Internet Draft   J. Wray
Common Authentication Technology Working Group Iris Associates
<draft-ietf-cat-gssv2-cbind-09.txt>   February 10, 1999

Generic Security Service API Version 2 : C-bindings

1. STATUS OF THIS MEMO

Comments on this document should be sent to "cat-ietf@mit.edu", the IETF Common Authentication Technology WG discussion list.

2. ABSTRACT

This draft document specifies C language bindings for Version 2 of the Generic Security Service Application Program Interface (GSS-API), which is described at a language-independent conceptual level in other drafts [GSSAPI]. It revises RFC-1509, making specific incremental changes in response to implementation experience and liaison requests. It is intended, therefore, that this draft or a successor version thereof will become the basis for subsequent progression of the GSS-API specification on the standards track.

The Generic Security Service Application Programming Interface provides security services to its callers, and is intended for implementation atop a variety of underlying cryptographic mechanisms. Typically, GSS-API callers will be application protocols into which security enhancements are integrated through invocation of services provided by the GSS-API. The GSS-API allows a caller application to authenticate a principal identity associated with a peer application, to delegate rights to a peer, and to apply security services such as confidentiality and integrity on a per-message basis.

3. INTRODUCTION

The Generic Security Service Application Programming Interface [GSSAPI] provides security services to calling applications. It allows a communicating application to authenticate the user associated with another application, to delegate rights to another application, and to apply security services such as confidentiality and integrity on a per-message basis.

There are four stages to using the GSS-API:

a) The application acquires a set of credentials with which it may prove its identity to other processes. The application's credentials vouch for its global identity, which may or may not be related to any local username under which it may be running.

b) A pair of communicating applications establish a joint security context using their credentials. The security context is a pair of GSS-API data structures that contain shared state information, which is required in order that per-message security services may be provided. Examples of state that might be shared between applications as part of a security context are cryptographic keys, and message sequence numbers. As part of the establishment of a security context, the context initiator is authenticated to the responder, and may require that the responder is authenticated in turn. The initiator may optionally give the responder the right to initiate further security contexts, acting as an agent or delegate of the initiator. This transfer of rights is termed delegation, and is achieved by creating a set of credentials, similar to those used by the initiating application, but which may be used by the responder.

To establish and maintain the shared information that makes up the security context, certain GSS-API calls will return a token data structure, which is an opaque data type that may contain cryptographically protected data. The caller of such a GSS-API routine is responsible for transferring the token to the peer application, encapsulated if necessary in an application-application protocol. On receipt of such a token, the peer application should pass it to a corresponding GSS-API routine which will decode the token and extract the information, updating the security context state information accordingly.

c) Per-message services are invoked to apply either:

   integrity and data origin authentication, or

   confidentiality, integrity and data origin authentication

to application data, which are treated by GSS-API as arbitrary octet-strings. An application transmitting a message that it wishes to protect will call the appropriate GSS-API routine (gss_get_mic or gss_wrap) to apply protection, specifying the appropriate security context, and send the resulting token to the receiving application. The receiver will pass the received token (and, in the case of data protected by gss_get_mic, the accompanying message-data) to the corresponding decoding routine (gss_verify_mic or gss_unwrap) to remove the protection and validate the data.

d) At the completion of a communications session (which may extend across several transport connections), each application calls a GSS-API routine to delete the security context. Multiple contexts may also be used (either successively or simultaneously) within a single communications association, at the option of the applications.

4.   GSS-API ROUTINES

This section lists the routines that make up the GSS-API, and offers a brief description of the purpose of each routine. Detailed descriptions of each routine are listed in alphabetical order in section 7.

Table 4-1 GSS-API Credential-management Routines

Routine	Section	Function
gss_acquire_cred	7.2	Assume a global identity; Obtain a GSS-API credential handle for pre-existing credentials.
gss_add_cred	7.3	Construct credentials incrementally
gss_inquire_cred	7.21	Obtain information about a credential
gss_inquire_cred_by_mech	7.22	Obtain per-mechanism information about a credential.
gss_release_cred	7.27	Discard a credential handle.

