepte pas l'expiration d'accréditifs, la valeur GSS_C_INDEFINITE sera retournée. Spécifier NUL si non exigé.

minor_status Entier, modifier.
Code d'état spécifique du mécanisme.

Valeur de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.
GSS_S_BAD_MECH	Un mécanisme indisponible a été demandé.
GSS_S_BAD_NAMETYPE	Le type contenu dans le paramètre desired_name n'est pas accepté.
GSS_S_BAD_NAME	La valeur fournie pour le paramètre desired_name est mal formée.
GSS_S_CREDENTIALS_EXPIRED	Les accréditifs n'ont pas pu être acquis parce qu'ils sont périmés.
GSS_S_NO_CRED	Aucun accréditif n'a été trouvé pour le nom spécifié.

5.3 gss_add_cred

```
OM_uint32      gss_add_cred (
  OM_uint32              *minor_status,
  const gss_cred_id_t    input_cred_handle,
  const gss_name_t       desired_name,
  const gss_OID          desired_mech,
  gss_cred_usage_t       cred_usage,
  OM_uint32              initiator_time_req,
  OM_uint32              acceptor_time_req,
  gss_cred_id_t          *output_cred_handle,
  gss_OID_set            *actual_mechs,
  OM_uint32              *initiator_time_rec,
  OM_uint32              *acceptor_time_rec)
```

Objet : Ajoute un élément d'accréditif à un accréditif. L'élément d'accréditif est identifié par le nom du principal auquel il se réfère. Les mises en œuvre GSS-API doivent imposer une politique locale de contrôle d'accès aux appelants de ce sous-programme pour empêcher des appelants non autorisés d'acquérir des éléments d'accréditif auxquels ils n'ont pas droit. Ce sous-programme n'est pas destiné à fournir une fonction de "connexion au réseau", car une telle fonction impliquerait la création de nouvelles données d'authentification spécifiques du mécanisme, plutôt que de simplement acquérir un lien GSS-API aux données existantes. De telles fonctions, si elles sont requises, devraient être définies dans des extensions spécifiques de la mise en œuvre à l'API.

Si le desired_name est GSS_C_NO_NAME, l'appel est interprété comme demande d'ajout d'un élément d'accréditif qui va invoquer le comportement par défaut lorsque il est passé à gss_init_sec_context() (si cred_usage est GSS_C_INITIATE ou GSS_C_BOTH) ou gss_accept_sec_context() (si cred_usage est GSS_C_ACCEPT ou GSS_C_BOTH).

Ce sous-programme est supposé être utilisé principalement par les accepteurs de contexte, car les mises en œuvre vont probablement fournir des façons spécifiques des mécanismes pour obtenir les accréditifs d'initiateur GSS-API du processus de connexion du système. Certaines mises en œuvre peuvent donc ne pas prendre en charge l'acquisition des accréditifs GSS_C_INITIATE ou GSS_C_BOTH via gss_acquire_cred pour tout nom autre que GSS_C_NO_NAME, ou un nom produit en appliquant soit gss_inquire_cred à un accréditif valide, soit gss_inquire_context à un contexte actif.

Si l'acquisition d'accréditif est une grosse consommation de temps pour un mécanisme, celui-ci peut choisir de retarder l'acquisition réelle jusqu'à ce que l'accréditif soit exigé (par exemple, par gss_init_sec_context ou gss_accept_sec_context). De telles décisions spécifiques du mécanisme devraient être invisibles à l'application appelante ; donc, une invocation de gss_inquire_cred qui suit immédiatement l'appel à gss_add_cred doit retourner des données d'accréditif valides, et peut donc subir la redondance d'une acquisition d'accréditif différée.

Ce sous-programme peut être utilisé pour composer un nouvel accréditif contenant tous les éléments d'accréditif de l'original en plus de l'élément d'accréditif nouvellement acquis, ou pour ajouter le nouvel élément d'accréditif à un accréditif existant. Si NUL est spécifié pour l'argument de paramètre de sortie output_cred_handle, le nouvel élément d'accréditif sera ajouté à l'accréditif identifié par input_cred_handle ; si un pointeur valide est spécifié pour le paramètre output_cred_handle, un nouveau lien d'accréditif sera créé.

Si GSS_C_NO_CREDENTIAL est spécifié comme input_cred_handle, gss_add_cred va composer un accréditif (et régler en conséquence le paramètre output_cred_handle) sur la base du comportement par défaut. C'est à dire que l'appel aura le même effet que si l'application avait d'abord fait un appel à gss_acquire_cred(), spécifiant le même usage et passant GSS_C_NO_NAME comme paramètre desired_name pour obtenir un lien d'accréditif explicite incorporant le comportement par défaut, passé ce lien d'accréditif à gss_add_cred(), et finalement appelé gss_release_cred() sur le premier lien d'accréditif.

Si GSS_C_NO_CREDENTIAL est spécifié comme paramètre input_cred_handle, un output_cred_handle non NUL doit être fourni.

Paramètres :

minor_status Entier, modifier.
Code d'état spécifique du mécanisme.

input_cred_handle gss_cred_id_t, lecture, facultatif.
Accréditif auquel un élément d'accréditif sera ajouté. Si GSS_C_NO_CREDENTIAL est spécifié, le

sous-programme va composer le nouvel accréditif sur la base du comportement par défaut (voir la description ci-dessus). Noter qu'alors que le lien d'accréditif n'est pas modifié par gss_add_cred(), l'accréditif sous-jacent sera modifié si output_credential_handle est NUL.

desired_name gss_name_t, lecture.
Nom du principal dont l'accréditif devrait être acquis.

desired_mech Identifiant d'objet, lecture.
Mécanisme de sécurité sous-jacent avec lequel l'accréditif peut être utilisé.

cred_usage gss_cred_usage_t, lecture.
GSS_C_BOTH – L'accréditif peut être utilisé pour initier ou accepter les contextes de sécurité.
GSS_C_INITIATE – L'accréditif ne sera utilisé que pour initier les contextes de sécurité.
GSS_C_ACCEPT – L'accréditif ne sera utilisé que pour accepter les contextes de sécurité.

initiateur_time_req Entier, lecture, facultatif
Nombre de secondes pendant lequel l'accréditif devrait rester valide pour initier les contextes de sécurité. Cet argument est ignoré si les accréditifs composés sont du type GSS_C_ACCEPT. Spécifier GSS_C_INDEFINITE pour demander que les accréditifs aient la durée de vie maximum permise par l'initiateur.

acceptor_time_req Entier, lecture, facultatif
Nombre de secondes pendant lequel l'accréditif devrait rester valide pour accepter les contextes de sécurité. Cet argument est ignoré si les accréditifs composés sont du type GSS_C_INITIATE. Spécifie GSS_C_INDEFINITE pour demander que les accréditifs aient la durée de vie maximum permise par l'initiateur.

output_cred_handle gss_cred_id_t, modifier, facultatif.
Le lien d'accréditif retourné, qui contient le nouvel élément d'accréditif et tous les éléments d'accréditif provenant de input_cred_handle. Si un pointeur valide sur un gss_cred_id_t est fourni pour ce paramètre, gss_add_cred crée un nouveau lien d'accréditif contenant tous les éléments d'accréditif provenant de input_cred_handle et l'élément d'accréditif nouvellement acquis ; si NUL est spécifié pour ce paramètre, l'élément d'accréditif nouvellement acquis sera ajouté à l'accréditif identifié par input_cred_handle.
Les ressources associées à tout lien d'accréditif retourné via ce paramètre doivent être libérées par l'application après usage avec un appel à gss_release_cred().

actual_mechs Ensemble d'identifiants d'objet, modifier, facultatif.
Ensemble complet des mécanismes pour lesquels le nouvel accréditif est valide. La mémorisation de l'ensemble d'OID retournés doit être libéré par l'application après usage avec un appel à gss_release_oid_set(). Spécifier NUL si non exigé.

initiator_time_rec Entier, modifier facultatif.
Nombre réel de secondes pendant lequel les accréditifs retournés vont rester valides pour les contextes initiateurs utilisant le mécanisme spécifié. Si la mise en œuvre ou le mécanisme n'accepte pas l'expiration des accréditifs, la valeur GSS_C_INDEFINITE sera retournée. Spécifier NUL si non exigé.

acceptor_time_rec Entier, modifier, facultatif.
Nombre réel de secondes pendant lequel les accréditifs retournés vont rester valides pour les contextes accepteurs utilisant le mécanisme spécifié. Si la mise en œuvre ou le mécanisme n'accepte pas l'expiration des accréditifs, la valeur GSS_C_INDEFINITE sera retournée. Spécifier NUL si non exigé.

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.
GSS_S_BAD_MECH	Mécanisme demandé indisponible.
GSS_S_BAD_NAMETYPE	Le type contenu dans le paramètre desired_name n'est pas accepté.
GSS_S_BAD_NAME	La valeur fournie pour le paramètre desired_name est mal formée.
GSS_S_DUPLICATE_ELEMENT	L'accréditif contient déjà un élément pour le mécanisme demandé avec une période d'usage et de validité qui le couvre.
GSS_S_CREDENTIALS_EXPIRED	Les accréditifs exigés n'ont pas pu être ajoutés parce qu'ils sont périmés.
GSS_S_NO_CRED	Aucun accréditif n'a été trouvé pour le nom spécifié.

5.4 gss_add_oid_set_member

```
OM_uint32      gss_add_oid_set_member (
  OM_uint32       *minor_status,
  const gss_OID   member_oid,
  gss_OID_set     *oid_set)
```

Objet : Ajoute un identifiant d'objet à un ensemble d'identifiants d'objet. Ce sous-programme est destiné à être utilisé en conjonction avec gss_create_empty_oid_set lors de la construction d'un ensemble d'OID de mécanisme à entrer dans gss_acquire_cred. Le paramètre oid_set doit se référer à un ensemble d'OID qui a été créé par GSS-API (par exemple, un ensemble retourné par gss_create_empty_oid_set()). GSS-API crée une copie du member_oid et l'insère dans l'ensemble, en agrandissant si nécessaire l'espace de mémorisation alloué à la matrice des éléments de l'ensemble d'OID. Le sous-programme peut ajouter l'OID du nouveau membre à tout endroit de la matrice des éléments, et les mises en œuvre devraient vérifier que le nouveau member_oid n'est pas déjà contenu dans la matrice des éléments ; si le member_oid est déjà présent, le oid_set devrait rester inchangé.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

member_oid Identifiant d'objet, lecture. L'identifiant d'objet à copier dans l'ensemble.

oid_set Ensemble d'identifiants d'objet, modifier. Ensemble dans lequel l'identifiant d'objet devrait être inséré.

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.

5.5 gss_canonicalize_name

```
OM_uint32      gss_canonicalize_name (
  OM_uint32              *minor_status,
  constante gss_name_t   input_name,
  constante gss_OID      mech_type,
  gss_name_t             *output_name)
```

Objet : Génère un nom canonique de mécanisme (MN) à partir d'un nom interne arbitraire. Le nom de mécanisme est le nom qui serait retourné à un accepteur de contexte suite à une authentification réussie d'un contexte où l'initiateur a utilisé le input_name dans une invocation réussie de gss_acquire_cred, spécifiant un ensemble d'OID contenant <mech_type> comme seul membre, suivi par une invocation de gss_init_sec_context, spécifiant <mech_type> comme mécanisme d'authentification.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

input_name gss_name_t, lecture. Nom pour lequel une forme canonique est désirée.

mech_type Identifiant d'objet, lecture.
Mécanisme d'authentification pour lequel la forme canonique du nom est désirée. Le mécanisme désiré doit être spécifié explicitement ; aucune valeur par défaut n'est fournie.

output_name gss_name_t, modifier.
Nom canonique résultant. La mémorisation associée à ce nom doit être libérée par l'application après usage avec un appel à gss_release_name().

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.
GSS_S_BAD_MECH Le mécanisme identifié n'est pas accepté.
GSS_S_BAD_NAMETYPE Le nom interne fourni ne contient pas d'élément qui puisse être traité par le mécanisme

spécifié.
GSS_S_BAD_NAME Le nom interne fourni était mal formé.

5.6 gss_compare_name

```
OM_uint32        gss_compare_name (
  OM_uint32                *minor_status,
  constante gss_name_t     name1,
  constante gss_name_t     name2,
  int                      *name_equal)
```

Objet : Permet à une application de comparer deux noms de forme interne pour déterminer si ils se réfèrent à la même entité. Si l'un ou l'autre nom présenté à gss_compare_name note un principal anonyme, le sous-programme devrait indiquer que les deux noms ne se réfèrent pas à la même identité.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

name1 gss_name_t, lecture. Nom de forme interne.

name2 gss_name_t, lecture. Nom de forme interne.

name_equal booléen, modifier.
non zéro – les noms se réfèrent à la même entité.
zéro – les noms se réfèrent à des entités différentes (strictement, il n'est pas connu que les noms se réfèrent à la même identité).

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.
GSS_S_BAD_NAMETYPE Les deux noms sont de types non comparables.
GSS_S_BAD_NAME name1 et/ou name2 était mal formé.

5.7 gss_context_time

```
OM_uint32        gss_context_time (
  OM_uint32              *minor_status,
  const gss_ctx_id_t     context_handle,
  OM_uint32              *time_rec)
```

Objet : Détermine le nombre de secondes pendant lequel le contexte spécifié va rester valide.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique de la mise en œuvre.

context_handle gss_ctx_id_t, lecture. Identifie le contexte à interroger.

time_rec Entier, modifier. Nombre de secondes pendant lequel le contexte va rester valide. Si le contexte est déjà périmé, zéro sera retourné.

