GWD: draft-ggf-gss-extensions-07
Grid Security Infrastructure (GSI) WG
http://www.gridforum.org/2_SEC/GSI.htm

S. Meder
V. Welch
U. Chicago
S. Tuecke
D. Engert
ANL

February 2001
Revised August 2002

# GSS-API Extensions

## Status of this Memo

This document provides information to the community regarding extensions to GSS-API as defined in RFC 2743 and RFC 2744.

This document is a product of the Grid Security Infrastructure (GSI) Working Group of the Global Grid Forum. The latest version of this document is available from the GSI WG webpage:

http://www.gridforum.org/2_SEC/GSI.htm

Distribution of this document is unlimited.

## Copyright



## Abstract

This document defines extensions to RFC 2743, Generic Security Service Application Program Interface Version 2, Update 1. Extensions include: credential export and import; delegation at any time; credential extensions (e.g. restrictions) handling.

meder@mcs.anl.gov

# Table of Contents

# 1 Introduction

RFC 2743 defines the Generic Security Service Application Program Interface (GSS-API) Version 2, Update 1 [3, 4], as an API for portably adding authentication, delegation, and message protection to distributed computing applications.

In 1997, the Globus Project (www.globus.org) introduced an implementation of GSS-API called the Grid Security Infrastructure (GSI). This implementation uses public key protocols for use in programming Grid applications [2] -- that is, applications that run in dynamic, inter-domain distributed computing environments. Based on this implementation, a great deal of experience has been gained on the use of GSS-API in numerous real applications and middleware toolkits. While this experience has been overwhelmingly positive, it has also led to an understanding of some deficiencies in the existing GSS-API.

This document defines extensions to the GSS-API to address these deficiencies. These extensions are:

- *Credential export and import*: Processes need to be able, in a controlled and standard fashion, to export credentials to and import credentials from other processes. This includes processes that are not written to the GSS-API.

- *Delegation at any time*: GSS-API only allows for delegation during the initial context establishment, via an argument to gss_init_sec_context. This document extends GSS-API to also allow for delegation at any time after initial context establishment.

- *Credential extensions handling*: When delegating a credential, it is often useful to attach additional data, such as restriction policies, to that delegated credential which restricts its usage. This document defines extensions to the GSS-API to allow such extensions to be specified during delegation and to be extracted from a security context after authentication. However, the approach is neutral to the actual attached data.

Section 2 defines the GSS-API extensions.

Section 3 contains changes to existing GSS-API functions.

Section 4 contains discussions of problems with the existing GSS-API that the authors have found, but have not had time to work out solutions for.

Section 5 contains security considerations.

Section 6 contains the references.

Section 7 contains acknowledgements.

Section 8 contains contact information for the authors.

Section 9 contains C-bindings for the API extensions.

Section 10 contains the change log for this document.

Section 11 contains the copyright information for this document.

Section 12 contains the intellectual property information for this document.

This document was written under the auspices of the Global Grid Forum Grid Security Infrastructure Working Group. For more information on this and other related work, see http://www.gridforum.org/2_SEC/GSI.htm.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

# 2 Extension Specifications

The following sections explain needed extensions to the GSS-API as defined in RFC 2743 and give proposed additional functions to be added to the API.

## 2.1 Credential Export and Import

In order to portably implement a reliable service that can accept delegated credentials from multiple clients, these functions are needed to allow the service to checkpoint and reload the delegated credential to and from permanent storage.

These functions also provide support for servers that need to pass a credential on to other processes, for example a sshd daemon that accepts a delegated credential from a client and then wants to make that credential available to the spawned shell process.

Previously applications which needed this functionality were forced to use mechanism specific lower level routines, defeating the purpose of a generic interface.

The gss_export_cred function allows exporting credentials in one of two forms as selected by the option_req parameter.

When option_req is equal to 0, it exports an opaque buffer suitable for storage in memory or on disk or passing to another process, which can import the buffer with gss_import_cred. This method of invocation will normally be used by applications that want to transfer a credential to another GSS application or save a credential to some storage medium so it can be loaded later (perhaps after a restart).

With option_req is equal to 1, it exports a buffer filled with mechanism-specific information that the calling application can use to pass the credential to another process that is not written to the GSS-API. This is intended to work with the existing Kerberos and GSI GSS implementations. These implementation normally store credentials by writing the credentials to a file and then storing the path to that file in an environment variable. In the case of these implementations this function stores the credential on disk and returns a buffer containing a string suitable for passing to putenv(3). This method of invocation will normally be used by applications, such as SSHD, which accept a delegated GSS credential and then want to make this credential available to the shell process they spawn.