Table 4-2 GSS-API Context-Level Routines

Routine	Section	Function
gss_init_sec_context	7.19	Initiate a security context with a peer application
gss_accept_sec_context	7.1	Accept a security context initiated by a peer application
gss_delete_sec_context	7.9	Discard a security context
gss_process_context_token	7.25	Process a token on a security context from a peer application
gss_context_time	7.7	Determine for how long a context will remain valid
gss_inquire_context	7.20	Obtain information about a security context
gss_wrap_size_limit	7.34	Determine token-size limit for gss_wrap on a context

gss_export_sec_context	7.14	Transfer a security context to another process
gss_import_sec_context	7.17	Import a transferred context

Table 4-3 GSS-API Per-message Routines

Routine	Section	Function
gss_get_mic	7.15	Calculate a cryptographic message integrity code (MIC) for a message; integrity service
gss_verify_mic	7.32	Check a MIC against a message; verify integrity of a received message
gss_wrap	7.33	Attach a MIC to a message, and optionally encrypt the message content; confidentiality service
gss_unwrap	7.31	Verify a message with attached MIC, and decrypt message content if necessary.

Table 4-4 GSS-API Name manipulation Routines

Routine	Section	Function
gss_import_name	7.16	Convert a contiguous string name to internal-form
gss_display_name	7.10	Convert internal-form name to text
gss_compare_name	7.6	Compare two internal-form names
gss_release_name	7.28	Discard an internal-form name
gss_inquire_names_for_mech	7.24	List the name-types supported by the specified mechanism
gss_inquire_mechs_for_name	7.23	List mechanisms that support the specified name-type
gss_canonicalize_name	7.5	Convert an internal name to an MN
gss_export_name	7.13	Convert an MN to export form
gss_duplicate_name	7.12	Create a copy of an internal name

Table 4-5 GSS-API Miscellaneous Routines

Routine	Section	Function
gss_add_oid_set_member	7.4	Add an object identifier to a set
gss_display_status	7.11	Convert a GSS-API status code to text
gss_indicate_mechs	7.18	Determine available underlying authentication mechanisms
gss_release_buffer	7.26	Discard a buffer
gss_release_oid_set	7.29	Discard a set of object identifiers
gss_create_empty_oid_set	7.8	Create a set containing no object identifiers
gss_test_oid_set_member	7.30	Determines whether an object identifier is a member of a set.

Individual GSS-API implementations may augment these routines by providing additional mechanism-specific routines if required functionality is not available from the generic forms. Applications are encouraged to use the generic routines wherever possible on portability grounds.

5.   DATA TYPES AND CALLING CONVENTIONS

The following conventions are used by the GSS-API C-language bindings:

5.1. Integer types

GSS-API uses the following integer data type:

```
   OM_uint32    32-bit unsigned integer
```

Where guaranteed minimum bit-count is important, this portable data type is used by the GSS-API routine definitions. Individual GSS-API implementations will include appropriate typedef definitions to map this type onto a built-in data type. If the platform supports the X/Open xom.h header file, the OM_uint32 definition contained therein should be used; the GSS-API header file in Appendix A contains logic that will detect the prior inclusion of xom.h, and will not attempt to re-declare OM_uint32. If the X/Open header file is not available on the platform, the GSS-API implementation should use the smallest natural unsigned integer type that provides at least 32 bits of precision.

5.2. String and similar data

Many of the GSS-API routines take arguments and return values that describe contiguous octet-strings. All such data is passed between the GSS-API and the caller using the gss_buffer_t data type. This data type is a pointer to a buffer descriptor, which consists of a length field that contains the total number of bytes in the datum, and a value field which contains a pointer to the actual datum:

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

Storage for data returned to the application by a GSS-API routine using the gss_buffer_t conventions is allocated by the GSS-API routine. The application may free this storage by invoking the gss_release_buffer routine. Allocation of the gss_buffer_desc object is always the responsibility of the application; unused gss_buffer_desc objects may be initialized to the value GSS_C_EMPTY_BUFFER.

5.2.1. Opaque data types

Certain multiple-word data items are considered opaque data types at the GSS-API, because their internal structure has no significance either to the GSS-API or to the caller. Examples of such opaque data types are the input_token parameter to gss_init_sec_context (which is opaque to the caller), and the input_message parameter to gss_wrap (which is opaque to the GSS-API). Opaque data is passed between the GSS-API and the application using the gss_buffer_t datatype.

5.2.2. Character strings

Certain multiple-word data items may be regarded as simple ISO Latin-1 character strings. Examples are the printable strings passed to gss_import_name via the input_name_buffer parameter. Some GSS-API routines also return character strings. All such character strings are passed between the application and the GSS-API implementation using the gss_buffer_t datatype, which is a pointer to a gss_buffer_desc object.