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.
GSS_S_CONTEXT_EXPIRED Le contexte est déjà périmé.
GSS_S_NO_CONTEXT Le paramètre context_handle n'identifie pas un contexte valide.

5.8 gss_create_empty_oid_set

```
OM_uint32      gss_create_empty_oid_set (
```

```
  OM_uint32       *minor_status,
  gss_OID_set     *oid_set)
```

Objet : Crée un ensemble d'identifiants d'objet qui ne contient pas d'identifiant d'objet, auquel les membres peuvent être ensuite ajoutés en utilisant le sous-programme gss_add_oid_set_member(). Ces sous-programmes sont destinés à être utilisés pour construire des ensembles d'identifiants d'objet de mécanisme, comme entrées à gss_acquire_cred.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique de mécanisme.

oid_set Ensemble d'identifiants d'objet, modifier.
L'ensemble d'identifiants d'objet vide. Le sous-programme va allouer l'objet gss_OID_set_desc, que l'application doit libérer après utilisation avec un appel à gss_release_oid_set().

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.

5.9 gss_delete_sec_context

```
OM_uint32       gss_delete_sec_context (
  OM_uint32       *minor_status,
  gss_ctx_id_t    *context_handle,
  gss_buffer_t    output_token)
```

Objet : Supprime un contexte de sécurité. gss_delete_sec_context va supprimer les structures de données locales associées au contexte de sécurité spécifié, et peut générer un jeton de sortie, qui lorsque il est passé au gss_process_context_token homologue va lui indiquer de faire la même chose. Si aucun jeton n'est requis par le mécanisme, la GSS-API devrait régler le champ Longueur du jeton de sortie (si il en est fourni un) à zéro. Aucun autre service de sécurité ne peut être obtenu en utilisant le contexte spécifié par context_handle.

En plus de supprimer les contextes de sécurité établis, gss_delete_sec_context doit aussi être capable de supprimer les contextes de sécurité "à demi construits" résultant d'une séquence incomplète d'invocations de gss_init_sec_context() ou gss_accept_sec_context().

Le paramètre output_token est conservé pour la compatibilité avec la version 1 de GSS-API. Il est recommandé que les deux applications homologues invoquent gss_delete_sec_context en passant la valeur GSS_C_NO_BUFFER pour le paramètre output_token, ce qui indique qu'aucun jeton n'est exigé, et que gss_delete_sec_context devrait simplement supprimer les structures de données de contexte locales. Si l'application passe bien une mémoire tampon valide à gss_delete_sec_context, les mécanismes sont invités à retourner un jeton de longueur zéro, ce qui indique qu'aucune action de l'homologue n'est nécessaire, et qu'aucun jeton ne devrait être transféré par l'application.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

context_handle gss_ctx_id_t, modifier. Lien de contexte identifiant le contexte à supprimer. Après suppression du contexte, la GSS-API va régler ce lien de contexte à GSS_C_NO_CONTEXT.

output_token mémoire tampon, opaque, modifier, facultatif.
Jeton à envoyer à l'application distante pour lui dire de supprimer aussi le contexte. Il est recommandé que les applications spécifient GSS_C_NO_BUFFER pour ce paramètre, demandant seulement la suppression locale. Si un paramètre mémoire tampon est fourni par l'application, le mécanisme peut y insérer un jeton en retour ; les mécanismes qui mettent en œuvre seulement la suppression locale devraient régler le champ de longueur de ce jeton à zéro pour indiquer à l'application qu'aucun jeton n'est à envoyer à l'homologue.

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.
GSS_S_NO_CONTEXT Aucun contexte valide n'a été fourni.

5.10 gss_display_name

```
OM_uint32      gss_display_name (
  OM_uint32          *minor_status,
  const gss_name_t   input_name,
  gss_buffer_t       output_name_buffer,
  gss_OID            *output_name_type)
```

Objet : Permet à une application d'obtenir une représentation textuelle d'un nom de forme interne opaque pour les besoins de l'affichage. La syntaxe d'un nom imprimable est définie par la mise en œuvre GSS-API.

Si le nom d'entrée note un principal anonyme, la mise en œuvre devrait retourner la valeur gss_OID de GSS_C_NT_ANONYMOUS comme output_name_type, et un nom textuel qui soit syntaxiquement distinct de tous les noms imprimables valides acceptés dans output_name_buffer.

Si input_name a été créé par un appel à gss_import_name, spécifiant GSS_C_NO_OID comme type de nom, les mises en œuvre qui emploient une conversion faible entre les types de noms peuvent retourner GSS_C_NO_OID via le paramètre output_name_type.

Paramètres :

minor_status	Entier, modifier. Code d'état spécifique du mécanisme.
input_name	gss_name_t, lecture. Nom à afficher.
output_name_buffer	Mémoire tampon, chaîne de caractères, modifier. Mémoire tampon pour recevoir la chaîne de nom textuelle. L'application doit libérer la mémorisation associée à ce nom après usage avec un appel à gss_release_buffer().
output_name_type	Identifiant d'objet, modifier, facultatif. Type du nom retourné. Le gss_OID retourné sera un pointeur sur la mémorisation statique, et devrait être traité en lecture seule par l'appelant (en particulier, l'application ne devrait pas tenter de la libérer). Spécifier NUL si non exigé.

Valeurs de la fonction :	Code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.
GSS_S_BAD_NAME	le nom d'entrée était mal formé.

5.11 gss_display_status

```
OM_uint32        gss_display_status (
  OM_uint32          *minor_status,
  OM_uint32          status_value,
  int                status_type,
  constante gss_OID  mech_type,
  OM_uint32          *message_context,
  gss_buffer_t       status_string)
```

Objet : Permet à une application d'obtenir une représentation textuelle d'un code d'état GSS-API, pour affichage à l'utilisateur ou pour les besoins de la connexion. Comme certaines valeurs d'état peuvent indiquer plusieurs conditions, les applications peuvent avoir besoin d'invoquer plusieurs fois gss_display_status, chaque invocation générant une seule chaîne de texte. Le paramètre message_context est utilisé par gss_display_status pour mémoriser les informations d'état au sujet desquelles les messages d'erreur ont déjà été extraits d'une certaine valeur d'état ; message_context doit être initialisé à 0 par l'application avant la première invocation, et gss_display_status va retourner une valeur non zéro dans ce paramètre si il y a d'autres messages à extraire.

Le paramètre message_context contient toutes les informations d'état requises par gss_display_status afin d'extraire d'autres messages de la valeur d'état ; même lorsque une valeur non zéro est retournée dans ce paramètre, l'application n'est pas obligée d'invoquer à nouveau gss_display_status sauf si d'autres messages sont désirés. Le code suivant extrait tous les messages d'un certain code d'état et les imprime à stderr :

```
OM_uint32          message_context;
```

```
        OM_uint32          status_code;
        OM_uint32          maj_status;
        OM_uint32          min_status;
        gss_buffer_desc    status_string;
                ...

message_context = 0 ;

  faire {

    maj_status = gss_display_status (
                &min_status,
                status_code,
                GSS_C_GSS_CODE,
                GSS_C_NO_OID,
                &message_context,
                &status_string)

    fprintf(stderr,
          "%.*s\n",
          (int)status_string.length,
          (char *)status_string.value);
    gss_release_buffer(&min_status, &status_string);
} alors que (message_context != 0);
```

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

status_value Entier, lecture. Valeur d'état à convertir.

status_type Entier, lecture.
GSS_C_GSS_CODE – status_value est un code d'état GSS.
GSS_C_MECH_CODE – status_value est un code d'état de mécanisme.

mech_type Identifiant d'objet, lecture, facultatif. Mécanisme sous-jacent (utilisé pour interpréter une valeur d'état mineur). Fournit GSS_C_NO_OID pour obtenir le système par défaut.

message_context Entier, lecture/modifier. Devrait être initialisé à zéro par l'application avant la première invocation. Au retour de gss_display_status(), un paramètre non zéro de status_value indique que des messages supplémentaires peuvent être extraits du code d'état via des invocations suivantes de gss_display_status(), passant les mêmes paramètres status_value, status_type, mech_type, et message_context.

status_string Mémoire tampon, chaîne de caractères, modifier. Interprétation textuelle de la valeur d'état.
La mémorisation associée à ce paramètre doit être libérée par l'application après usage avec un appel à gss_release_buffer().

Valeurs de la fonction : code d'état GSS.

GSS_S_COMPLETE	Achèvement réussi.
GSS_S_BAD_MECH	Indique que la traduction conformément à un type de mécanisme pris en charge a été demandée.
GSS_S_BAD_STATUS	La valeur d'état n'a pas été reconnue, ou le type d'état n'était ni GSS_C_GSS_CODE ni GSS_C_MECH_CODE.

5.12 ²gss_duplicate_name

```
OM_uint32      gss_duplicate_name (
  OM_uint32          *minor_status,
  const gss_name_t   src_name,
  gss_name_t         *dest_name)
```

Objet : Crée un duplicata exact du nom interne existant src_name. Le nouveau dest_name sera indépendant du src_name (c'est-à-dire, src_name et dest_name doivent tous deux être libérés, et la libération de l'un ne devra pas affecter la validité de l'autre).

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

src_name gss_name_t, lecture. Nom interne à dupliquer.

dest_name gss_name_t, modifier. Copie résultante de <src_name>. La mémorisation associée à ce nom doit être libérée par l'application après usage avec un appel à gss_release_name().

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.
GSS_S_BAD_NAME Le paramètre src_name était mal formé.

5.13 gss_export_name

```
OM_uint32     gss_export_name (
  OM_uint32           *minor_status,
  const gss_name_t    input_name,
  gss_buffer_t        exported_name)
```

Objet : Produire une représentation de chaîne contiguë canonique d'un nom de mécanisme (MN), convenable pour une comparaison directe (par exemple, avec memcmp) à utiliser dans des fonctions d'autorisation (par exemple, faire correspondre des entrées dans une liste de contrôle d'accès). Le paramètre <input_name> doit spécifier un MN valide (c'est-à-dire, un nom interne généré par gss_accept_sec_context ou par gss_canonicalize_name).

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme

input_name gss_name_t, lecture. Le MN à exporter.

exported_name gss_buffer_t, chaîne d'octets, modifier.
Forme de chaîne contiguë canonique de <input_name>. La mémorisation associée à cette chaîne doit être libérée par l'application après usage avec gss_release_buffer().

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.
GSS_S_NAME_NOT_MN Le nom interne fourni n'était pas un nom de mécanisme.
GSS_S_BAD_NAME Le nom interne fourni était mal formé.
GSS_S_BAD_NAMETYPE Le nom interne était d'un type non accepté par la mise en œuvre GSS-API.

5.14 gss_export_sec_context

```
OM_uint32     gss_export_sec_context (
  OM_uint32       *minor_status,
  gss_ctx_id_t    *context_handle,
  gss_buffer_t    interprocess_token)
```

Objet : Fourni pour prendre en charge le partage du travail entre plusieurs processus. Ce sous-programme va normalement être utilisé par l'accepteur de contexte, dans une application où un seul procès reçoit les demandes de connexion entrantes et accepte les contextes de sécurité sur elles, puis passe le contexte établi à un ou plusieurs autres processus pour l'échange de message. gss_export_sec_context() désactive le contexte de sécurité pour le processus d'appel et crée un jeton interprocès qui, lorsque il est passé à gss_import_sec_context dans un autre procès, va réactiver le contexte dans le second procès. Une seule instanciation d'un certain contexte peut être active à un instant donné ; une tentative suivante d'un exporteur de contexte d'accéder au contexte de sécurité exporté échouera.

La mise en œuvre peut imposer l'ensemble de processus par lequel le jeton interprocès peut être importé, soit comme une fonction de politique de sécurité locale, soit comme résultat de décisions de mise en œuvre. Par exemple, certaines mises en œuvre peut imposer que les contextes ne soient passés qu'entre des processus qui fonctionnent sous le même compte, ou qui font partie du même groupe de processus.

Le jeton interprocès peut contenir des informations sensibles pour la sécurité (par exemple, des clés de chiffrement). Bien que les mécanismes soient encouragés à éviter de placer de telles informations sensibles au sein des jetons interprocès, ou de chiffrer le jeton avant de le retourner à l'application, ceci peut n'être pas possible dans une mise en œuvre GSS-API normale de bibliothèque d'objets. L'application doit donc veiller à protéger le jeton interprocès, et s'assurer que tout processus auquel le jeton est transféré est digne de confiance.

Si la création du jeton interprocès est réussie, la mise en œuvre devra désallouer toutes les ressources associées au contexte de sécurité au niveau du processus, et régler le lien de contexte à GSS_C_NO_CONTEXT. Dans le cas où une erreur rendrait impossible l'achèvement de l'exportation du contexte de sécurité, la mise en œuvre ne doit pas retourner un jeton interprocès, et devrait s'efforcer de laisser inchangé le contexte de sécurité référencé par le paramètre context_handle. Si c'est impossible, il est permis à la mise en œuvre de supprimer le contexte de sécurité, pourvu qu'elle règle aussi le paramètre context_handle à GSS_C_NO_CONTEXT.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme

context_handle gss_ctx_id_t, modifier. Lien de contexte qui identifie le contexte à transférer.

interprocess_token mémoire tampon, opaque, modifier.
Jeton à transférer au processus cible. La mémorisation associée à ce jeton doit être libérée par l'application après usage avec un appel à gss_release_buffer().

Valeurs de la fonction :	code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.
GSS_S_CONTEXT_EXPIRED	Le contexte est périmé.
GSS_S_NO_CONTEXT	Le contexte était invalide.
GSS_S_UNAVAILABLE	L'opération n'est pas acceptée.