Note that in either case, you cannot pass the exported credential to another machine over the network and assume it will work. Passing a credential to another process may also fail if the process does not have the same privileges as the original process.

A credential can also expire between export and import. In this case the import of the credential may fail, returning an error.

The gss_import_cred function is the complimentary function to gss_export_cred. When called with the exported buffer and option_req set to the same value used in the gss_export_cred call it will return a credential handle suitable for use with other GSS-API functions. While this function is similar in function to gss_acquire_cred, there is no compatibility between the two (i.e. one cannot take a buffer from gss_export_cred and use gss_acquire_cred to import it).

### 2.1.1 gss_export_cred call

Inputs:

- option_req INTEGER – 0 = Export simple data good only for reuse with gss_import_cred, 1 = Export mechanism-specific data that can be passed to non-GSS applications (see following function description).

- cred_handle CREDENTIAL HANDLE – credential to be exported. May be modified but is stable and usable after call.

- desired_mech OBJECT IDENTIFIER – desired mechanism for exported credential, may be NULL to indicate system default

- protection_key OPTIONAL OCTET STRING – if specified, a key used to protect (e.g. encrypt) the exported credential.

Outputs:

- major_status INTEGER

- minor_status INTEGER

- export_buffer OCTET STRING – token to be used in future gss_import_cred call (if option_req == 0) or mechanism specific data (if option_req == 1). To be freed by caller using gss_release_buffer.

- actual_mech OBJECT_IDENTIFIER – actual mechanism of the exported credential

Returned major_status codes:

- GSS_S_COMPLETE indicates that the operation was successfully completed and export_buffer contains the requested data.

- GSS_S_CREDENTIALS_EXPIRED indicates the credential referred to by cred_handle has expired and cannot be exported.

- GSS_S_UNAVAILABLE indicates that the requested operation is not supported by the underlying mechanism.

- GSS_S_BAD_MECH indicates that the requested mechanism is unsupported.

- GSS_S_FAILURE indicates a failure unspecified at the GSS-API level.

GSS_export_cred is used to export a credential so that it can be passed to another process or reloaded by the same process after its use of a different credential in the interim.

If the caller passes in a value of 0 for option_req the returned data will be suitable for use in a call to gss_import_cred at a later date or in a different process. Note that the data may contain the actual credential itself or just a pointer to it (e.g. a filename). In any case it should be assumed that the returned buffer contains sensitive information and care should be taken to protect the privacy of its contents.

If the caller passes in a value of 1 for option_req the return data contains mechanism-specific data that the calling application can use to pass the credential to another process that does not use the GSS-API. The caller is responsible for understanding this data and knowing what to do with it. For example in the case of both Kerberos 5 and GSI the credential will be stored to a file and the returned data will be a string suitable for passing to putenv(3). In this case the caller may need to change the ownership of the credential file.

In either case the data is returned in export_buffer, which the calling application is responsible for freeing using gss_release_buffer.

If the protection_key is provided by the caller, the underlying mechanism MUST use this key to protect the exported key (probably by encryption but this is mechanism-specific). If a

protection_key is provided and protection is not supported by the underlying mechanism a GSS_S_UNAVAILABLE error MUST be returned.

This function is intended to be used primarily by complex servers that must juggle multiple credentials, storing them for a time in files and/or passing them to other processes.

### 2.1.2 gss_import_cred

Inputs:

- option_req INTEGER – 0 = import_buffer is opaque data as created by gss_export_cred called with option_req = 0, 1 = import_buffer is handle to a credential as created by gss_export_cred called with option_req = 1 (e.g. strings of the form "KRB5CCNAME=FILE:filename").

- desired_mech OBJECT IDENTIFIER – desired mechanism for the imported credential, may be NULL to indicate system default.

- import_buffer OCTET STRING – buffer containing data for import as created by a call to gss_export_cred.

- lifetime_req INTEGER – in seconds, 0 requests default.

- protection_key OPTIONAL OCTET STRING – if specified, a key used to access the imported credential.

Outputs:

- major_status INTEGER

- minor_status INTEGER

- cred_handle CREDENTIAL HANDLE – the returned credential handle. Resources associated with the credential handle must be released by the application after use with a call to gss_release_cred.

- lifetime_rec INTEGER – in seconds, with reserved value for INDEFINITE.