When a gss_buffer_desc object describes a printable string, the length field of the gss_buffer_desc should only count printable characters within the string. In particular, a trailing NUL character should NOT be included in the length count, nor should either the GSS-API implementation or the application assume the presence of an uncounted trailing NUL.

5.3. Object Identifiers

Certain GSS-API procedures take parameters of the type gss_OID, or Object identifier. This is a type containing ISO-defined tree-structured values, and is used by the GSS-API caller to select an underlying security mechanism and to specify namespaces. A value of type gss_OID has the following structure:

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

The elements field of this structure points to the first byte of an octet string containing the ASN.1 BER encoding of the value portion of the normal BER TLV encoding of the gss_OID. The length field contains the number of bytes in this value. For example, the gss_OID value corresponding to {iso(1) identified-organization(3) icd-ecma(12) member-company(2) dec(1011) cryptoAlgorithms(7) DASS(5)}, meaning the DASS X.509 authentication mechanism, has a length field of 7 and an elements field pointing to seven octets containing the following octal values: 53,14,2,207,163,7,5. GSS-API implementations should provide constant gss_OID values to allow applications to request any supported mechanism, although applications are encouraged on portability grounds to accept the default mechanism. gss_OID values should also be provided to allow applications to specify particular name types (see section 5.10). Applications should treat gss_OID_desc values returned by GSS-API routines as read-only. In particular, the application should not attempt to deallocate them with free(). The gss_OID_desc datatype is equivalent to the X/Open OM_object_identifier datatype[XOM].

5.4. Object Identifier Sets

Certain GSS-API procedures take parameters of the type gss_OID_set. This type
represents one or more object identifiers (section 5.3). A gss_OID_set object has the following structure:

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

The count field contains the number of OIDs within the set. The elements field is a pointer to an array of gss_OID_desc objects, each of which describes a single OID. gss_OID_set values are used to name the available mechanisms supported by the GSS-API, to request the use of specific mechanisms, and to indicate which mechanisms a given credential supports.

All OID sets returned to the application by GSS-API are dynamic objects (the gss_OID_set_desc, the "elements" array of the set, and the "elements" array of each member OID are all dynamically allocated), and this storage must be deallocated by the application using the gss_release_oid_set() routine.

5.5. Credentials

A credential handle is a caller-opaque atomic datum that identifies a GSS-API credential data structure. It is represented by the caller-opaque type gss_cred_id_t, which should be implemented as a pointer or arithmetic type. If a pointer implementation is chosen, care must be taken to ensure that two gss_cred_id_t values may be compared with the == operator.

GSS-API credentials can contain mechanism-specific principal authentication data for multiple mechanisms. A GSS-API credential is composed of a set of credential-elements, each of which is applicable to a single mechanism. A credential may contain at most one credential-element for each supported mechanism. A credential-element identifies the data needed by a single mechanism to authenticate a single principal, and conceptually contains two credential-references that describe the actual mechanism-specific authentication data, one to be used by GSS-API for initiating contexts, and one to be used for accepting contexts. For mechanisms that do not distinguish between acceptor and initiator credentials, both references would point to the same underlying mechanism-specific authentication data.

Credentials describe a set of mechanism-specific principals, and give their holder the ability to act as any of those principals. All principal identities asserted by a single GSS-API credential should belong to the same entity, although enforcement of this property is an implementation-specific matter. The GSS-API does not make the actual credentials available to applications; instead a credential handle is used to identify a particular credential, held internally by GSS-API. The combination of GSS-API credential handle and mechanism identifies the principal whose identity will be asserted by the credential when used with that mechanism.

The gss_init_sec_context and gss_accept_sec_context routines allow the value GSS_C_NO_CREDENTIAL to be specified as their credential handle parameter. This special credential-handle indicates a desire by the application to act as a default principal. While individual GSS-API implementations are free to determine such default behavior as appropriate to the mechanism, the following default behavior by these routines is recommended for portability:

gss_init_sec_context

1) If there is only a single principal capable of initiating security contexts for the chosen mechanism that the application is authorized to act on behalf of, then that principal shall be used, otherwise

2) If the platform maintains a concept of a default network-identity for the chosen mechanism, and if the application is authorized to act on behalf of that identity for the purpose of initiating security contexts, then the principal corresponding to that identity shall be used, otherwise