5.15 gss_get_mic

```
OM_uint32        gss_get_mic (
  OM_uint32                 *minor_status,
  constante gss_ctx_id_t    context_handle,
  gss_qop_t                 qop_req,
  constante gss_buffer_t    message_buffer,
  gss_buffer_t              msg_token)
```

Objet : Génère un MIC cryptographique pour le message fourni, et place le MIC dans un jeton pour transfert à l'application homologue. Le paramètre qop_req permet un choix entre plusieurs algorithmes de chiffrement, si c'est accepté par le mécanisme choisi.

Comme certains protocoles de niveau application souhaitent utiliser des jetons émis par gss_get_mic() pour fournir un "tramage sécurisé", les mises en œuvre doivent accepter la déduction des MIC de messages de longueur zéro.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique de la mise en œuvre.

context_handle gss_ctx_id_t, lecture. Identifie le contexte sur lequel le message sera envoyé.

qop_req gss_qop_t, lecture, facultatif.
Spécifie la qualité de protection demandée. Les appelants sont invités, au nom de la portabilité, à accepter la qualité de protection par défaut offerte par le mécanisme choisi, qui peut être demandée en spécifiant GSS_C_QOP_DEFAULT pour ce paramètre. Si une force de protection non prise en charge est demandée, gss_get_mic va retourner un état majeur de GSS_S_BAD_QOP.

message_buffer mémoire tampon, opaque, lecture. Message à protéger.

msg_token mémoire tampon, opaque, modifier. Mémoire tampon pour recevoir le jeton. L'application doit libérer la mémorisation associée à cette mémoire tampon après usage avec un appel à gss_release_buffer().

Valeurs de la fonction :	code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.
GSS_S_CONTEXT_EXPIRED	Le contexte est déjà périmé.
GSS_S_NO_CONTEXT	Le paramètre context_handle n'identifiait pas un contexte valide.
GSS_S_BAD_QOP	La QOP spécifiée n'est pas acceptée par le mécanisme.

5.16 gss_import_name

```
OM_uint32        gss_import_name (
  OM_uint32            *minor_status,
  const gss_buffer_t   input_name_buffer,
  const gss_OID        input_name_type,
  gss_name_t           *output_name)
```

Objet : Convertit un nom de chaîne contiguë en forme interne. En général, le nom interne retourné (via le paramètre <nom_de_sortie>) ne sera pas un MN ; une exception à cela est si le <input_name_type> indique que la chaîne contiguë fournie via le paramètre <input_name_buffer> est du type GSS_C_NT_EXPORT_NAME, auquel cas le nom interne retourné sera un MN pour le mécanisme qui a exporté le nom.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme

input_name_buffer Mémoire tampon, chaîne d'octets, lecture.
Mémoire tampon qui contient le nom de chaîne contiguë à convertir.

input_name_type Identifiant d'objet, lecture, facultatif.
Identifiant d'objet qui spécifie le type du nom imprimable. Les applications peuvent spécifier soit GSS_C_NO_OID pour utiliser une syntaxe imprimable par défaut spécifique du mécanisme, soit un OID reconnu par la mise en œuvre GSS-API pour désigner un espace de noms spécifique.

output_name gss_name_t, modifier. Nom retourné en forme interne. La mémorisation associée à ce nom doit être libérée par l'application après usage avec un appel à gss_release_name().

Valeurs de la fonction :	code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.
GSS_S_BAD_NAMETYPE	Le type de nom d'entrée n'a pas été reconnu.
GSS_S_BAD_NAME	Le paramètre input_name n'a pas pu être interprété comme nom du type spécifié.
GSS_S_BAD_MECH	Le type de nom d'entrée était GSS_C_NT_EXPORT_NAME, mais le mécanisme contenu dans le nom d'entrée n'est pas accepté.

5.17 gss_import_sec_context

```
OM_uint32      gss_import_sec_context (
  OM_uint32                *minor_status,
  constante gss_buffer_t   interprocess_token,
  gss_ctx_id_t             *context_handle)
```

Objet : Permettre à un processus d'importer un contexte de sécurité établi par un autre processus. Un certain jeton interprocessus peut être importé une seule fois. Voir gss_export_sec_context.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme

interprocess_token mémoire tampon, opaque, modifier. Jeton reçu du processus exportateur.

context_handle gss_ctx_id_t, modifier.
Lien de contexte du contexte qui vient d'être réactivé. Les ressources associées à ce lien de contexte doivent être libérées par l'application après usage avec un appel à gss_delete_sec_context().

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi..
GSS_S_NO_CONTEXT	Le jeton ne contient pas une référence de contexte valide.
GSS_S_DEFECTIVE_TOKEN	Le jeton est invalide.
GSS_S_UNAVAILABLE	L'opération est indisponible.
GSS_S_UNAUTHORIZED	La politique locale empêche d'importation de ce contexte par le processus actuel.

5.18 gss_indicate_mechs

```
OM_uint32   gss_indicate_mechs (
  OM_uint32       *minor_status,
  gss_OID_set     *mech_set)
```

Objet : Permet à une application de déterminer quels mécanismes de sécurité sous-jacents sont disponibles.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

mech_set Ensemble d'identifiants d'objet, modifier. Ensemble de mécanismes pris en charge par la mise en œuvre. La valeur de gss_OID_set retournée sera un ensemble d'OID alloués de façon dynamique qui devrait être libéré par l'appelant après usage avec un appel à gss_release_oid_set().

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.

5.19 gss_init_sec_context

```
OM_uint32      gss_init_sec_context (
  OM_uint32                         *minor_status,
  constante gss_cred_id_t           initiator_cred_handle,
  gss_ctx_id_t                      *context_handle,\
  constante gss_name_t              target_name,
  constante gss_OID                 mech_type,
  OM_uint32                         req_flags,
  OM_uint32                         time_req,
  constante gss_channel_bindings_t  input_chan_bindings,
  constante gss_buffer_t            input_token
  gss_OID                           *actual_mech_type,
  gss_buffer_t                      output_token,
  OM_uint32                         *ret_flags,
  OM_uint32                         *time_rec )
```

Objet : Initie l'établissement d'un contexte de sécurité entre l'application et un homologue distant. Initialement, le paramètre input_token devrait être spécifié soit comme GSS_C_NO_BUFFER, soit comme un pointeur sur un objet gss_buffer_desc dont le champ de longueur contient la valeur zéro. Le sous-programme peut retourner un output_token qui devrait être transféré à l'application homologue, et l'application homologue va le présenter à gss_accept_sec_context. Si aucun jeton ne doit être envoyé, gss_init_sec_context va indiquer cela en réglant le champ de longueur de l'argument output_token à zéro. Pour achever l'établissement du contexte, un ou plusieurs jetons de réponse peuvent être nécessaires de la part de l'application homologue ; si il en est ainsi, gss_init_sec_context va retourner un état contenant le bit d'informations supplémentaires GSS_S_CONTINUE_NEEDED. Dans ce cas, gss_init_sec_context devrait être invoqué à nouveau lorsque le jeton de réponse est reçu de l'application homologue, passant le jeton de réponse à gss_init_sec_context via les paramètres input_token.

Les applications portables devraient être construites de façon à utiliser la longueur de jeton et l'état de retour pour déterminer si un jeton a besoin d'être envoyé ou attendu. Donc, un appelant portable normal devrait toujours invoquer gss_init_sec_context au sein d'une boucle:

```
int context_established = 0;
 gss_ctx_id_t context_hdl = GSS_C_NO_CONTEXT;
      ...
 input_token->longueur = 0;

  lorsque (!context_established) {
   maj_stat = gss_init_sec_context(&min_stat,
                                   cred_hdl,
                                   &context_hdl,
                                   target_name,
                                   desired_mech,
                                   desired_services,
                                   desired_time,
                                   input_bindings,
                                   input_token,
                                   &actual_mech,
                                   output_token,
                                   &actual_services,
                                   &actual_time);
    si (GSS_ERROR(maj_stat)) {
     report_error(maj_stat, min_stat);
    };

    si (output_token->length != 0) {
     send_token_to_peer(output_token);
     gss_release_buffer(&min_stat, output_token)
    };
   si (GSS_ERROR(maj_stat)) {

     si (context_hdl != GSS_C_NO_CONTEXT)
       gss_delete_sec_context(&min_stat,
                              &context_hdl,
                              GSS_C_NO_BUFFER);
     break;
    };

   si (maj_stat & GSS_S_CONTINUE_NEEDED) {
     receive_token_from_peer(input_token);
    } autrement {
     context_established = 1;
    };
  };
```

Chaque fois que le sous-programme retourne un état majeur qui comporte la valeur GSS_S_CONTINUE_NEEDED, le contexte n'est pas pleinement établi et les restrictions suivantes s'appliquent aux paramètres de sortie :

La valeur retournée via le paramètre time_rec est indéfinie sauf si le paramètre d'accompagnement ret_flags contient le bit GSS_C_PROT_READY_FLAG, indiquant que les services par message peuvent être appliqués avant un état d'achèvement réussi, la valeur retournée via le paramètre actual_mech_type est indéfinie jusqu'à ce que le sous-programme retourne une valeur d'état majeur de GSS_S_COMPLETE.

Les valeurs des bits GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG et GSS_C_ANON_FLAG retournées via le paramètre ret_flags devraient contenir les valeurs que la mise en œuvre s'attend à être valides si l'établissement de contexte devait réussir. En particulier, si l'application a demandé un service tel qu'une délégation ou une authentification anonyme via l'argument req_flags, et qu'un tel service est indisponible de la part du mécanisme sous-jacent, gss_init_sec_context devrait générer un jeton qui ne va pas fournir le service, et indiquera via l'argument ret_flags que le service ne sera pas pris en charge. L'application peut choisir d'interrompre l'établissement de contexte en invoquant gss_delete_sec_context (si elle

ne peut pas continuer en l'absence de ce service) ou elle peut choisir de transmettre le jeton et continuer l'établissement de contexte (si le service était simplement désiré mais pas obligatoire).

Les valeurs des bits GSS_C_PROT_READY_FLAG et GSS_C_TRANS_FLAG au sein de ret_flags devraient indiquer l'état actuel au moment du retour de gss_init_sec_context, que le contexte soit ou non pleinement établi.
Les mises en œuvre GSS-API qui prennent en charge la protection par message sont invitées à établir GSS_C_PROT_READY_FLAG dans le ret_flags final retourné à un appelant (c'est-à-dire, lorsque il est accompagné par un code d'état GSS_S_COMPLETE). Cependant, les applications ne devraient pas compter sur ce comportement car le fanion n'était pas défini dans la version 1 de GSS-API. Les applications devraient plutôt déterminer quels services par message sont disponibles après la réussite d'un établissement de contexte conformément aux valeurs de GSS_C_INTEG_FLAG et GSS_C_CONF_FLAG.

Tous les autres bits au sein de l'argument ret_flags devraient être réglés à zéro.

Si l'appel initial de gss_init_sec_context() échoue, la mise en œuvre ne devrait pas créer un objet contexte, et devrait laisser la valeur du paramètre context_handle réglée à GSS_C_NO_CONTEXT pour l'indiquer. Dans le cas d'un échec d'un appel ultérieur, il est permis à la mise en œuvre de supprimer le contexte de sécurité "à moitié construit" (auquel cas, elle devrait régler le paramètre context_handle à GSS_C_NO_CONTEXT), mais le comportement préféré est de laisser inchangé le contexte de sécurité pour que l'application le supprime (en utilisant gss_delete_sec_context).

Durant l'établissement de contexte, les bits GSS_S_OLD_TOKEN et GSS_S_DUPLICATE_TOKEN d'état pour information indiquent des erreurs fatales, et les mécanismes GSS-API devraient toujours les retourner en association avec une erreur de sous-programme de GSS_S_FAILURE. Cette exigence d'appariement n'existait pas dans la version 1 de la spécification GSS-API, de sorte que les applications qui souhaitent fonctionner sous une mise en œuvre de version 1 doivent faire un cas particulier de ces codes.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

initiator_cred_handle gss_cred_id_t, lecture, facultatif. Liens pour accréditifs réclamés. Fournit GSS_C_NO_CREDENTIAL pour agir comme principal initiateur par défaut. Si aucun initiateur par défaut n'est défini, la fonction va retourner GSS_S_NO_CRED.

context_handle gss_ctx_id_t, lecture/modifier.
Lien de contexte pour nouveau contexte. Fournit GSS_C_NO_CONTEXT pour le premier appel ; utilise la valeur retournée par le premier appel dans les appels de continuation. Les ressources associées à ce lien de contexte doivent être libérées par l'application après usage avec un appel à gss_delete_sec_context().

target_name gss_name_t, lecture. Nom de la cible.

mech_type OID, lecture, facultatif. Identifiant d'objet du mécanisme désiré. Fournit GSS_C_NO_OID pour obtenir une valeur spécifique de la mise en œuvre.

req_flags gabarit binaire, lecture. Contient divers fanions indépendants, dont chacun demande que le contexte prenne en charge une option de service spécifique. Les noms symboliques sont fournis pour chaque fanion, et les noms symboliques correspondant aux fanions requis devraient être logiquement Ouixés ensemble pour former la valeur du gabarit binaire. Les fanions sont :

GSS_C_DELEG_FLAG
Vrai – Délègue les accréditifs à l'homologue distant.
Faux – Ne pas déléguer.

GSS_C_MUTUAL_FLAG
Vrai – Demande que l'homologue distant s'authentifie.
Faux – S'authentifie seulement lui-même auprès de l'homologue distant.

GSS_C_REPLAY_FLAG
Vrai – Active la détection de répétition pour les messages protégés par gss_wrap ou gss_get_mic.
Faux – Ne pas tenter de détecter les messages répétés.