- actual_mech OBJECT_IDENTIFIER – actual mechanism of the imported credential.

Returned major_status codes:

- GSS_S_COMPLETE indicates that the operation was successful and cred_handle contains a handle to a usable credential.

- GSS_S_BAD_MECH indicates that the requested mechanism is unsupported.

- GSS_S_DEFECTIVE_TOKEN indicates the import buffer is unparsable.

- GSS_S_NO_CRED indicates that the imported credential is unusable or inaccessible.

- GSS_S_CREDENTIALS_EXPIRED indicates that the imported credential has expired.
- GSS_S_FAILURE indicates an error unspecified at the GSS-API level.

Gss_import_cred is the complimentary function to gss_export_cred and is intended to be used to allow use of a credential that was passed from another process or read from storage.

The credential, or the handle to the credential, is read from import_buffer, which is unmodified. A handle to the credential is returned in cred_handle.

The lifetime_rec result indicates the length of time for which the acquired credential will be valid, as an offset from the present. A mechanism may return a reserved value indicating INDEFINITE if no constraints on credential lifetime are imposed. A caller of gss_import_cred can request a length of time for which acquired the credential is to be valid (using the lifetime_req parameter), beginning at the present, or can request a credential with a default validity interval. Requests for postdated credentials are not supported within the GSS-API. Certain mechanisms and implementations may supply credential validity period specifiers at a point prior to invocation of the gss_import_cred call (e.g., in conjunction with user login procedures). As a result, callers requesting non-default values for lifetime_req must recognize that such requests cannot always be honored and must be prepared to accommodate the return of a credential with a different lifetime as indicated by lifetime_rec. The lifetime_req parameter should be viewed as advisory and no error should be returned if the lifetime request cannot be honored.

The protection_key parameters should be used to provide a key to the underlying mechanism to be used to access the credentials in the event they are protected. If the credentials are not protected and a key is provided it SHOULD be ignored. If the credentials are protected and no key is provided a GSS_S_NO_CRED error MUST be returned.

## 2.2 Delegation At Any Time

These functions are designed to allow more flexible protocols to be built on top of the GSS-API. They allow delegation to happen at times other than context initiation, in either direction regardless of the direction of context initiation (i.e. the acceptor in context initiation can be the delegator) and with a credential other than the credential used for context setup.

These functions also allow the initiator and acceptor to attach extensions, such as restriction polices, to a delegated credential. These extensions are mechanism-specific and completely opaque to the GSS-API. See the following section on Credential Extension Handling for information about accessing the extension data.

Furthermore, these functions make it possible to delegate credentials associated with mechanisms other than the mechanism used for context establishment. Application may for example wish to delegate a Kerberos credential over a security context established using GSI.

The two functions for doing delegation are gss_init_delegation and gss_accept_delegation. These functions are used in a similar manner as gss_init_sec_context and gss_accept_sec_context; they should be called in a loop as long as they return GSS_S_CONTINUE_NEEDED and the tokens output by each should be passed into the other.

Note that mechanisms are not required to support intermingling of calls to other GSS-API functions (e.g. gss_wrap) with the sequence of calls to needed to perform delegation of a credential. This also implies that if an application initiates a second delegation while a previous one has yet to complete the results may be undefined.

### 2.2.1 gss_init_delegation

Inputs:

- context_handle CONTEXT HANDLE – handle to existing security context.
- cred_handle CREDENTIAL HANDLE – handle to credential to be delegated. NULL (GSS_C_NO_CREDENTIAL) specifies the use of the default credential.
- mech_type OBJECT IDENTIFIER – NULL parameter specifies that the system default should be used.
- lifetime_req INTEGER - 0 specifies default lifetime
- extension_oids SEQUENCE OF OBJECT IDENTIFIER – identifiers for the type of data in the extensions_buffers. See description below for a discussion of this option.
- extension_buffers SEQUENCE OF OCTET STRINGS – opaque extension data to be applied to the delegated credential. See description below for a discussion of this option.
- input_token OCTET STRING – GCC_C_NO_BUFFER or token received from target.

Outputs:

- major_status INTEGER
- minor_status INTEGER
- output_token OCTET STRING –token to pass to target. May be empty. Caller must release with gss_release_buffer.