3) If the platform maintains a concept of a default local identity, and provides a means to map local identities into network-identities for the chosen mechanism, and if the application is authorized to act on behalf of the network-identity image of the default local identity for the purpose of initiating security contexts using the chosen mechanism, then the principal corresponding to that identity shall be used, otherwise

4) A user-configurable default identity should be used.

gss_accept_sec_context

1) If there is only a single authorized principal identity capable of accepting security contexts for the chosen mechanism, then that principal shall be used, otherwise

2) If the mechanism can determine the identity of the target principal by examining the context-establishment token, and if the accepting application is authorized to act as that principal for the purpose of accepting security contexts using the chosen mechanism, then that principal identity shall be used, otherwise

3) If the mechanism supports context acceptance by any principal, and if mutual authentication was not requested, any principal that the application is authorized to accept security contexts under using the chosen mechanism may be used, otherwise

4)A user-configurable default identity shall be used.

The purpose of the above rules is to allow security contexts to be established by both initiator and acceptor using the default behavior wherever possible. Applications requesting default behavior are likely to be more portable across mechanisms and platforms than ones that use gss_acquire_cred to request a specific identity.

5.6. Contexts

The gss_ctx_id_t data type contains a caller-opaque atomic value that identifies one end of a GSS-API security context. It should be implemented as a pointer or arithmetic type. If a pointer type is chosen, care should be taken to ensure that two gss_ctx_id_t values may be compared with the == operator.

The security context holds state information about each end of a peer communication, including cryptographic state information.

5.7. Authentication tokens

A token is a caller-opaque type that GSS-API uses to maintain synchronization between the context data structures at each end of a GSS-API security context. The token is a cryptographically protected octet-string, generated by the underlying mechanism at one end of a GSS-API security context for use by the peer mechanism at the other end. Encapsulation (if required) and transfer of the token are the responsibility of the peer applications. A token is passed between the GSS-API and the application using the gss_buffer_t conventions.

5.8. Interprocess tokens

Certain GSS-API routines are intended to transfer data between processes in multi-process programs. These routines use a caller-opaque octet-string, generated by the GSS-API in one process for use by the GSS-API in another process. The calling application is responsible for transferring such tokens between processes in an OS-specific manner. Note that, while GSS-API implementors are encouraged to avoid placing sensitive information within interprocess tokens, or to cryptographically protect them, many implementations will be unable to avoid placing key material or other sensitive data within them. It is the application's responsibility to ensure that interprocess tokens are protected in transit, and transferred only to processes that are trustworthy. An interprocess token is passed between the GSS-API and the application using the gss_buffer_t conventions.

5.9. Status values

Every GSS-API routine returns two distinct values to report status information to the caller: GSS status codes and Mechanism status codes.

5.9.1.   GSS status codes

GSS-API routines return GSS status codes as their OM_uint32 function value. These codes indicate errors that are independent of the underlying mechanism(s) used to provide the security service. The errors that can be indicated via a GSS status code are either generic API routine errors (errors that are defined in the GSS-API specification) or calling errors (errors that are specific to these language bindings).

A GSS status code can indicate a single fatal generic API error from the routine and a single calling error. In addition, supplementary status information may be indicated via the setting of bits in the supplementary info field of a GSS status code.

These errors are encoded into the 32-bit GSS status code as follows:

```
      MSB                                                        LSB
      |------------------------------------------------------------|
      |  Calling Error | Routine Error  |    Supplementary Info    |
      |------------------------------------------------------------|
   Bit 31            24 23            16 15                       0
```

Hence if a GSS-API routine returns a GSS status code whose upper 16 bits contain a non-zero value, the call failed. If the calling error field is non-zero, the invoking application's call of the routine was erroneous. Calling errors are defined in table 5-1. If the routine error field is non-zero, the routine failed for one of the routine-specific reasons listed below in table 5-2. Whether or not the upper 16 bits indicate a failure or a success, the routine may indicate additional information by setting bits in the supplementary info field of the status code. The meaning of individual bits is listed below in table 5-3.