GSS_C_SEQUENCE_FLAG
Vrai – Active la détection des messages protégés hors séquence.
Faux – Ne pas tenter de détecter les messages hors séquence.

GSS_C_CONF_FLAG
Vrai – Demande que le service de confidentialité soit disponible (via gss_wrap).
Faux – Aucun service de confidentialité par message n'est requis.

GSS_C_INTEG_FLAG
Vrai – Demande que le service d'intégrité soit rendu disponible (via gss_wrap ou gss_get_mic).
Faux – Aucun service d'intégrité par message n'est requis.
GSS_C_ANON_FLAG
Vrai – Ne pas révéler l'identité de l'initiateur à l'accepteur.
Faux – Authentifier normalement.

time_req Entier, lecture, facultatif
Nombre désiré de secondes pendant lequel le contexte devrait rester valide. Fournir 0 pour demander une période de validité par défaut.

input_chan_bindings liens de canal, lecture, facultatif.
Liens spécifiques de l'application. Permet à l'application de lier en toute sécurité les informations d'identification de canal au contexte de sécurité. Spécifier GSS_C_NO_CHANNEL_BINDINGS si les liens de canal ne sont pas utilisés.

input_token Mémoire tampon, opaque, lecture, facultatif (voir le texte).
Jeton reçu de l'application homologue. Fournit GSS_C_NO_BUFFER, ou un pointeur à une mémoire tampon qui contient la valeur GSS_C_EMPTY_BUFFER sur l'appel initial.

actual_mech_type OID, modifier, facultatif. Mécanisme réellement utilisé. L'OID retourné via ce paramètre sera un pointeur sur une mémorisation statique qui devrait être traitée en lecture seule ; en particulier l'application ne devrait pas tenter de la libérer. Spécifier NUL si non exigé.

output_token Mémoire tampon, opaque, modifier.
Jeton à envoyer à l'application homologue. Si le champ de longueur de la mémoire tampon retournée est zéro, aucun jeton ne doit être envoyé à l'application homologue. La mémorisation associée à cette mémoire tampon doit être libérée par l'application après usage avec un appel à gss_release_buffer().

ret_flags Gabarit binaire, modifier., facultatif.
Contient divers fanions indépendants, dont chacun indique que le contexte prend en charge une option de service spécifique. Spécifier NUL si non exigé. Les noms symboliques sont fournis pour chaque fanion, et les noms symboliques correspondant aux fanions requis devraient être combinés par l'opérateur logique ET avec la valeur de ret_flags pour vérifier si une certaine option est acceptée par le contexte. Les fanions sont :
GSS_C_DELEG_FLAG
Vrai – Les accréditifs ont été délégués à l'homologue distant.
Faux – Aucun accréditif n'a été délégué.
GSS_C_MUTUAL_FLAG
Vrai – L'homologue distant s'est authentifié.
Faux – L'homologue distant ne s'est pas authentifié.
GSS_C_REPLAY_FLAG
Vrai – La répétition des messages protégés sera détectée.
Faux – Les messages répétés ne seront pas détectés.
GSS_C_SEQUENCE_FLAG
Vrai – Les messages protégés hors séquence seront détectés.
Faux – Les messages hors séquence ne seront pas détectés.
GSS_C_CONF_FLAG
Vrai – Le service de confidentialité peut être invoqué en appelant le sous-programme gss_wrap.
Faux – Aucun service de confidentialité (via gss_wrap) n'est disponible. gss_wrap va fournir seulement les services d'encapsulation de message, d'authentification et d'intégrité.
GSS_C_INTEG_FLAG
Vrai – Le service d'intégrité peut être invoqué en appelant les sous-programmes gss_get_mic ou gss_wrap.
Faux – L'intégrité par message est indisponible.
GSS_C_ANON_FLAG
Vrai – L'identité de l'initiateur n'a pas été révélée, et ne sera pas révélée si un jeton émis est passé à l'accepteur.
Faux – L'identité de l'initiateur a été ou sera authentifiée normalement.
GSS_C_PROT_READY_FLAG
Vrai – Les services de protection (comme spécifié par les états de GSS_C_CONF_FLAG et GSS_C_INTEG_FLAG) sont disponibles à l'utilisation si l'état mineur d'accompagnement retourne une valeur de GSS_S_COMPLETE ou de GSS_S_CONTINUE_NEEDED.
Faux – Les services de protection (comme spécifié par les états de GSS_C_CONF_FLAG et GSS_C_INTEG_FLAG) ne sont disponibles que si l'état majeur qui les accompagne retourne la

valeur de GSS_S_COMPLETE.

GSS_C_TRANS_FLAG
Vrai – Le contexte de sécurité résultant peut être transféré aux autres processus via un appel à gss_export_sec_context().
Faux – Le contexte de sécurité n'est pas transférable.

Tous les autres bits devraient être réglés à zéro.

time_rec Entier, modifier, facultatif.
Nombre de secondes pendant lequel le contexte va rester valide. Si la mise en œuvre n'accepte pas l'expiration de contexte, la valeur GSS_C_INDEFINITE sera retournée. Spécifier NUL si non exigé.

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.
GSS_S_CONTINUE_NEEDED	Indique qu'un jeton provenant de l'application homologue est exigé pour terminer le contexte, et que gss_init_sec_context doit être invoqué de nouveau avec ce jeton.
GSS_S_DEFECTIVE_TOKEN	Indique l'échec des vérifications de cohérence effectuées sur le jeton d'entrée.
GSS_S_DEFECTIVE_CREDENTIAL	Indique l'échec des vérifications de cohérence effectuées sur les accréditifs.
GSS_S_NO_CRED	Les accréditifs fournis n'étaient pas valides pour l'initiation de contexte, ou le lien d'accréditif ne faisait référence à aucun accréditif.
GSS_S_CREDENTIALS_EXPIRED	Les accréditifs référencés sont périmés.
GSS_S_BAD_BINDINGS	Le jeton d'entrée contient des liens de canal différents de ceux spécifiés via le paramètre input_chan_bindings.
GSS_S_BAD_SIG	Le jeton d'entrée contient un MIC invalide, ou un MIC qui n'a pas pu être vérifié.
GSS_S_OLD_TOKEN	Le jeton d'entrée est trop vieux. C'est une erreur fatale durant l'établissement de contexte.
GSS_S_DUPLICATE_TOKEN	Le jeton d'entrée est valide, mais est un duplicata d'un jeton déjà traité. C'est une erreur fatale durant l'établissement de contexte.
GSS_S_NO_CONTEXT	Indique que le lien de contexte fourni ne se réfère pas à un contexte valide.
GSS_S_BAD_NAMETYPE	Le paramètre target_name fourni contient un type de nom invalide ou non accepté.
GSS_S_BAD_NAME	Le paramètre target_name fourni est mal formé.
GSS_S_BAD_MECH	Le mécanisme spécifié n'est pas accepté par l'accréditif fourni, ou n'est pas reconnu par la mise en œuvre.

5.20 gss_inquire_context

```
OM_uint32      gss_inquire_context (
  OM_uint32           *minor_status,
  const. gss_ctx_id_t context_handle,
  gss_name_t          *src_name,
  gss_name_t          *targ_name,
  OM_uint32           *lifetime_rec,
  gss_OID             *mech_type,
  OM_uint32           *ctx_flags,
  int                 *locally_initiated,
  int                 *open )
```

Objet : Obtient des informations sur un contexte de sécurité. L'appelant doit avoir déjà obtenu un lien qui se réfère au contexte, bien que celui-ci n'ait pas besoin d'être pleinement établi.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme

context_handle gss_ctx_id_t, lecture. Lien qui se réfère au contexte de sécurité.

src_name gss_name_t, modifier, facultatif.
Nom de l'initiateur de contexte. Si le contexte était établi en utilisant l'authentification anonyme, et si l'application qui invoque gss_inquire_context est l'accepteur de contexte, un nom anonyme sera retourné. La mémorisation associée à ce nom doit être libérée par l'application après usage avec un appel à gss_release_name(). Spécifier NUL si non exigé.

targ_name gss_name_t, modifier., facultatif.
Nom de l'accepteur de contexte. La mémorisation associée à ce nom doit être libérée par l'application après usage avec un appel à gss_release_name(). Si l'accepteur du contexte ne s'est pas authentifié, et si l'initiateur n'avait pas spécifié un nom de cible dans son appel à gss_init_sec_context(), la valeur GSS_C_NO_NAME sera retournée. Spécifier NUL si non exigé.

lifetime_rec Entier, modifier, facultatif.
Nombre de secondes pendant lequel le contexte va rester valide. Si le contexte a expiré, ce paramètre sera réglé à zéro. Si la mise en œuvre n'accepte pas l'expiration de contexte, la valeur GSS_C_INDEFINITE sera retournée. Spécifier NUL si non exigé.

mech_type gss_OID, modifier, facultatif.
Mécanisme de sécurité qui fournit le contexte. L'OID retourné sera un pointeur sur la mémorisation statique qui devrait être traitée en lecture seule par l'application ; en particulier l'application ne devrait pas tenter de la libérer. Spécifier NUL si non exigé.

ctx_flags Gabarit binaire, modifier, facultatif.
Contient divers fanions indépendants, dont chacun indique que le contexte prend en charge (ou est supposé prendre en charge, si ctx_open est faux) une option spécifique de service. Si il n'est pas nécessaire, spécifier NUL. Des noms symboliques sont fournis pour chaque fanion, et les noms symboliques correspondant aux fanions requis devraient être soumis à l'opérateur logique ET avec la valeur ret_flags pour vérifier si une certaine option est acceptée par le contexte. Les fanions sont :

GSS_C_DELEG_FLAG
Vrai – Les accréditifs ont été délégués de l'initiateur à l'accepteur.
Faux – Aucun accréditif n'est délégué.

GSS_C_MUTUAL_FLAG
Vrai – L'accepteur s'est authentifié auprès de l'initiateur.
Faux – L'accepteur ne s'est pas authentifié.

GSS_C_REPLAY_FLAG
Vrai – La répétition des messages protégés sera détectée.
Faux – Les messages répétés ne seront pas détectés.

GSS_C_SEQUENCE_FLAG
Vrai – Les messages protégés hors séquence seront détectés.
Faux – Les messages hors séquence ne seront pas détectés.

GSS_C_CONF_FLAG
Vrai – Le service de confidentialité peut être invoqué en appelant le sous-programme gss_wrap.
Faux – Aucun service de confidentialité (via gss_wrap) disponible. gss_wrap fournira seulement les services d'encapsulation de message, d'authentification d'origine et d'intégrité des données.

GSS_C_INTEG_FLAG
Vrai – Le service d'intégrité peut être invoqué en appelant les sous-programmes gss_get_mic ou gss_wrap.
Faux – Le service d'intégrité par message n'est pas disponible.

GSS_C_ANON_FLAG
Vrai – L'identité de l'initiateur ne sera pas révélée à l'accepteur. Le paramètre src_name (si il est demandé) contient un nom interne anonyme.
Faux – L'initiateur a été authentifié normalement.

GSS_C_PROT_READY_FLAG
Vrai – Les services de protection (comme spécifié par les états de GSS_C_CONF_FLAG et GSS_C_INTEG_FLAG) sont disponibles pour utilisation.
Faux – Les services de protection (comme spécifié par les états de GSS_C_CONF_FLAG et GSS_C_INTEG_FLAG) ne sont disponibles que si le contexte est pleinement établi (c'est-à-dire, si le paramètre open est différent de zéro).

GSS_C_TRANS_FLAG
Vrai – Le contexte de sécurité résultant peut être transféré aux autres processus via un appel à gss_export_sec_context().
Faux – Le contexte de sécurité n'est pas transférable.

locally_initiated Booléen, modifier.
Non zéro si l'application invoquante est l'initiateur de contexte. Spécifier NUL si non exigé.

open Booléen, modifier.
Non zéro si le contexte est pleinement établi ; zéro si un jeton d'établissement de contexte est attendu de l'application homologue. Spécifier NUL si non exigé.

Valeurs de la fonction :	code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.
GSS_S_NO_CONTEXT	On ne peut pas accéder au contexte référencé.

5.21 gss_inquire_cred

```
OM_uint32      gss_inquire_cred (
  OM_uint32              *minor_status,
  const. gss_cred_id_t   cred_handle,
  gss_name_t             *name,
  OM_uint32              *lifetime,
  gss_cred_usage_t       *cred_usage,
  gss_OID_set            *mechanisms )
```

Objet : Obtient des informations sur un accréditif.

Paramètres :

minor_status Entier, modifier.
Code d'état spécifique du mécanisme

cred_handle gss_cred_id_t, lecture.
Lien qui se réfère à l'accréditif cible. Spécifier GSS_C_NO_CREDENTIAL pour chercher le principal d'initiateur par défaut.

name gss_name_t, modifier, facultatif.
Nom dont l'identité est affirmée par l'accréditif. La mémorisation associée à ce nom devrait être libérée par l'application après usage avec un appel à gss_release_name(). Spécifier NUL si non exigé.

lifetime Entier, modifier, facultatif.
Nombre de secondes pendant lequel l'accréditif va rester valide. Si l'accréditif est périmé, ce paramètre sera réglé à zéro. Si la mise en œuvre n'accepte pas l'expiration d'accréditif, la valeur GSS_C_INDEFINITE sera retournée. Spécifier NUL si non exigé.

cred_usage gss_cred_usage_t, modifier, facultatif.
Comment l'accréditif peut être utilisé. Un des suivants : GSS_C_INITIATE, GSS_C_ACCEPT, GSS_C_BOTH. Spécifier NUL si non exigé.

mechanisms gss_OID_set, modifier, facultatif.
Ensemble des mécanismes acceptés par l'accréditif. La mémorisation associée à cet ensemble d'OID doit être libérée par l'application après usage avec un appel à gss_release_oid_set(). Spécifier NUL si non exigé.