Returned major_status codes:

- GSS_S_COMPLETE indicates that delegation is complete and it is not necessary to call gss_init_delegation again. The returned output_token may have data to be sent to the target to successfully complete the delegation.
- GSS_S_CONTINUE_NEEDED indicates that the information in the returned output_token must be sent to the target and a reply must be received and passed as the input_token argument in a subsequent call to gss_init_delegation.
- GSS_S_DEFECTIVE_TOKEN – consistency checks on input token failed.
- GSS_S_BAD_SIG – bad integrity check on input token.
- GSS_S_NO_CRED – credential is invalid.
- GSS_S_CREDENTIALS_EXPIRED – provided credential has expired.
- GSS_S_OLD_TOKEN – input_token is too old.
- GSS_S_DUPLICATE_TOKEN – input_token is a duplicate.
- GSS_S_BAD_MECH – unsupported mechanism.
- GSS_S_BAD_BINDINGS – this indicates that the extensions provided were invalid (e.g. a mismatch of number of OIDs and buffers).
- GSS_S_FAILURE – error unspecified at the GSS-API level.

This routine is used by either side of a security context to perform a delegation to the peer. The credential being delegated can either be the same credential used to setup the context or a different one.

Input parameters other than input_token must correspond to the same valid GSS-API structures during the delegation process.

The extension_oids and extension_buffers parameters can be used to specify a set of extensions for the delegated credential. There should be an identical number of OIDs and buffers, with each OID indicating what the type of data (e.g. policy language) its corresponding buffer contains. These values may be GSS_C_NO_OID_SET and GSS_C_NO_BUFFER_SET, respectively, to indicate that no extensions are to be applied to the credential.

### 2.2.2 gss_accept_delegation

Inputs:

- sec_context CONTEXT HANDLE – handle to existing security context.
- input_token OCTET STRING – token received from initiator.
- extension_oids SEQUENCE OF OBJECT IDENTIFIER – identifiers for the type of data in the extension_buffers. See description below for a discussion of this option.
- extension_buffers SEQUENCE OF OCTET STRINGS – requested data for extensions to be applied to the delegated credential. See description below for a discussion of this option.
- lifetime_req INTEGER - in seconds, 0 requests default

Outputs:

- delegated_cred CREDENTIAL HANDLE – on completion this will contain a handle to the delegated credential. The caller must release with gss_release_cred.
- lifetime_rec INTEGER - in seconds, or reserved value for INDEFINITE
- output_token OCTET STRING – token to pass to initiator. May be empty. Caller must release with gss_release_buffer.
- mech_type OBJECT_IDENTIFIER – mechanism corresponding to delegated credential; read only, caller should not attempt to release.

Return major_status codes:

- GSS_S_COMPLETE indicates that the delegation has been successful and delegated_cred contains a handle to a usable credential.

- GSS_S_CONTINUE_NEEDED indicates that the information in the returned output_token must be sent to the initiator and a reply must be received and passed as the input_token argument in a subsequent call to gss_accept_delegation.

- GSS_S_DEFECTIVE_TOKEN – malformed token.

- GSS_S_BAD_SIG – integrity check failed.

- GSS_S_OLD_TOKEN – token too old.

- GSS_S_NO_CONTEXT – bad context.

- GSS_S_BAD_MECH – unsupported mechanism for the delegated credential.

- GSS_S_BAD_BINDINGS – this indicates that the extensions provided were invalid (e.g. a mismatch of number of OIDs and buffers).

- GSS_S_FAILURE – Error unspecified at the GSS-API level.

This routine is used by either side of a security context to accept a delegation being performed by gss_init_delegation. The credential being delegated can be either the same credential used to setup the context or a different one.

Input parameters other than input_token must correspond to the same valid GSS-API structures during the delegation process.

The lifetime_rec parameter, on successful completion, will contain the remaining lifetime, in seconds, of the delegated credential. A mechanism may return a reserved value indicating INDEFINITE if no constraints on credential lifetime are imposed. A caller of gss_accept_delegation may also request that the delegated credential be valid for a specified period of time (via the lifetime_req argument), but should be prepared to handle a delegated credential with shorter than requested lifetime.

The extension_oids and extension_buffers parameters can be used to request extensions to be applied to the delegated credential. There should be an identical number of OIDs and buffers with each OID indicating the type of data (e.g. policy language) its corresponding buffer contains. See gss_init_delegation for a discussion of these OIDs. Note that depending on the mechanism, the initiator may ultimately decide to ignore the extensions requested by the acceptor. These values may be GSS_C_NO_OID_SET and GSS_C_NO_BUFFER_SET, respectively, to indicate that no extensions are requested in the credential.