Table 5-1 Calling Errors

Name	Value in field	Meaning
GSS_S_CALL_INACCESSIBLE_READ	1	A required input parameter could not be read
GSS_S_CALL_INACCESSIBLE_WRITE	2	A required output parameter could not be written.
GSS_S_CALL_BAD_STRUCTURE	3	A parameter was malformed

Table 5-2 Routine Errors

Name	Value in field	Meaning
GSS_S_BAD_MECH	1	An unsupported mechanism was requested
GSS_S_BAD_NAME	2	An invalid name was supplied
GSS_S_BAD_NAMETYPE	3	A supplied name was of an unsupported type
GSS_S_BAD_BINDINGS	4	Incorrect channel bindings were supplied

GSS_S_BAD_STATUS	5	An invalid status code was supplied
GSS_S_BAD_MIC GSS_S_BAD_SIG	6	A token had an invalid MIC
GSS_S_NO_CRED	7	No credentials were supplied, or the credentials were unavailable or inaccessible.
GSS_S_NO_CONTEXT	8	No context has been established
GSS_S_DEFECTIVE_TOKEN	9	A token was invalid
GSS_S_DEFECTIVE_CREDENTIAL	10	A credential was invalid
GSS_S_CREDENTIALS_EXPIRED	11	The referenced credentials have expired
GSS_S_CONTEXT_EXPIRED	12	The context has expired
GSS_S_FAILURE	13	Miscellaneous failure (see text)
GSS_S_BAD_QOP	14	The quality-of-protection requested could not be provided
GSS_S_UNAUTHORIZED	15	The operation is forbidden by local security policy
GSS_S_UNAVAILABLE	16	The operation or option is unavailable
GSS_S_DUPLICATE_ELEMENT	17	The requested credential element already exists
GSS_S_NAME_NOT_MN	18	The provided name was not a mechanism name

Table 5-3 Supplementary Status Bits

Name	Bit Number	Meaning
GSS_S_CONTINUE_NEEDED	0 (LSB)	Returned only by gss_init_sec_context or gss_accept_sec_context. The routine must be called again to complete its function. See routine documentation for detailed description
GSS_S_DUPLICATE_TOKEN	1	The token was a duplicate of an earlier token

GSS_S_OLD_TOKEN	2	The token's validity period has expired
GSS_S_UNSEQ_TOKEN	3	A later token has already been processed
GSS_S_GAP_TOKEN	4	An expected per-message token was not received

The routine documentation also uses the name GSS_S_COMPLETE, which is a zero value, to indicate an absence of any API errors or supplementary information bits.

All GSS_S_xxx symbols equate to complete OM_uint32 status codes, rather than to bitfield values. For example, the actual value of the symbol GSS_S_BAD_NAMETYPE (value 3 in the routine error field) is 3<<16.
The macros GSS_CALLING_ERROR(), GSS_ROUTINE_ERROR() and GSS_SUPPLEMENTARY_INFO() are provided, each of which takes a GSS status code and removes all but the relevant field. For example, the

value obtained by applying GSS_ROUTINE_ERROR to a status code removes the calling errors and supplementary info fields, leaving only the routine errors field. The values delivered by these macros may be directly compared with a GSS_S_xxx symbol of the appropriate type.
The macro GSS_ERROR() is also provided, which when applied to a GSS status code returns a non-zero value if the status code indicated a calling or routine error, and a zero value otherwise. All macros defined by GSS-API evaluate their argument(s) exactly once.

A GSS-API implementation may choose to signal calling errors in a platform-specific manner instead of, or in addition to the routine value; routine errors and supplementary info should be returned via major status values only.

The GSS major status code GSS_S_FAILURE is used to indicate that the underlying mechanism detected an error for which no specific GSS status code is defined. The mechanism-specific status code will provide more details about the error.

5.9.2. Mechanism-specific status codes

GSS-API routines return a minor_status parameter, which is used to indicate specialized errors from the underlying security mechanism. This parameter may contain a single mechanism-specific error, indicated by a OM_uint32 value.

The minor_status parameter will always be set by a GSS-API routine, even if it returns a calling error or one of the generic API errors indicated above as fatal, although most other output parameters may remain unset in such cases. However, output parameters that are expected to return pointers to storage allocated by a routine must always be set by the routine, even in the event of an error, although in such cases the GSS-API routine may elect to set the returned parameter value to NULL to indicate that no storage was actually allocated. Any length field associated with such pointers (as in a gss_buffer_desc structure) should also be set to zero in such cases.

5.10.Names

A name is used to identify a person or entity. GSS-API authenticates the relationship between a name and the entity claiming the name.

Since different authentication mechanisms may employ different namespaces for identifying their principals, GSSAPI's naming support is necessarily complex in multi-mechanism environments (or even in some single-mechanism environments where the underlying mechanism supports multiple namespaces).