Valeurs de la fonction :	code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.
GSS_S_NO_CRED	On n'a pas pu accéder aux accréditifs référencés.
GSS_S_DEFECTIVE_CREDENTIAL	Les accréditifs référencés sont invalides.
GSS_S_CREDENTIALS_EXPIRED	Les accréditifs référencés sont périmés. Si le paramètre Durée de vie n'a pas été passé comme NUL, il sera réglé à 0.

5.22 gss_inquire_cred_by_mech

```
OM_uint32      gss_inquire_cred_by_mech (
  OM_uint32              *minor_status,
  const. gss_cred_id_t   cred_handle,
  const. gss_OID         mech_type,
  gss_name_t             *name,
  OM_uint32              *initiator_lifetime,
  OM_uint32              *acceptor_lifetime,
  gss_cred_usage_t       *cred_usage )
```

Objet : Obtient des informations par mécanisme sur un accréditif.

Paramètres :

minor_status Entier, modifier.
Code d'état spécifique du mécanisme

cred_handle gss_cred_id_t, lecture.
Lien qui se réfère à l'accréditif cible. Spécifie GSS_C_NO_CREDENTIAL pour s'enquérir du principal initiateur par défaut.

mech_type gss_OID, lecture.
Mécanisme pour lequel les informations devraient être retournées.

name gss_name_t, modifier, facultatif.
Nom dont l'accréditif affirme l'identité. La mémorisation associée à ce nom doit être libérée par l'application après usage avec un appel à gss_release_name(). Spécifier NUL si non exigé.

initiator_lifetime Entier, modifier, facultatif.
Nombre de secondes pendant lequel l'accréditif va rester capable d'initier des contextes de sécurité sous le mécanisme spécifié. Si l'accréditif ne peut plus être utilisé pour initier des contextes, ou si l'usage de l'accréditif pour ce mécanisme est GSS_C_ACCEPT, ce paramètre sera réglé à zéro. Si la mise en œuvre n'accepte pas l'expiration des accréditifs d'initiateur, la valeur GSS_C_INDEFINITE sera retournée. Spécifier NUL si non exigé.

acceptor_lifetime Entier, modifier, facultatif.
Nombre de secondes pendant lequel l'accréditif va rester capable d'accepter des contextes de sécurité sous le mécanisme spécifié. Si l'accréditif ne peut plus être utilisé pour accepter des contextes, ou si l'usage d'accréditif pour ce mécanisme est GSS_C_INITIATE, ce paramètre va être réglé à zéro. Si la mise en œuvre n'accepte pas l'expiration des accréditifs d'accepteur, la valeur GSS_C_INDEFINITE sera retournée. Spécifier NUL si non exigé.

cred_usage gss_cred_usage_t, modifier, facultatif.
Comment l'accréditif peut être utilisé avec le mécanisme spécifié. C'est un des suivants : GSS_C_INITIATE, GSS_C_ACCEPT, GSS_C_BOTH. Spécifier NUL si non exigé.

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.
GSS_S_NO_CRED	On n'a pas pu accéder aux accréditifs référencés.
GSS_S_DEFECTIVE_CREDENTIAL	Les accréditifs référencés sont invalides.
GSS_S_CREDENTIALS_EXPIRED	Les accréditifs référencés sont périmés. Si le paramètre Durée de vie n'a pas été passé comme NUL, il sera réglé à 0.

5.23 gss_inquire_mechs_for_name

```
OM_uint32        gss_inquire_mechs_for_name (
  OM_uint32            *minor_status,
  const. gss_name_t    input_name,
  gss_OID_set          *mech_types )
```

Objet : Retourne l'ensemble de mécanismes acceptés par la mise en œuvre GSS-API qui peuvent être capables de traiter le nom spécifié.

Chaque mécanisme retourné va reconnaître au moins un élément au sein du nom. Il est permis à ce sous-programme d'être mis en œuvre au sein d'une couche GSS-API indépendante du mécanisme, en utilisant les informations de type contenues dans le nom présenté, et sur la base des informations d'enregistrement fournies par les mises en œuvre individuelles de mécanisme. Cela signifie que les ensembles de type de mécanisme retournés peuvent indiquer qu'un certain mécanisme va comprendre le nom alors qu'en fait il va refuser d'accepter le nom comme entrée à gss_canonicalize_name, gss_init_sec_context, gss_acquire_cred ou gss_add_cred (du fait de certaines propriétés spécifiques du nom, par opposition au type de nom). Donc, ce sous-programme ne devrait être utilisé que comme pré filtre pour un appel à un sous-programme spécifique de mécanisme ultérieur.

Paramètres :

minor_status Entier, modifier.
Code d'état spécifique de la mise en œuvre.

input_name gss_name_t, lecture.
Nom auquel se rapporte l'enquête.

mech_types gss_OID_set, modifier.
Ensemble de mécanismes qui peuvent accepter le nom spécifié. L'ensemble d'OID retourné doit être libéré par l'appelant après usage avec un appel à gss_release_oid_set().

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.
GSS_S_BAD_NAME Le paramètre input_name était mal formé.
GSS_S_BAD_NAMETYPE Le paramètre input_name contenait un type de nom invalide ou non accepté.

5.24 gss_inquire_names_for_mech

```
OM_uint32        gss_inquire_names_for_mech (
  OM_uint32           *minor_status,
  const. gss_OID      mécanisme,
  gss_OID_set         *name_types)
```

Objet : Retourne l'ensemble de types de nom acceptés par le mécanisme spécifié.

Paramètres :

minor_status Entier, modifier.
Code d'état spécifique de la mise en œuvre.

mechanism gss_OID, lecture.
Mécanisme à interroger.

name_types gss_OID_set, modifier.
Ensemble de types de nom acceptés par le mécanisme spécifié. L'ensemble d'OID retourné doit être libéré par l'application après usage avec un appel à gss_release_oid_set().

Valeurs de la fonction : code d'état GSS
GSS_S_COMPLETE Achèvement réussi.

5.25 gss_process_context_token

```
OM_uint32        gss_process_context_token (
  OM_uint32           *minor_status,
  const. gss_ctx_id_t context_handle,
  const. gss_buffer_t token_buffer)
```

Objet : Fournit un moyen de passer un jeton asynchrone au service de sécurité. La plupart des jetons de niveau contexte sont émis et traités de façon synchrone par gss_init_sec_context et gss_accept_sec_context, et l'application est informée de ce que d'autres jetons sont attendus par le bit d'état majeur GSS_C_CONTINUE_NEEDED. À l'occasion, un mécanisme peut avoir besoin d'émettre un jeton de niveau contexte à un moment où l'entité homologue n'attend pas de jeton. Par exemple, l'appel final de l'initiateur à gss_init_sec_context peut émettre un jeton et retourner un état de GSS_S_COMPLETE, mais l'appel de l'accepteur à gss_accept_sec_context peut échouer. Le mécanisme de l'accepteur peut souhaiter envoyer un jeton contenant une indication d'erreur à l'initiateur, mais l'initiateur n'attend pas un jeton à ce moment, croyant que le contexte est pleinement établi. Gss_process_context_token fournit un moyen pour passer un tel jeton au mécanisme à tout moment.

Paramètres :

minor_status Entier, modifier.
Code d'état spécifique de la mise en œuvre.

context_handle gss_ctx_id_t, lecture.
Lien de contexte sur lequel le jeton doit être traité.

token_buffer mémoire tampon, opaque, lecture. Jeton à traiter.

Valeurs de la fonction :	Code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.
GSS_S_DEFECTIVE_TOKEN	Indique que les vérifications de cohérence effectuées sur le jeton ont échoué.
GSS_S_NO_CONTEXT	Le lien de contexte ne se réfère pas à un contexte valide.

5.26 gss_release_buffer

```
OM_uint32      gss_release_buffer (
  OM_uint32      *minor_status,
  gss_buffer_t   buffer)
```

Objet : Mémorisation libre associée à une mémoire tampon. La mémorisation doit avoir été allouée par un sous-programme GSS-API. En plus de libérer la mémorisation associée, le sous-programme va mettre à zéro le champ de longueur dans le descripteur auquel se réfère le paramètre mémoire tampon, et les mises en œuvre sont invitées de plus à régler le champ pointeur à NUL dans le descripteur. Tout objet Mémoire tampon retourné par un sous-programme GSS-API peut être passé à gss_release_buffer (même si il n'y a pas de mémorisation associée à la mémoire tampon).

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme

buffer mémoire tampon, modifier.
La mémorisation associée à la mémoire tampon sera supprimée. L'objet gss_buffer_desc ne sera pas libéré, mais son champ de longueur sera mis à zéro.

Valeurs de la fonction :	code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.

5.27 gss_release_cred

```
OM_uint32      gss_release_cred (
  OM_uint32        *minor_status,
  gss_cred_id_t    *cred_handle)
```

Objet : Informe GSS-API que le lien d'accréditif spécifié n'est plus nécessaire à l'application, et libère les ressources associées. Les mises en œuvre sont invitées à régler cred_handle à GSS_C_NO_CREDENTIAL à l'achèvement réussi de cet appel.

Paramètres :

cred_handle gss_cred_id_t, modifier, facultatif.
Lien opaque qui identifie un accréditif à libérer. Si GSS_C_NO_CREDENTIAL est fourni, le sous-programme va s'achever avec succès, mais ne va rien faire.

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

Valeurs de la fonction :	code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.
GSS_S_NO_CRED	On n'a pas pu accéder aux accréditifs.

5.28 gss_release_name

```
OM_uint32      gss_release_name (
  OM_uint32      *minor_status,
  gss_name_t     *name)
```

Objet : Libère la mémorisation allouée par GSS-API associée à un nom de forme interne. Les mises en œuvre sont invitées à régler le nom à GSS_C_NO_NAME lors de l'achèvement réussi de cet appel.

Paramètres :

minor_status	Entier, modifier. Code d'état spécifique du mécanisme
name	gss_name_t, modifier. Le nom à supprimer.

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.
GSS_S_BAD_NAME	Le paramètre Nom ne contient pas un nom valide.

5.29 gss_release_oid_set

```
OM_uint32      gss_release_oid_set (
  OM_uint32      *minor_status,
  gss_OID_set    *set)
```

Objet : Libérer la mémorisation associée à un objet gss_OID_set généré par GSS-API. Le paramètre Set doit se référer à un ensemble d'OID qui a été retourné d'un sous-programme GSS-API. gss_release_oid_set() va libérer la mémorisation associée à chaque OID de membre individuel, la matrice d'éléments de l'ensemble d'OID, et le gss_OID_set_desc.

Les mises en œuvre sont invitées à régler le paramètre gss_OID_set à GSS_C_NO_OID_SET à l'achèvement réussi de ce sous-programme.

Paramètres :

minor_status	Entier, modifier. Code d'état spécifique du mécanisme
set	Ensemble d'OID d'objet, modifier. La mémorisation associée au gss_OID_set sera supprimée.

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.

5.30 gss_test_oid_set_member

```
OM_uint32        gss_test_oid_set_member (
  OM_uint32          *minor_status,
  const. gss_OID     member,
  const. gss_OID_set set,
  int                *present)
```

Objet : Interroge un ensemble d'identifiants d'objet pour déterminer si un identifiant d'objet spécifié est un membre. Ce sous-programme est destiné à être utilisé avec les ensembles d'OID retournés par gss_indicate_mechs(), gss_acquire_cred(), et gss_inquire_cred(), mais va aussi fonctionner avec des ensembles générés par l'usager.

Paramètres :

minor_status	Entier, modifier. Code d'état spécifique du mécanisme
member	Identifiant d'objet, lecture. L'identifiant d'objet dont la présence est à vérifier.
set	Ensemble d'identifiants d'objet, lecture. L'ensemble d'identifiants d'objet.
present	Booléen, modifier. Non zéro si l'OID spécifié est un membre de l'ensemble, zéro sinon.

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.

5.31 gss_unwrap

```
OM_uint32      gss_unwrap (
  OM_uint32           *minor_status,
  const. gss_ctx_id_t context_handle,
  const. gss_buffer_t input_message_buffer,
  gss_buffer_t        output_message_buffer,
  int                 *conf_state,
  gss_qop_t           *qop_state)
```

Objet : Convertit un message précédemment protégé par gss_wrap en une forme utilisable, vérifiant le MIC incorporé. Le paramètre conf_state indique si le message était chiffré ; le paramètre qop_state indique la force de la protection qui était utilisée pour fournir les services de confidentialité et d'intégrité.

Comme certains protocoles de niveau application peuvent souhaiter utiliser des jetons émis par gss_wrap() pour fournir un "tramage sécurisé", les mises en œuvre doivent accepter l'enveloppement et le désenveloppement des messages de longueur zéro.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

context_handle gss_ctx_id_t, lecture. Identifie le contexte sur lequel le message est arrivé.

input_message_buffer mémoire tampon, opaque, lecture. Message protégé.

output_message_buffer mémoire tampon, opaque, modifier.
Mémoire tampon pour recevoir un message désenveloppé. La mémorisation associée à cette mémoire tampon doit être libérée par l'application après usage avec un appel à gss_release_buffer().

conf_state Booléen, modifier., facultatif.
Non zéro – La protection de la confidentialité et de l'intégrité ont été utilisées.
Zéro – Seul le service d'intégrité a été utilisé. Spécifier NUL si non exigé.

qop_state gss_qop_t, modifier, facultatif
Qualité de protection fournie. Spécifier NUL si non exigé.