## 2.3 Extended Credential Inquiry

Applications wanting to make more complex authorization decisions require more information about a client's credential than just the client's identity. In order to provide this information the gss_inquire_sec_context_by_oid and gss_inquire_cred_by_oid functions were added. These functions give applications a means to retrieve arbitrary data about a context or credential.

The gss_inquire_sec_context_by_oid and gss_inquire_cred_by_oid functions are called with an object identifier identifying the information that the application is interested in. This object identifier will either be one of a number of generic OIDs or may be mechanism-specific. Generic OIDs should be defined in a future GGF OID namespace. In the absence of such a namespace we recommend the use of the following OID from the Globus OID namespace:

1.3.6.1.4.1.3536.1.1.1.3 – A OID designating any restrictions placed on a credential. Normally these restrictions are placed in a credential using the extensions parameter in the "delegation at any time" functions.

### 2.3.1 gss_inquire_sec_context_by_oid call

Inputs:

- sec_context CONTEXT HANDLE – handle to existing security context.
- desired_object OBJECT IDENTIFIER – OID of desired objects.

Outputs:

- major_status INTEGER
- minor_status INTEGER
- data_set SET OF OCTET STRING – zero or more pieces of data corresponding to the data associated with the desired_object OID. To be freed by caller using gss_release_buffer_set.

Return major_status codes:

- GSS_S_COMPLETE indicates that the function completed successfully and data was returned.
- GSS_S_CONTINUE_NEEDED indicates that there are more data items to be returned (see following description).

- GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context handle provided.

- GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level.

The desired_object parameter should contain an OID that references information the application is interested in. This may for example be a OID that identifies a extension used in the delegation routines.

If the requested data is not present, major_status will be set to GSS_S_COMPLETE and 0 objects will be returned in data_set.

Note that there is disagreement between the current GSS-API RFC [3] and the RFC specifying the C-language bindings [4]. While the GSS-API RFC states that functions that functions of this nature should not iterate using GSS_S_CONTINUE_NEEDED, the RFC specifying language bindings does so. Applications should be prepared to handle this function returning only a single piece of data per call and call it iteratively if it returns GSS_S_CONTINUE_NEEDED.

### 2.3.2 gss_inquire_cred_by_oid call

Inputs:

- cred_handle CREDENTIAL HANDLE - handle to existing credential.

- desired_object OBJECT IDENTIFIER - OID of desired objects.

Outputs:

- major_status INTEGER

- minor_status INTEGER

- data_set SET OF OCTET STRING - zero or more pieces of data corresponding to the data associated with the desired_object OID. To be freed by caller using gss_release_buffer_set.

Return major_status codes:

- GSS_S_COMPLETE indicates that the function completed successfully and data was returned.

- GSS_S_CONTINUE_NEEDED indicates that there are more data items to be returned (see following description).

- GSS_S_NO_CRED indicates there was no valid credential

- GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level.

The desired_object parameter should contain an OID that references information the application is interested in. This may for example correspond to a OID used in the extended delegation routines.

If the requested data is not present, major_status will be set to GSS_S_COMPLETE and 0 objects will be returned in data_set.

Note there is disagreement between the current GSS-API RFC [3] and the RFC specifying the C-language bindings [4]. While the GSS-API RFC states that functions that functions of this nature should not iterate using GSS_S_CONTINUE_NEEDED, the RFC specifying language bindings does so. Applications should be prepared to handle this function returning only a single piece of data per call and call it iteratively if it returns GSS_S_CONTINUE_NEEDED.

## 2.4 Context options

Currently there is no method for passing flags into the gss_accept_sec_context call. Instead of changing the arguments to this function we propose a method that allows us to set options on a security context and then pass the context into a first call of gss_accept_sec_context or gss_init_sec_context. This provides the means for an application to achieve more fine-grained control of the underlying GSS-API mechanism.

The option being changed may be mechanism specific or one of the generic options defined below. The Global Grid Forum will define OIDs for the options below.

Generic options:

GSS_DISALLOW_ENCRYPTION – Setting this option to 1 will cause the underlying GSS library to disallow any encryption of application data.

GSS_PROTECTION_FAIL_ON_CONTEXT_EXPIRATION – Setting this option to 1 will case the underlying GSS library to fail if any message protection operation is attempted after the security context has expired. It is recommended that the underlying mechanism allow this option to be set to 0 after such an error in returned so that an application can finish an ongoing process before returning an error to the user. For example, an application may wish to complete a file transfer already in progress before refusing to start another transfer. Note

that some implementations may not be able to perform message protection after security context expiration.