Two distinct representations are defined for names:

An internal form. This is the GSS-API "native" format for names, represented by the implementation-specific gss_name_t type.

It is opaque to GSS-API callers. A single gss_name_t object may contain multiple names from different namespaces, but all names should refer to the same entity. An example of such an internal name would be the name returned from a call to the gss_inquire_cred routine, when applied to a credential containing credential elements for multiple authentication mechanisms employing different namespaces. This gss_name_t object will contain a distinct name for the entity for each authentication mechanism.

For GSS-API implementations supporting multiple namespaces, objects of type gss_name_t must contain sufficient information to determine the namespace to which each primitive name belongs.

Mechanism-specific contiguous octet-string forms. A format
    capable of containing a single name (from a single
    namespace). Contiguous string names are always accompanied
    by an object identifier specifying the namespace to which
    the name belongs, and their format is dependent on the
    authentication mechanism that employs the name. Many, but
    not all, contiguous string names will be printable, and may
    therefore be used by GSS-API applications for communication
    with their users.

Routines (gss_import_name and gss_display_name) are provided to convert names between contiguous string representations and the internal gss_name_t type. gss_import_name may support multiple syntaxes for each supported namespace, allowing users the freedom to choose a preferred name representation. gss_display_name should use an implementation-chosen printable syntax for each supported name-type.

If an application calls gss_display_name(), passing the internal name resulting from a call to gss_import_name(), there is no guarantee the the resulting contiguous string name will be the same as the original imported string name. Nor do name-space identifiers necessarily survive unchanged after a journey through the internal name-form. An example of this might be a mechanism that authenticates X.500 names, but provides an algorithmic mapping of Internet DNS names into X.500. That mechanism's implementation of gss_import_name() might, when presented with a DNS name, generate an internal name that contained both the original DNS name and the equivalent X.500 name. Alternatively, it might only store the X.500 name. In the latter case, gss_display_name() would most likely generate a printable X.500 name, rather than the original DNS name.

The process of authentication delivers to the context acceptor an internal name. Since this name has been authenticated by a single mechanism, it contains only a single name (even if the internal name presented by the context initiator to gss_init_sec_context had multiple components). Such names are termed internal mechanism names, or "MN"s and the names emitted by gss_accept_sec_context() are always of this type. Since some applications may require MNs without wanting to incur the overhead of an authentication operation, a

second function, gss_canonicalize_name(), is provided to convert a general internal name into an MN.

Comparison of internal-form names may be accomplished via the gss_compare_name() routine, which returns true if the two names being compared refer to the same entity. This removes the need for the application program to understand the syntaxes of the various printable names that a given GSS-API implementation may support. Since GSS-API assumes that all primitive names contained within a given internal name refer to the same entity, gss_compare_name() can return true if the two names have at least one primitive name in common. If the implementation embodies knowledge of equivalence relationships between names taken from different namespaces, this knowledge may also allow successful comparison of internal names containing no overlapping primitive elements.

When used in large access control lists, the overhead of invoking gss_import_name() and gss_compare_name() on each name from the ACL may be prohibitive. As an alternative way of supporting this case, GSS-API defines a special form of the contiguous string name which may be compared directly (e.g. with memcmp()). Contigous names suitable for comparison are generated by the gss_export_name() routine, which requires an MN as input. Exported names may be re-imported by the gss_import_name() routine, and the resulting internal name will also be an MN. The gss_OID constant GSS_C_NT_EXPORT_NAME indentifies the "export name" type, and the value of this constant is given in Appendix A. Structurally, an exported name object consists of a header containing an OID identifying the mechanism that authenticated the name, and a trailer containing the name itself, where the syntax of the trailer is defined by the individual mechanism specification. The precise format of an export name is defined in the language-independent GSS-API specification [GSSAPI].

Note that the results obtained by using gss_compare_name() will in general be different from those obtained by invoking gss_canonicalize_name() and gss_export_name(), and then comparing the exported names. The first series of operation determines whether two (unauthenticated) names identify the same principal; the second whether a particular mechanism would authenticate them as the same principal. These two operations will in general give the same results only for MNs.

The gss_name_t datatype should be implemented as a pointer type. To allow the compiler to aid the application programmer by performing type-checking, the use of (void *) is discouraged. A pointer to an implementation-defined type is the preferred choice.