Valeurs de la fonction :	code d'état GSS
GSS_S_COMPLETE	Achèvement réussi.
GSS_S_DEFECTIVE_TOKEN	Les vérifications de cohérence du jeton ont échoué.
GSS_S_BAD_SIG	Le MIC était incorrect.
GSS_S_DUPLICATE_TOKEN	Le jeton est valide, et contient un MIC correct pour le message, mais il a déjà été traité.
GSS_S_OLD_TOKEN	Le jeton est valide, et contient un MIC correct pour le message, mais il est trop vieux pour qu'on recherche s'il est un dupliqué.
GSS_S_UNSEQ_TOKEN	Le jeton est valide, et contient un MIC correct pour le message, mais il a été vérifié qu'il est hors séquence ; un jeton ultérieur a déjà été reçu.
GSS_S_GAP_TOKEN	Le jeton est valide, et contient un MIC correct pour le message, mais il a été vérifié qu'il est hors séquence ; un jeton attendu plus tôt n'a pas encore été reçu.
GSS_S_CONTEXT_EXPIRED	Le contexte est déjà périmé.
GSS_S_NO_CONTEXT	Le paramètre context_handle n'identifie pas un contexte valide.

5.32 gss_verify_mic

```
OM_uint32      gss_verify_mic (
  OM_uint32           *minor_status,
  const. gss_ctx_id_t context_handle,
  const. gss_buffer_t message_buffer,
  const. gss_buffer_t token_buffer,
  gss_qop_t           *qop_state)
```

Objet : Vérifie qu'un MIC cryptographique, contenu dans le paramètre Jeton, correspond au message fourni. Le paramètre

qop_state permet à un receveur de message de déterminer la force de la protection qui a été appliquée au message.

Comme certains protocoles de niveau application souhaitent utiliser des jetons émis par gss_wrap() pour fournir un "tramage sécurisé", les mises en œuvre doivent prendre en charge le calcul et la vérification des MIC sur les messages de longueur zéro.

Paramètres :

minor_status	Entier, modifier. Code d'état spécifique du mécanisme.
context_handle	gss_ctx_id_t, lecture. Identifie le contexte sur lequel le message est arrivé.
message_buffer	Mémoire tampon, opaque, lecture. Message à vérifier.
token_buffer	Mémoire tampon, opaque, lecture. Jeton associé au message.
qop_state	gss_qop_t, modifier, facultatif. Qualité de protection obtenue du MIC. Spécifier NUL si non exigé.

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.
GSS_S_DEFECTIVE_TOKEN	Les vérifications de cohérence sur le jeton ont échoué.
GSS_S_BAD_SIG	Le MIC est incorrect.
GSS_S_DUPLICATE_TOKEN	Le jeton est valide et contient un MIC correct pour le message, mais il a déjà été traité.
GSS_S_OLD_TOKEN	Le jeton est valide et contient un MIC correct pour le message, mais il est trop vieux pour qu'on recherche s'il est un dupliqué.
GSS_S_UNSEQ_TOKEN	Le jeton est valide et contient un MIC correct pour le message, mais il a été vérifié qu'il est hors séquence ; un jeton ultérieur a déjà été reçu.
GSS_S_GAP_TOKEN	Le jeton est valide et contient un MIC correct pour le message, mais il a été vérifié qu'il est hors séquence ; un jeton attendu plus tôt n'a pas encore été reçu.
GSS_S_CONTEXT_EXPIRED	Le contexte est déjà périmé.
GSS_S_NO_CONTEXT	Le paramètre context_handle n'identifie pas un contexte valide.

5.33 gss_wrap

```
OM_uint32      gss_wrap (
  OM_uint32            *minor_status,
  const. gss_ctx_id_t  context_handle,
  int                  conf_req_flag,
  gss_qop_t            qop_req
  const. gss_buffer_t  input_message_buffer,
  int                  *conf_state,
  gss_buffer_t         output_message_buffer )
```

Objet : Rattache un MIC cryptographique et chiffre facultativement le message d'entrée spécifié. Le message de sortie contient à la fois le MIC et le message. Le paramètre qop_req permet un choix entre plusieurs algorithmes de chiffrement, si ils sont acceptés par le mécanisme choisi.

Comme certains protocoles de niveau application peuvent souhaiter utiliser les jetons émis par gss_wrap() pour fournir un "tramage sécurisé", les mises en œuvre doivent prendre en charge l'enveloppement des messages de longueur zéro.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme.

context_handle gss_ctx_id_t, lecture. Identifie le contexte sur lequel le message sera envoyé.

conf_req_flag Booléen, lecture.
Non zéro – Les deux services de confidentialité et d'intégrité sont demandés.
Zéro – Seul le service d'intégrité est demandé.

qop_req gss_qop_t, lecture, facultatif.
Spécifie la qualité de protection demandée. Une valeur par défaut spécifique du mécanisme peut être demandée en réglant qop_req à GSS_C_QOP_DEFAULT. Si une force de protection non acceptée est demandée, gss_wrap va retourner un état majeur de GSS_S_BAD_QOP.

input_message_buffer Mémoire tampon, opaque, lecture. Message à protéger.

conf_state booléen, modifier, facultatif.
Non zéro – Les services de confidentialité, authentification de l'origine des données et intégrité ont été appliqués.
Zéro – Seuls les services d'intégrité et d'origine des données ont été appliqués.
Spécifier NUL si non exigé.

output_message_buffer Mémoire tampon, opaque, modifier.
Mémoire tampon pour recevoir le message protégé. La mémorisation associée à ce message doit être libérée par l'application après usage avec un appel à gss_release_buffer().

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.
GSS_S_CONTEXT_EXPIRED	Le contexte est déjà périmé.
GSS_S_NO_CONTEXT	Le paramètre context_handle n'identifie pas un contexte valide.
GSS_S_BAD_QOP	La QOP spécifiée n'est pas acceptée par le mécanisme.

5.34 gss_wrap_size_limit

```
OM_uint32     gss_wrap_size_limit (
  OM_uint32           *minor_status,
  const. gss_ctx_id_t context_handle,
  int                 conf_req_flag,
  gss_qop_t           qop_req,
  OM_uint32           req_output_size,
  OM_uint32           *max_input_size)
```

Objet :
Permet à une application de déterminer la taille maximum de message qui, si elle est présentée à gss_wrap avec les mêmes paramètres conf_req_flag et qop_req, va résulter en un jeton de sortie ne contenant pas plus de req_output_size octets.

Cet appel est destiné à l'usage des applications qui communiquent sur des protocoles qui imposent une taille maximum de message. Il permet à l'application de fragmenter les messages avant d'appliquer la protection.

Il est recommandé aux mises en œuvre GSS-API, mais elles n'y sont pas obligées, de détecter les valeurs de QOP invalides lorsque gss_wrap_size_limit() est invoqué. Ce sous-programme garantit seulement une taille maximum de message, et non la disponibilité de valeurs spécifiques de QOP pour la protection de message.

L'achèvement réussi. de cet appel ne garantit pas que gss_wrap va être capable de protéger un message de longueur max_input_size octets, car cette capacité peut dépendre de la disponibilité de ressources système au moment de l'invocation de gss_wrap. Cependant, si la mise en œuvre impose elle-même une limite supérieure à la longueur des messages qui peuvent être traités par gss_wrap, la mise en œuvre ne devrait pas retourner une valeur via max_input_bytes qui soit supérieure à cette longueur.

Paramètres :

minor_status Entier, modifier. Code d'état spécifique du mécanisme

context_handle gss_ctx_id_t, lecture. Lien qui se réfère à la sécurité sur laquelle les messages seront envoyés.

conf_req_flag Booléen, lecture.
Indique si il sera demandé à gss_wrap d'appliquer la protection de la confidentialité en plus de celle de l'intégrité. Voir la description du sous-programme de gss_wrap pour les détails.

qop_req gss_qop_t, lecture.
Indique le niveau de protection qu'il sera demandé à gss_wrap de fournir. Voir la description du sous-programme de gss_wrap pour les détails.

req_output_size Entier, lecture. Taille maximum désirée pour les jetons émis par gss_wrap.

max_input_size Entier, modifier. Taille maximum de message d'entrée qui peut être présentée à gss_wrap afin de garantir

que le jeton émis ne sera pas plus grand que req_output_size octets.

Valeurs de la fonction : code d'état GSS

GSS_S_COMPLETE	Achèvement réussi.
GSS_S_NO_CONTEXT	On n'a pas pu accéder au contexte référencé.
GSS_S_CONTEXT_EXPIRED	Le contexte est périmé.
GSS_S_BAD_QOP	La QOP spécifiée n'est pas prise en charge par le mécanisme.

6. Considérations pour la sécurité

Le présent document spécifie une interface de service pour les facilités et services de sécurité ; à ce titre, les considérations de sécurité apparaissent tout au long de la spécification. Néanmoins, il est approprié de résumer certains points spécifiques pertinents pour les mises en œuvre de GSS-API et les applications appelantes. L'usage de l'interface GSS-API ne fournit pas par lui-même de services ou assurances de sécurité ; ces attributs dépendent plutôt du ou des mécanismes sous-jacents que prend en charge une mise en œuvre GSS-API. Les appelants doivent être attentifs aux demandes faites aux appels GSS-API et aux indicateurs d'état retournés par GSS-API, car ils spécifient les caractéristiques du service de sécurité que va fournir GSS-API. Lorsque la facilité de transfert de contexte interprocessus est utilisée, les contrôles locaux appropriés devraient être appliqués aux jetons interprocès à accès restreint et aux données sensibles qu'ils contiennent.