GSS_APPLICATION_WILL_CHECK_EXTENSIONS – Setting this option to a list of OIDs (using the SET OF OBJECT IDENTIFIER data type) will inform the underlying GSSAPI library that the application will take responsibility for checking any extensions in the peer credential identified by these OIDs . If the GSS mechanism encounters a extension during context establishment that the application has not declared its responsibility for, then the GSS mechanism may choose to fail the authentication. The reason for this is that without the knowledge that the application is aware of and taking responsibility for handling the extensions the GSSAPI mechanism cannot be assured they will be respected and must take the safe course of failure.

### 2.4.1 gss_set_sec_context_option call

Inputs:

- sec_context CONTEXT HANDLE – handle to security context. NULL specifies that a new security context should be created.
- option OBJECT IDENTIFIER – OID of desired option.
- value OCTET STRING – value of option.

Outputs:

- major_status INTEGER
- minor_status INTEGER

Return major_status codes:

- GSS_S_COMPLETE indicates that the function completed successfully.
- GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level. This includes an attempt to change an immutable option.

The sec_context parameter should either be an existing security context or NULL, indicating a new security context should be initialized and the specified option set on the new context.

The option parameter should contain an OID that references the option that the application wishes to set.

The value parameter should contain the value the option should be set to. The size of the octet string will vary depending on the option being set (analogous to the UNIX setsockopt function).

## 2.5 Buffer Set Functions

These functions are needed to support other new functions added in this section.

### 2.5.1 gss_create_empty_buffer_set

Inputs:

- None

Outputs:

- buffer_set SET OF OCTET STRING To be freed by caller using gss_release_buffer_set.
- major_status INTEGER
- minor_status INTEGER

Return major_status codes:

- GSS_S_COMPLETE indicates that the function completed successfully.
- GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level.

Creates a empty buffer set, to which members may subsequently be added using the gss_add_buffer_set_member routine.

### 2.5.2 gss_add_buffer_set_member call

Inputs:

- member_buffer OCTET_STRING
- buffer_set SET OF OCTET STRING

Outputs:

- major_status INTEGER

- minor_status INTEGER

Return major_status codes:

- GSS_S_COMPLETE indicates that the function completed successfully.
- GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level.

This function adds a buffer to a buffer set. The member_buffer argument may be released after the completion of this call.

### 2.5.3 gss_release_buffer_set call

Inputs:

- buffer_set SET OF OCTET STRING

Outputs:

- major_status INTEGER
- minor_status INTEGER

Return major_status codes:

- GSS_S_COMPLETE indicates that the function completed successfully and data was returned.
- GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level.

This function allows callers to release the storage associated with a SET OF OCTET STRING allocated by another GSS-API call. This call's specific behavior depends on the language and programming environment within which a GSS-API implementation operates, and is therefore detailed within applicable bindings specifications; in particular, implementation and invocation of this call may be superfluous (and may be omitted) within bindings where memory management is automatic.

# 3 Changes to existing functions

This section contains changes to the behavior of the existing GSSAPI functions.

## 3.1 gss_accept_sec_context

The gss_accept_sec_context call will now accept a non-NULL security context, as created by the gss_set_sec_context_option call, on initial call.

## 3.2 gss_init_sec_context

The gss_init_sec_context call will now accept a non-NULL security context, as created by the gss_set_sec_context_option call, on initial call.

## 3.3 gss_getMIC, gss_verifyMIC, gss_wrap, gss_unwrap

If the GSS_PROTECTION_FAIL_ON_CONTEXT_EXPIRATION option has been set to 1 by the gss_set_sec_context_option call then these functions should fail if the security context being used has expired.

If the GSS_PROTECTION_FAIL_ON_CONTEXT_EXPIRATION option is set to 0 then functions should continue to function as long as they possibly can regardless of the expiration of security context. Some implementation may not be able to continue functioning after security context expiration.

# 4 Additional Desired Functionality

The following sections describe shortcomings in the GSSAPI and propose possible schemes for modifying the API to address these but do not lay out a specific proposal for change. It is the authors' intention to propose a scheme at a later date.

## 4.1 Token Framing

One major shortcoming in the GSS-API specification is that it does not prescribe or recommend a standard mechanism independent way of framing tokens generated by GSS-API functions.
We feel that the absence of a framing mechanism negates the utility of the mech_type and similar parameters, since demultiplexing tokens based on mechanism without said framing becomes nearly impossible.