Storage is allocated by routines that return gss_name_t values. A procedure, gss_release_name, is provided to free storage associated with an internal-form name.

5.11.Channel Bindings

GSS-API supports the use of user-specified tags to identify a given context to the peer application. These tags are intended to be used to identify the particular communications channel that carries the context. Channel bindings are communicated to the GSS-API using the following structure:

```
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;
```

The initiator_addrtype and acceptor_addrtype fields denote the type of addresses contained in the initiator_address and acceptor_address buffers. The address type should be one of the following:

GSS_C_AF_UNSPEC	Unspecified address type
GSS_C_AF_LOCAL	Host-local address type
GSS_C_AF_INET	Internet address type (e.g. IP)
GSS_C_AF_IMPLINK	ARPAnet IMP address type
GSS_C_AF_PUP	pup protocols (eg BSP) address type
GSS_C_AF_CHAOS	MIT CHAOS protocol address type
GSS_C_AF_NS	XEROX NS address type
GSS_C_AF_NBS	nbs address type
GSS_C_AF_ECMA	ECMA address type
GSS_C_AF_DATAKIT	datakit protocols address type
GSS_C_AF_CCITT	CCITT protocols
GSS_C_AF_SNA	IBM SNA address type
GSS_C_AF_DECnet	DECnet address type
GSS_C_AF_DLI	Direct data link interface address type
GSS_C_AF_LAT	LAT address type
GSS_C_AF_HYLINK	NSC Hyperchannel address type
GSS_C_AF_APPLETALK	AppleTalk address type
GSS_C_AF_BSC	BISYNC 2780/3780 address type
GSS_C_AF_DSS	Distributed system services address type
GSS_C_AF_OSI	OSI TP4 address type
GSS_C_AF_X25	X.25
GSS_C_AF_NULLADDR	No address specified

Note that these symbols name address families rather than specific addressing formats. For address families that contain several alternative address forms, the initiator_address and acceptor_address fields must contain sufficient information to determine which address form is used. When not otherwise specified, addresses should be specified in network byte-order (that is, native byte-ordering for the address family).

Conceptually, the GSS-API concatenates the initiator_addrtype, initiator_address, acceptor_addrtype, acceptor_address and application_data to form an octet string. The mechanism calculates a MIC over this octet string, and binds the MIC to the context establishment token emitted by gss_init_sec_context. The same

bindings are presented by the context acceptor to gss_accept_sec_context, and a MIC is calculated in the same way. The calculated MIC is compared with that found in the token, and if the MICs differ, gss_accept_sec_context will return a GSS_S_BAD_BINDINGS error, and the context will not be established. Some mechanisms may include the actual channel binding data in the token (rather than just a MIC); applications should therefore not use confidential data as channel-binding components.

Individual mechanisms may impose additional constraints on addresses and address types that may appear in channel bindings. For example, a mechanism may verify that the initiator_address field of the channel bindings presented to gss_init_sec_context contains the correct network address of the host system. Portable applications should therefore ensure that they either provide correct information for the address fields, or omit addressing information, specifying GSS_C_AF_NULLADDR as the address-types.

5.12.Optional parameters

Various parameters are described as optional. This means that they follow a convention whereby a default value may be requested. The following conventions are used for omitted parameters. These conventions apply only to those parameters that are explicitly documented as optional.

5.12.1. **gss_buffer_t types**

Specify GSS_C_NO_BUFFER as a value. For an input parameter this signifies that default behavior is requested, while for an output parameter it indicates that the information that would be returned via the parameter is not required by the application.

5.12.2. Integer types (input)

Individual parameter documentation lists values to be used to indicate default actions.

5.12.3. Integer types (output)

Specify NULL as the value for the pointer.

5.12.4. Pointer types

Specify NULL as the value.

5.12.5. Object IDs

Specify GSS_C_NO_OID as the value.

5.12.6. Object ID Sets

Specify GSS_C_NO_OID_SET as the value.

5.12.7. Channel Bindings

Specify GSS_C_NO_CHANNEL_BINDINGS to indicate that channel bindings are not to be used.

6. ADDITIONAL CONTROLS

This section discusses the optional services that a context initiator may request of the GSS-API at context establishment. Each of these services is requested by setting a flag in the req_flags input parameter to gss_init_sec_context.