Appendice A Fichier d'en-tête gssapi.h de GSS-API C

Les mises en œuvre GSS-API en langage C devraient inclure une copie du fichier d'en-tête suivant.

```
  #ifndef GSSAPI_H_
  #define GSSAPI_H_

/* D'abord, inclure stddef.h pour que size_t soit définie. */
  #include <stddef.h>

/* Si la plateforme accepte le fichier d'en-tête xom.h, il devrait être inclus ici. */
  #include <xom.h>

/* Définir ensuite les trois types dépendants de la mise en œuvre. */
  typedef <platform-specific> gss_ctx_id_t;
  typedef <platform-specific> gss_cred_id_t;
  typedef <platform-specific> gss_name_t;

/* Le type suivant doit être défini comme le plus petit entier naturel non signé accepté par la plateforme qui a au moins
     32 bits de précision. */
  typedef <platform-specific> gss_uint32;

  #ifdef OM_STRING
/* On a inclus le fichier d'en-tête xom.h. Vérifier que OM_uint32 est défini correctement. */

  #if sizeof(gss_uint32) != sizeof(OM_uint32)
  #error Incompatible definition of OM_uint32 from xom.h
  #endif

  typedef OM_object_identifier gss_OID_desc, *gss_OID;

  #else

/* On ne peut pas utiliser les définitions X/Open, de sorte qu'on aura les nôtres. */

typedef gss_uint32 OM_uint32;

typedef struct gss_OID_desc_struct {
    OM_uint32 length;
    void      *elements;
} gss_OID_desc, *gss_OID;
```

```
#endif

typedef struct gss_OID_set_desc_struct  {
    size_t     count;
    gss_OID    elements;
} gss_OID_set_desc, *gss_OID_set;

typedef struct gss_buffer_desc_struct {
    size_t length;
    void *value;
} gss_buffer_desc, *gss_buffer_t;

typedef struct gss_channel_bindings_struct {
    OM_uint32 initiator_addrtype;
    gss_buffer_desc initiator_address;
    OM_uint32 acceptor_addrtype;
    gss_buffer_desc acceptor_address;
    gss_buffer_desc application_data;
} *gss_channel_bindings_t;

/* Maintenant, on définit un Type de QOP comme un OM_uint32. */
typedef OM_uint32 gss_qop_t;

typedef int gss_cred_usage_t;

/* Bits de fanion pour les services de niveau contexte. */

#define GSS_C_DELEG_FLAG      1
#define GSS_C_MUTUAL_FLAG     2
#define GSS_C_REPLAY_FLAG     4
#define GSS_C_SEQUENCE_FLAG   8
#define GSS_C_CONF_FLAG       16
#define GSS_C_INTEG_FLAG      32
#define GSS_C_ANON_FLAG       64
#define GSS_C_PROT_READY_FLAG 128
#define GSS_C_TRANS_FLAG      256

/* Options d'usage d'accréditif. */
#define GSS_C_BOTH     0
#define GSS_C_INITIATE 1
#define GSS_C_ACCEPT   2

/* Types de code d'état pour gss_display_status. */
#define GSS_C_GSS_CODE  1
#define GSS_C_MECH_CODE 2

/* Définitions des constantes pour les familles d'adresse de liens de canal. */
#define GSS_C_AF_UNSPEC            0
#define GSS_C_AF_LOCAL             1
#define GSS_C_AF_INET              2
#define GSS_C_AF_IMPLINK           3
#define GSS_C_AF_PUP               4
#define GSS_C_AF_CHAOS             5
#define GSS_C_AF_NS                6
#define GSS_C_AF_NBS               7
#define GSS_C_AF_ECMA              8
#define GSS_C_AF_DATAKIT           9
#define GSS_C_AF_CCITT             10
#define GSS_C_AF_SNA               11
#define GSS_C_AF_DECnet            12
#define GSS_C_AF_DLI               13
#define GSS_C_AF_LAT               14
```

```
#define GSS_C_AF_HYLINK         15
#define GSS_C_AF_APPLETALK      16
#define GSS_C_AF_BSC            17
#define GSS_C_AF_DSS            18
#define GSS_C_AF_OSI            19
#define GSS_C_AF_X25            21
#define GSS_C_AF_NULLADDR       255

/* Diverses valeurs Nulles. */
#define GSS_C_NO_NAME ((gss_name_t) 0)
#define GSS_C_NO_BUFFER ((gss_buffer_t) 0)
#define GSS_C_NO_OID ((gss_OID) 0)
#define GSS_C_NO_OID_SET ((gss_OID_set) 0)
#define GSS_C_NO_CONTEXT ((gss_ctx_id_t) 0)
#define GSS_C_NO_CREDENTIAL ((gss_cred_id_t) 0)
#define GSS_C_NO_CHANNEL_BINDINGS ((gss_channel_bindings_t) 0)
#define GSS_C_EMPTY_BUFFER {0, NULL}

/* Noms de remplacement pour deux des valeurs ci-dessus. Elles sont définies pour la compatibilité avec la version 1. */
#define GSS_C_NULL_OID GSS_C_NO_OID
#define GSS_C_NULL_OID_SET GSS_C_NO_OID_SET

/* Définir la qualité de protection par défaut pour les services par message. Noter qu'une mise en œuvre qui offre plusieurs
   niveaux de QOP peut définir GSS_C_QOP_DEFAULT comme étant soit zéro (comme ici) pour signifier "protection par
   défaut", soit une valeur spécifique de QOP explicite. Cependant, une valeur de 0 devrait toujours être interprétée par
   une mise en œuvre GSS-API comme la demande du niveau de protection par défaut. */
#define GSS_C_QOP_DEFAULT 0

/* Un temps d'expiration de 2^32-1 secondes signifie une durée de vie infinie pour un accréditif ou contexte de sécurité. */
#define GSS_C_INDEFINITE 0xfffffffful

/* La mise en œuvre doit réserver de la mémorisation statique pour un objet gss_OID_desc contenant la valeur {10, (void
   *)"\x2a\x86\x48\x86\xf7\x12" "\x01\x02\x01\x01"}, correspondant à une valeur d'identifiant d'objet de {iso(1)
   member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) user_name(1)}. La constante
   GSS_C_NT_USER_NAME devrait être initialisée pour pointer sur ce gss_OID_desc. */
extern gss_OID GSS_C_NT_USER_NAME;

/* La mise en œuvre doit réserver de la mémorisation statique pour un objet gss_OID_desc contenant la valeur {10, (void
   *)"\x2a\x86\x48\x86\xf7\x12" "\x01\x02\x01\x02"}, correspondant à une valeur d'identifiant d'objet de {iso(1)
   member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) machine_uid_name(2)}. La constante
   GSS_C_NT_MACHINE_UID_NAME devrait être initialisée pour pointer sur ce gss_OID_desc. */
extern gss_OID GSS_C_NT_MACHINE_UID_NAME;

/*  La mise en œuvre doit réserver une mémorisation statique pour un objet gss_OID_desc contenant la valeur {10, (void
   *)"\x2a\x86\x48\x86\xf7\x12" "\x01\x02\x01\x03"}, correspondant à une valeur d'identifiant d'objet {iso(1) member-
   body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) string_uid_name(3)}. La constante
   GSS_C_NT_STRING_UID_NAME devrait être initialisée de façon à pointer sur gss_OID_desc. */
extern gss_OID GSS_C_NT_STRING_UID_NAME;

/* La mise en œuvre doit réserver une mémorisation statique pour un objet gss_OID_desc contenant la valeur {6, (void
   *)"\x2b\x06\x01\x05\x06\x02"}, correspondant à une valeur d'identifiant d'objet de {iso(1) org(3) dod(6) internet(1)
   security(5) nametypes(6) gss-host-based-services(2)). La constante GSS_C_NT_HOSTBASED_SERVICE_X devrait
   être initialisée pour pointer sur gss_OID_desc. C'est une valeur d'OID déconseillée, et les mises en œuvre qui
   souhaitent prendre en charge des noms de service fondés sur l'hôte devraient plutôt utiliser l'OID
   GSS_C_NT_HOSTBASED_SERVICE, défini ci-dessous, pour identifier de tels noms;
   GSS_C_NT_HOSTBASED_SERVICE_X devrait être accepté comme synonyme pour
   GSS_C_NT_HOSTBASED_SERVICE lorsque présenté comme paramètre d'entrée, mais ne devrait pas être émis par
   les mises en œuvre GSS-API. */
extern gss_OID GSS_C_NT_HOSTBASED_SERVICE_X;

/* La mise en œuvre doit réserver une mémorisation statique pour un objet gss_OID_desc contenant la valeur {10, (void
   *)"\x2a\x86\x48\x86\xf7\x12" "\x01\x02\x01\x04"}, correspondant à une valeur d'identifiant d'objet de {iso(1)
   member-body(2) Unites States(840) mit(113554) infosys(1) gssapi(2) generic(1) service_name(4)}. La constante
```

```
    GSS_C_NT_HOSTBASED_SERVICE devrait être initialisée pour pointer sur ce gss_OID_desc. */
extern gss_OID GSS_C_NT_HOSTBASED_SERVICE;

/* La mise en œuvre doit réserver une mémorisation statique pour un objet gss_OID_desc contenant la valeur {6, (void
    *)"\x2b\x06\01\x05\x06\x03"}, correspondant à une valeur d'identifiant d'objet de {1(iso), 3(org), 6(dod), 1(internet),
    5(security), 6(nametypes), 3(gss-anonymous-name)}. La constante et GSS_C_NT_ANONYMOUS devraient être
    initialisée pour pointer sur ce gss_OID_desc. */
extern gss_OID GSS_C_NT_ANONYMOUS;

/* La mise en œuvre doit réserver une mémorisation statique pour un objet gss_OID_desc contenant la valeur {6, (void
    *)"\x2b\x06\x01\x05\x06\x04"}, correspondant à une valeur d'identifiant d'objet de {1(iso), 3(org), 6(dod), 1(internet),
    5(security), 6(nametypes), 4(gss-api-exported-name)}. La constante GSS_C_NT_EXPORT_NAME devrait être
    initialisée pour pointer sur ce gss_OID_desc. */
extern gss_OID GSS_C_NT_EXPORT_NAME;

/* Codes d'état majeur. */

#define GSS_S_COMPLETE 0

/* Quelques définitions "d'aide" pour rendre évidentes les macros de code d'état. */
#define GSS_C_CALLING_ERROR_OFFSET      24
#define GSS_C_ROUTINE_ERROR_OFFSET      16
#define GSS_C_SUPPLEMENTARY_OFFSET      0
#define GSS_C_CALLING_ERROR_MASK        0377ul
#define GSS_C_ROUTINE_ERROR_MASK        0377ul
#define GSS_C_SUPPLEMENTARY_MASK        0177777ul

/* Les macros qui vérifient les codes d'état pour chercher les conditions d'erreur. Noter que la macro GSS_ERROR() a
    légèrement changé depuis la GSS-API version 1 de sorte qu'elle évalue maintenant seulement une fois son argument. */
#define GSS_CALLING_ERROR(x) \ (x & (GSS_C_CALLING_ERROR_MASK <<
                                 GSS_C_CALLING_ERROR_OFFSET))
#define GSS_ROUTINE_ERROR(x) \ (x & (GSS_C_ROUTINE_ERROR_MASK <<
                                 GSS_C_ROUTINE_ERROR_OFFSET))
#define GSS_SUPPLEMENTARY_INFO(x) \ (x & (GSS_C_SUPPLEMENTARY_MASK <<
                                 GSS_C_SUPPLEMENTARY_OFFSET))
#define GSS_ERROR(x) \ (x & ((GSS_C_CALLING_ERROR_MASK << GSS_C_CALLING_ERROR_OFFSET) | \
                                 (GSS_C_ROUTINE_ERROR_MASK << GSS_C_ROUTINE_ERROR_OFFSET)))

/* Maintenant, les définitions de code d'état réelles. */

/* Erreurs d'appel : */
#define GSS_S_CALL_INACCESSIBLE_READ \ (1ul << GSS_C_CALLING_ERROR_OFFSET)
#define GSS_S_CALL_INACCESSIBLE_WRITE \ (2ul << GSS_C_CALLING_ERROR_OFFSET)
#define GSS_S_CALL_BAD_STRUCTURE \ (3ul << GSS_C_CALLING_ERROR_OFFSET)

/* Erreurs de sous-programme : */
#define GSS_S_BAD_MECH                  (1ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_BAD_NAME                  (2ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_BAD_NAMETYPE              (3ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_BAD_BINDINGS              (4ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_BAD_STATUS                (5ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_BAD_SIG                   (6ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_BAD_MIC GSS_S_BAD_SIG
#define GSS_S_NO_CRED                   (7ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_NO_CONTEXT                (8ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_DEFECTIVE_TOKEN           (9ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_DEFECTIVE_CREDENTIAL      (10ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_CREDENTIALS_EXPIRED       (11ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_CONTEXT_EXPIRED           (12ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_FAILURE                   (13ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_BAD_QOP                   (14ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_UNAUTHORIZED              (15ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_UNAVAILABLE               (16ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

```
#define GSS_S_DUPLICATE_ELEMENT         (17ul << GSS_C_ROUTINE_ERROR_OFFSET)
#define GSS_S_NAME_NOT_MN               (18ul << GSS_C_ROUTINE_ERROR_OFFSET)

/* Bits d'informations supplémentaires : */
#define GSS_S_CONTINUE_NEEDED \ (1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 0))
#define GSS_S_DUPLICATE_TOKEN \ (1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 1))
#define GSS_S_OLD_TOKEN \ (1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 2))
#define GSS_S_UNSEQ_TOKEN \ (1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 3))
#define GSS_S_GAP_TOKEN \ (1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 4))

/* Finalement, les prototypes de fonctions pour les sous-programmes GSS-API. */

OM_uint32 gss_acquire_cred
        (OM_uint32 ,            /*  minor_status */
         const gss_name_t,      /* desired_name */
         OM_uint32,             /* time_req */
         const gss_OID_set,     /* desired_mechs */
         gss_cred_usage_t,      /* cred_usage */
         gss_cred_id_t ,        /* output_cred_handle */
         gss_OID_set ,          /* actual_mechs */
         OM_uint32 *            /* time_rec */
        );

OM_uint32       gss_release_cred
        (OM_uint32 ,            /* minor_status */
         gss_cred_id_t *        /* cred_handle */
        );

OM_uint32       gss_init_sec_context
        (OM_uint32 ,                    /* minor_status */
         const gss_cred_id_t,           /* initiator_cred_handle */
         gss_ctx_id_t ,                 /* context_handle */
         const gss_name_t,              /* target_name */
         const gss_OID,                 /* mech_type */
         OM_uint32,                     /* req_flags */
         OM_uint32,                     /* time_req */
         const gss_channel_bindings_t,  /* input_chan_bindings */
         const gss_buffer_t,            /* input_token */
         gss_OID ,                      /* actual_mech_type */
         gss_buffer_t,                  /* output_token */
         OM_uint32 ,                    /* ret_flags */
         OM_uint32 *                    /* time_rec */
        );

OM_uint32       gss_accept_sec_context
        (OM_uint32 ,                    /* minor_status */
         gss_ctx_id_t ,                 /* context_handle */
         const gss_cred_id_t,           /* acceptor_cred_handle */
         const gss_buffer_t,            /* input_token_buffer */
         const gss_channel_bindings_t,  /* input_chan_bindings */
         gss_name_t ,                   /* src_name */
         gss_OID ,                      /* mech_type */
         gss_buffer_t,                  /* output_token */
         OM_uint32 ,                    /* ret_flags */
         OM_uint32 ,                    /* time_rec */
         gss_cred_id_t *                /* delegated_cred_handle */
        );

OM_uint32       gss_process_context_token
        (OM_uint32 ,            /* minor_status */
         const gss_ctx_id_t,    /* context_handle */
         const gss_buffer_t     /* token_buffer */
        );
```

```
OM_uint32       gss_delete_sec_context
        (OM_uint32 ,            /* minor_status */
         gss_ctx_id_t ,         /* context_handle */
         gss_buffer_t           /* output_token */
        );

OM_uint32       gss_context_time
        (OM_uint32 ,            /* minor_status */
         const gss_ctx_id_t,    /* context_handle */
         OM_uint32 *            /* time_rec */
        );

OM_uint32       gss_get_mic
        (OM_uint32 ,            /* minor_status */
         const gss_ctx_id_t,    /* context_handle */
         gss_qop_t,             /* qop_req */
         const gss_buffer_t,    /* message_buffer */
         gss_buffer_t           /* message_token */
        );

OM_uint32       gss_verify_mic
        (OM_uint32 ,            /* minor_status */
         const gss_ctx_id_t,    /* context_handle */
         const gss_buffer_t,    /* message_buffer */
         const gss_buffer_t,    /* token_buffer */
         gss_qop_t *            /* qop_state */
        );

OM_uint32       gss_wrap
        (OM_uint32 ,            /* minor_status */
         const gss_ctx_id_t,    /* context_handle */
         int,                   /* conf_req_flag */         gss_qop_t,          /* qop_req */
         const gss_buffer_t,    /* input_message_buffer */
         int ,                  /* conf_state */
         gss_buffer_t           /* output_message_buffer */
        );

OM_uint32       gss_unwrap
        (OM_uint32 ,            /* minor_status */
         const gss_ctx_id_t,    /* context_handle */
         const gss_buffer_t,    /* input_message_buffer */
         gss_buffer_t,          /* output_message_buffer */
         int ,                  /* conf_state */
         gss_qop_t *            /* qop_state */
        );

OM_uint32       gss_display_status
        (OM_uint32 ,            /* minor_status */
         OM_uint32,             /* status_value */
         int,                   /* status_type */
         const gss_OID,         /* mech_type */
         OM_uint32 ,            /* message_context */
         gss_buffer_t           /* status_string */
        );

OM_uint32       gss_indicate_mechs
        (OM_uint32 ,            /* minor_status */
         gss_OID_set *          /* mech_set */
        );