We realize that almost all mechanisms used to implement the GSS-API provide their own framing mechanism and may not want to add additional framing since this may hinder

interoperability with non-GSS-API using applications. Thus any proposed framing should not be mandatory.

It is our opinion that any proposed framing mechanism should at the bare minimum allow for the following:

1. A mechanism for determining the length of the token
2. A way of determining the mechanism that produced the token
3. A mechanism for identifying the token type, where token type indicates the origin (e.g. context establishment, application data …) of the token

Finally, it may be argued that framing could and should be done at the application level, but even in this case a recommendation would give the benefit of greater interoperability between mechanisms which implement framing at the GSSAPI level.

## 4.2 Different levels of verbosity with gss_display_status

It is desirable to have different levels of error message verbosity at different times. When returning an error message to a user, simplicity is often the best course. Users tend to want a simple description of the error and maybe a little guidance on how to fix the problem. Long error messages tend to scare them away.

On the other hand, if a developer or other user seriously interested in debugging is reading the error message, the more information the better.

The gss_display_status call currently offers no way to select different levels of verbosity in the messages it returns. Some possible ways of implementing this are:

- Add a way of passing in a flag to the gss_display_call perhaps by defining more values for status type.

- Defining a predefined string at the beginning of all strings returned by gss_display_status to indicate the verbosity level of the string. This has the advantage that the server can send these strings over the network to a client that can decide the level of verbosity without involving the server.

- Defining a new call, gss_display_status_ext, that has an option to specify verbosity level.

# 5 Security Considerations

## 5.1 Credential export and import

Since exported credential buffers may contain sensitive information (e.g. the credential itself) care must be taken to maintain the privacy of this information. Precautions should include:

- Storing the exported credential only in files with carefully controlled access polices
- Not sending the exported credential over the network in the clear (unencrypted)
- Being careful which processes are given access to the credential

## 5.2 Delegation at any time and restricted delegation handling

Applications delegating credentials need to be sure that they are delegating credentials only to other processes and systems that can be trusted. Since credentials are often stored on disk, compromises of systems can lead to stolen credentials from trusted processes.

The use of restrictive policies is encouraged for this reason as it can greatly limit the use of any stolen credential.

# 6 C-Bindings

The following are the C-bindings for the API extensions:

```
typedef gss_buffer_set_desc_struct {
    size_t                  count;
    gss_buffer_desc *       elements;
} gss_buffer_set_desc, *gss_buffer_set_t;

OM_uint32
gss_create_empty_buffer_set(
    OM_uint32 *             minor_status,
    gss_buffer_set_t *      buffer_set)

OM_uint32
gss_add_buffer_set_member(
    OM_uint32 *             minor_status,
    const gss_buffer_t      member_buffer,
    gss_buffer_set_t *      buffer_set)

OM_uint32
gss_release_buffer_set(
    OM_uint32 *             minor_status,
    gss_buffer_set_t        buffer_set)

OM_uint32
gss_export_cred(
    OM_uint32 *             minor_status,
    const gss_cred_id_t     cred_handle,
    const gss_OID           desired_mech,
    gss_OID *               actual_mech,
    OM_uint32               option_req,
    const gss_buffer_t      protection_key,
    gss_buffer_t            export_buffer);

OM_uint32
gss_import_cred(
    OM_uint32 *             minor_status,
    gss_cred_id_t *         output_cred_handle,
    const gss_OID           desired_mech,
    gss_OID *               actual_mech,
    OM_uint32               option_req,
    const gss_buffer_t      import_buffer,
    const gss_buffer_t      protection_key,
```

```
    OM_uint32               time_req,
    OM_uint32 *             time_rec);

OM_uint32
gss_init_delegation(
    OM_uint32 *             minor_status,
    const gss_ctx_id_t      context_handle,
    const gss_cred_id_t     cred_handle,
    const gss_OID           desired_mech,
    const gss_OID_set       extension_oids,
    const gss_buffer_set_t  extension_buffers,
                            const gss_buffer_t    input_token,
    OM_uint32               time_req,
    gss_buffer_t            output_token);

OM_uint32
gss_accept_delegation(
    OM_uint32 *             minor_status,
    const gss_ctx_id_t      context_handle,
    const gss_OID_set       extension_oids,
    const gss_buffer_set_t  extension_buffers,
    const gss_buffer_t      input_token,
    OM_uint32               time_req,
    OM_uint32 *             time_rec,
    gss_cred_id_t *         delegated_cred_handle,
    gss_OID                 mech_type,
    gss_buffer_t            output_token);


OM_uint32
gss_inquire_sec_context_by_oid(
    OM_uint32 *             minor_status,
    const gss_ctx_id_t      context_handle,
    const gss_OID           desired_object,
    gss_buffer_set_t        data_set);


OM_uint32
gss_inquire_cred_by_oid(
    OM_uint32 *             minor_status,
    const gss_ced_id_t      cred_handle,
    const gss_OID           desired_object,
    gss_buffer_set_t *      data_set);

OM_uint32
```

```
gss_set_sec_context_option(
    OM_uint32 *          minor_status,
    gss_ctx_id_t *       context_handle,
    const gss_OID        option,
    gss_buffer_t         value);
```

# 7 References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997.

[2] Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, 2001.

[3] Linn, J., "Generic Security Service Application Program Interface, Version 2, Update 1," RFC 2743, January 2000.

[4] Wray, J., "Generic Security Service API Version 2, C-bindings," RFC 2744, January 2000.

# 8 Acknowledgments

We are grateful to numerous colleagues for discussions on the topics covered in this paper, in particular (in alphabetical order, with apologies to anybody we've missed): Joe Bester, Randy Butler, Carl Kesselman, Keith Jackson, Bill Johnston, Ian Foster, Marty Humphrey, Cliff Neuman, Laura Pearlman, Frank Siebenlist, Mary Thompson, Gene Tsudik.

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; by the National Science Foundation; and by the NASA Information Power Grid project.

# 9 Change Log

Changes between version 0.6 and 0.7

1. Added protection_key parameter to gss_import_cred() and gss_export_cred() functions.

Changes between version 0.5 and 0.6

2. Updated section on token framing
3. Added lifetime_req parameter to gss_accept_delegation
4. Changed formatting to comply with GGF standard
5. Added actual_mech parameter to gss_export_cred and gss_import_cred
6. Updated C bindings

Changes between version 0.4 and 0.5:

1. Renamed delegation restrictions to more generic delegation extensions.
2. Added gss_create_empty_buffer_set() and gss_add_buffer_set_member()
3. Added gss_inquire_cred_by_oid()

Changes between version 0.3 and 0.4:

1. Added time_req argument to gss_init_delegation().
2. NULL credential to gss_init_delegation() means to use the default credential.
3. Oids and buffers in calls to init_delegation and accept_delegation are now sequences and not sets.
4. Added text to 2.2 stating essentially that only one delegation is allowed to be in progress at a given time.
5. Revised text in 2.2.1 and 2.2.2 to attempt to clarify restriction oids specify policy languages in restriction buffers.
6. Added text to 2.3 about generic oids for retrieving information from the credential and in particular an oid for policies inserted by gss_init_delegation().
7. Added lifetime_rec output to gss_accept_delegation().
8. Added lifetime_req and lifetime_rec parameters to gss_import_context().
9. Added section 2.4: gss_set_sec_context_option(). Deleted section in section 3 that talked about wanting to do this.
10. Added new section 3 disussing changes to existing functions with renumbering elsewhere in the document.
11. Added section 2.5, Housekeeping functions, with subsection 2.5.1 describing the gss_release_buffer_set call.
12. Message protection behavior on context expiration is now defined by sections 2.4 and 3.3.
13. Added section 4.2 on the need for verbosity levels in gss_display_status.
14. In Section 5.1 now use credential singular consistently

15. In 2.2.1 and 2.2.2 renamed restrictions_oid to restrictions_oid and restrictions_buffers to restriction_buffers. Also indicated these may be GSS_C_NO_OID_SET and GSS_C_NO_BUFFER_SET respectively.
16. 2.2.1 input_token: GSS_C_NO_BUFFER instead of NULL
17. 2.2.1 and 2.2.2 output_token: May be empty instead of NULL

# 10 Contact Information

Samuel Meder
Distributed Systems Laboratory
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
Phone: 630-252-1752
Email: meder@mcs.anl.gov

Von Welch
University of Chicago
Email: welch@mcs.anl.gov

Steven Tuecke
Argonne National Laboratory
Email: tuecke@mcs.anl.gov

Doug Engert
Argonne National Laboratory
Email: deengert@anl.gov

# 11 Copyright Notice

# 12 Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.