OM_uint32       gss_compare_name
        (OM_uint32 ,            /* minor_status */
```

```
            const gss_name_t,     /* name1 */
            const gss_name_t,     /* name2 */
            int *                 /* name_equal */
           );

OM_uint32       gss_display_name
           (OM_uint32 ,           /* minor_status */
            const gss_name_t,     /* input_name */
            gss_buffer_t,         /* output_name_buffer */
            gss_OID *             /* output_name_type */
           );

OM_uint32       gss_import_name
             (OM_uint32 ,         /* minor_status */
              const gss_buffer_t, /* input_name_buffer */
              const gss_OID,      /* input_name_type */
              gss_name_t *        /* output_name */
             );

OM_uint32       gss_export_name
           (OM_uint32,           /* minor_status */
            const gss_name_t,    /* input_name */
            gss_buffer_t         /* exported_name */
           );

OM_uint32       gss_release_name
           (OM_uint32 *,         /* minor_status */
            gss_name_t *         /* input_name */
           );

OM_uint32       gss_release_buffer
           (OM_uint32 ,          /* minor_status */
            gss_buffer_t         /* buffer */
           );

OM_uint32       gss_release_oid_set
           (OM_uint32 ,          /* minor_status */
            gss_OID_set *        /* set */
           );

OM_uint32       gss_inquire_cred
           (OM_uint32 ,          /* minor_status */
            const gss_cred_id_t, /* cred_handle */
            gss_name_t ,         /* name */
            OM_uint32 ,          /* lifetime */
            gss_cred_usage_t ,   /* cred_usage */
            gss_OID_set *        /* mechanisms */
           );

OM_uint32       gss_inquire_context (
            OM_uint32 ,          /* minor_status */
            const gss_ctx_id_t,  /* context_handle */
            gss_name_t ,         /* src_name */
            gss_name_t ,         /* targ_name */
            OM_uint32 ,          /* lifetime_rec */
            gss_OID ,            /* mech_type */
            OM_uint32 ,          /* ctx_flags */
            int ,                /* locally_initiated */
            int *                /* open */
           );

OM_uint32       gss_wrap_size_limit (
            OM_uint32 ,          /* minor_status */
```

```
            const gss_ctx_id_t,     /* context_handle */
            int,                    /* conf_req_flag */
            gss_qop_t,              /* qop_req */
            OM_uint32,              /* req_output_size */
            OM_uint32 *             /* max_input_size */
           );

OM_uint32   gss_add_cred (
            OM_uint32 ,             /* minor_status */
            const gss_cred_id_t,    /* input_cred_handle */
            const gss_name_t,       /* desired_name */
            const gss_OID,          /* desired_mech */
            gss_cred_usage_t,       /* cred_usage */
            OM_uint32,              /* initiator_time_req */
            OM_uint32,              /* acceptor_time_req */
            gss_cred_id_t ,         /* output_cred_handle */
            gss_OID_set ,           /* actual_mechs */
            OM_uint32 ,             /* initiator_time_rec */
            OM_uint32 *             /* acceptor_time_rec */
           );

OM_uint32   gss_inquire_cred_by_mech (
            OM_uint32 ,             /* minor_status */
            const gss_cred_id_t,    /* cred_handle */
            const gss_OID,          /* mech_type */
            gss_name_t ,            /* name */
            OM_uint32 ,             /* initiator_lifetime */
            OM_uint32 ,             /* acceptor_lifetime */
            gss_cred_usage_t *      /* cred_usage */
           );

OM_uint32   gss_export_sec_context (
            OM_uint32 ,             /* minor_status */
            gss_ctx_id_t ,          /* context_handle */
            gss_buffer_t            /* interprocess_token */
           );

OM_uint32   gss_import_sec_context (
            OM_uint32 ,             /* minor_status */
            const gss_buffer_t,     /* interprocess_token */
            gss_ctx_id_t *          /* context_handle */
           );

OM_uint32   gss_create_empty_oid_set (
            OM_uint32 ,             /* minor_status */
            gss_OID_set *           /* oid_set */
           );

OM_uint32   gss_add_oid_set_member (
            OM_uint32 ,             /* minor_status */
            const gss_OID,          /* member_oid */
            gss_OID_set *           /* oid_set */
           );

OM_uint32   gss_test_oid_set_member (
            OM_uint32 ,             /* minor_status */
            const gss_OID,          /* member */
            const gss_OID_set,      /* set */
            int *                   /* present */
           );

OM_uint32   gss_inquire_names_for_mech (
            OM_uint32 ,             /* minor_status */
```

```
        const gss_OID,          /* mechanism */
        gss_OID_set *           /* name_types */
       );

OM_uint32       gss_inquire_mechs_for_name (
        OM_uint32 ,             /* minor_status */
        const gss_name_t,       /* input_name */
        gss_OID_set *           /* mech_types */
       );

OM_uint32       gss_canonicalize_name (
        OM_uint32 ,             /* minor_status */
        const gss_name_t,       /* input_name */
        const gss_OID,          /* mech_type */
        gss_name_t *            /* output_name */
       );

OM_uint32       gss_duplicate_name (
        OM_uint32 ,             /* minor_status */
        const gss_name_t,       /* src_name */
        gss_name_t *            /* dest_name */
       );
```

/* Les sous-programmes suivants sont des variantes obsolètes de gss_get_mic, gss_verify_mic, gss_wrap et gss_unwrap. Ils devraient être fournis par les mises en œuvre GSS-API v2 pour la rétro compatibilité avec les applications de la version 1. Des points d'entrée distincts (par opposition à #defines) devraient être fournis, à la fois pour permettre aux applications GSS-API v1 de se relier aux mises en œuvre GSS-API v2 , et pour conserver les légères différences de type de paramètre entre les versions obsolètes de ces sous-programmes et leur forme actuelle. */

```
OM_uint32       gss_sign
       (OM_uint32 ,             /* minor_status */
        gss_ctx_id_t,           /* context_handle */
        int,                    /* qop_req */
        gss_buffer_t,           /* message_buffer */
        gss_buffer_t            /* message_token */
       );

OM_uint32       gss_verify
       (OM_uint32 ,             /* minor_status */
        gss_ctx_id_t,           /* context_handle */
        gss_buffer_t,           /* message_buffer */
        gss_buffer_t,           /* token_buffer */
        int *                   /* qop_state */
       );

OM_uint32       gss_seal
       (OM_uint32 ,             /* minor_status */
        gss_ctx_id_t,           /* context_handle */
        int,                    /* conf_req_flag */
        int,                    /* qop_req */
        gss_buffer_t,           /* input_message_buffer */
        int ,                   /* conf_state */
        gss_buffer_t            /* output_message_buffer */
       );

OM_uint32       gss_unseal
       (OM_uint32 ,             /* minor_status */
        gss_ctx_id_t,           /* context_handle */
        gss_buffer_t,           /* input_message_buffer */
        gss_buffer_t,           /* output_message_buffer */
        int ,                   /* conf_state */
        int *                   /* qop_state */
       );
```

```
#endif          /* GSSAPI_H_ */
```

Appendice B Contraintes supplémentaires pour la portabilité binaire d'application

L'objet de ce document sur les liaisons C est d'encourager la portabilité au niveau de la source des applications à travers les mises en œuvre GSS-API sur différentes plateformes et par dessus différents mécanismes. Des objectifs supplémentaires qui n'ont pas été explicitement traités par le présent document sont la portabilité au moment de la liaison et au moment du lancement.

La portabilité au moment de la liaison donne la capacité de compiler une application à l'égard d'une mise en œuvre de GSS-API, et ensuite de la lier avec une mise en œuvre différente sur la même plateforme. Elle a des exigences plus strictes que la portabilité au niveau de la source.

La portabilité au moment du lancement ne diffère de la portabilité au moment de la liaison que sur les plateformes qui mettent en œuvre des GSS-API à chargement dynamique, mais n'offrent pas de résolution de symbole au moment du chargement. Sur de telles plateformes, la portabilité au moment du lancement est une exigence plus stricte que la portabilité au moment de la liaison, et va normalement inclure le placement précis des divers sous-programmes GSS-API au sein des vecteurs de point d'entrée de la bibliothèque.

Les plateformes individuelles imposent leurs propres règles qui doivent être suivies pour réaliser la portabilité au moment de la liaison (et du lancement, si il est différent). Afin d'assurer les deux formes de portabilité binaire, une spécification d'ABI doit être rédigée pour les mises en œuvre GSS-API sur cette plateforme. Cependant, on reconnaît qu'il y a quelques problèmes qui vont vraisemblablement être communs à toutes ces spécifications d'ABI. Le présent appendice est destiné à collecter ces problèmes communs, et à contenir des suggestions auxquelles les spécifications individuelles d'ABI peuvent choisir de se référer. Comme les architectures de machines varient largement, il se peut qu'il ne soit pas possible ou désirable de suivre ces suggestions sur toutes les plateformes.

B.1 Pointeurs

Bien que l'ANSI-C fournisse un seul type de pointeur pour chaque type déclaré, plus un seul type (void *), certaines plateformes (notamment celles qui utilisent des architectures de mémoire segmentée) augmentent cela avec divers types de pointeur modifié (par exemple, pointeurs lointains, pointeurs proches). Ces liens de langage supposent ANSI-C, et donc ne visent pas de telles mises en œuvre non standard. Les mises en œuvre GSS-API pour de telles plateformes doivent choisir un modèle de mémoire approprié, et devraient l'utiliser tout le temps de façon cohérente. Par exemple, si un modèle de mémoire choisi exige l'utilisation de pointeurs lointains lors du passage de paramètres de sous-programme, des pointeurs lointains devraient alors aussi être utilisés au sein des structures définies par GSS-API.

B.2 Alignement de structure interne

GSS-API définit plusieurs structures de données qui contiennent des champs de tailles différentes. Une spécification ABI devrait inclure une description détaillée de la façon dont les champs de telles structures sont alignés, et si il y a du bourrage interne dans ces structures de données. On recommande l'utilisation d'un compilateur par défaut pour la plateforme.

B.3 Types de liens

Les liens C spécifient que les types gss_cred_id_t et gss_ctx_id_t devraient être mis en œuvre comme des types pointeur ou arithmétique, et que si le type pointeur est utilisé, on devrait prendre soin de s'assurer que deux liens peuvent être comparés avec l'opérateur ==. Noter que ANSI-C ne garantit pas que deux valeurs de pointeur peuvent être comparées avec l'opérateur == sauf si les deux pointeurs pointent sur les membres d'une seule matrice, ou si au moins un des pointeurs contient une valeur NUL.

Pour la portabilité binaire, des contraintes supplémentaires sont requises. Nous allons tenter ci-après de définir des contraintes indépendantes de la plateforme.

La taille du type de lien doit être la même que celle de sizeof(void *), en utilisant le modèle de mémoire approprié.

L'opérateur == pour le type choisi doit être une simple comparaison au bit près. C'est-à-dire que pour deux objets de lien en mémoire h1 et h2, la valeur booléenne de l'expression (h1 == h2) devrait toujours être la même que la valeur booléenne de l'expression (memcmp(&h1, &h2, sizeof(h1)) == 0).

L'utilisation du type (void *) pour les types de liens est déconseillée, non pour des raisons de portabilité binaire, mais parce qu'elle désactive effectivement beaucoup des vérifications de type pendant la compilation que pourrait autrement effectuer le compilateur, et elle n'est donc pas "neutre pour le programmeur". Si une mise en œuvre de pointeur est désirée, et si la mise en œuvre de pointeurs de la plateforme le permet, les liens devraient être mis en œuvre comme pointeurs pour distinguer les types définis par la mise en œuvre.

B.4 Type gss_name_t

Le type gss_name_t, qui représente l'objet Nom interne, devrait être mis en œuvre comme un type de pointeur. L'utilisation du type (void *) est déconseillée car elle ne permet pas au compilateur d'effectuer une vérification forte du type. Cependant, le type de pointeur choisi devrait être de la même taille que le type (void *). Pourvu que cette règle soit observée, les spécifications d'ABI n'ont pas besoin d'autres contraintes sur la mise en œuvre des objets gss_name_t.

B.5 Type int et size_t

Certaines plateformes peuvent prendre en charge des mises en œuvre des types "int" et "size_t" de différentes tailles, choisies peut-être par des commutations de compilateur, et peut-être selon le modèle de mémoire. Une spécification d'ABI pour une telle plateforme devrait inclure les mises en œuvre requises pour ces types. Il est recommandé que soit choisie la mise en œuvre par défaut (pour le modèle de mémoire choisi, si c'est approprié).

B.6 Conventions de procédure d'invocation

Certaines plateformes prennent en charge un certain nombre de conventions binaires différentes pour les procédures d'invocation. De telles conventions couvrent des choses comme le format de trame de pile, l'ordre dans lequel les paramètres du sous-programme sont poussés sur la pile, si un compte de paramètres est poussé sur la pile, si un ou des arguments ou valeurs de retour sont à passer dans les registres, et si le sous-programme invoqué ou l'appelant est responsable du retrait de la trame de pile en retour. Pour de telles plateformes, une spécification d'ABI devrait spécifier quelle convention d'invocation est à utiliser par les mises en œuvre GSS-API.

Références

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

[XOM] OSI Object Management API Specification, Version 2.0 t", X.400 API Association & X/Open Company Limited, 24 août 1990 "Specification of datatypes and routines for manipulating information objects".

Adresse de l'auteur

John Wray
Iris Associates
5 Technology Park Drive,
Westford, MA 01886
USA
téléphone +1-978-392-6689
mél : John_Wray@Iris.com

Déclaration complète de droits de reproduction

Remerciement

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