The optional services currently defined are:

Delegation - The (usually temporary) transfer of rights from initiator to acceptor, enabling the acceptor to authenticate itself as an agent of the initiator.

Mutual Authentication - In addition to the initiator authenticating its identity to the context acceptor, the context acceptor should also authenticate itself to the initiator.

Replay detection - In addition to providing message integrity services, gss_get_mic and gss_wrap should include message numbering information to enable gss_verify_mic and gss_unwrap to detect if a message has been duplicated.

Out-of-sequence detection - In addition to providing message integrity services, gss_get_mic and gss_wrap should include message sequencing information to enable gss_verify_mic and gss_unwrap to detect if a message has been received out of sequence.

Anonymous authentication - The establishment of the security context should not reveal the initiator's identity to the context acceptor.

Any currently undefined bits within such flag arguments should be ignored by GSS-API implementations when presented by an application, and should be set to zero when returned to the application by the GSS-API implementation.

Some mechanisms may not support all optional services, and some mechanisms may only support some services in conjunction with others. Both gss_init_sec_context and gss_accept_sec_context inform the applications which services will be available from the context when the establishment phase is complete, via the ret_flags output parameter. In general, if the security mechanism is capable of providing a requested service, it should do so, even if additional services must be enabled in order to provide the requested service. If the mechanism is incapable of providing a requested service, it should proceed without the service, leaving the application to abort the context establishment process if it considers the requested service to be mandatory.

Some mechanisms may specify that support for some services is optional, and that implementors of the mechanism need not provide it.

This is most commonly true of the confidentiality service, often because of legal restrictions on the use of data-encryption, but may apply to any of the services. Such mechanisms are required to send at least one token from acceptor to initiator during context establishment when the initiator indicates a desire to use such a service, so that the initiating GSS-API can correctly indicate whether the service is supported by the acceptor's GSS-API.

6.1. Delegation

The GSS-API allows delegation to be controlled by the initiating application via a boolean parameter to gss_init_sec_context(), the routine that establishes a security context. Some mechanisms do not support delegation, and for such mechanisms attempts by an application to enable delegation are ignored.

The acceptor of a security context for which the initiator enabled delegation will receive (via the delegated_cred_handle parameter of gss_accept_sec_context) a credential handle that contains the delegated identity, and this credential handle may be used to initiate subsequent GSS-API security contexts as an agent or delegate of the initiator. If the original initiator's identity is "A" and the delegate's identity is "B", then, depending on the underlying mechanism, the identity embodied by the delegated credential may be either "A" or "B acting for A".

For many mechanisms that support delegation, a simple boolean does not provide enough control. Examples of additional aspects of delegation control that a mechanism might provide to an application are duration of delegation, network addresses from which delegation is valid, and constraints on the tasks that may be performed by a delegate. Such controls are presently outside the scope of the GSS-API. GSS-API implementations supporting mechanisms offering additional controls should provide extension routines that allow these controls to be exercised (perhaps by modifying the initiator's GSS-API credential prior to its use in establishing a context). However, the simple delegation control provided by GSS-API should always be able to over-ride other mechanism-specific delegation controls - If the application instructs gss_init_sec_context() that delegation is not desired, then the implementation must not permit delegation to occur. This is an exception to the general rule that a mechanism may enable services even if they are not requested -

delegation may only be provided at the explicit request of the application.

6.2. Mutual authentication

Usually, a context acceptor will require that a context initiator authenticate itself so that the acceptor may make an access-control decision prior to performing a service for the initiator. In some cases, the initiator may also request that the acceptor authenticate itself. GSS-API allows the initiating application to request this mutual authentication service by setting a flag when calling gss_init_sec_context.

The initiating application is informed as to whether or not the context acceptor has authenticated itself. Note that some mechanisms may not support mutual authentication, and other mechanisms may always perform mutual authentication, whether or not the initiating application requests it. In particular, mutual authentication my be required by some mechanisms in order to support replay or out-of-sequence message detection, and for such mechanisms a request for either of these services will automatically enable mutual authentication.

6.3. Replay and out-of-sequence detection

The GSS-API may provide detection of mis-ordered message once a security context has been established. Protection may be applied to messages by either application, by calling either gss_get_mic or gss_wrap, and verified by the peer application by calling gss_verify_mic or gss_unwrap.

gss_get_mic calculates a cryptographic MIC over an application message, and returns that MIC in a token. The application should pass both the token and the message to the peer application, which presents them to gss_verify_mic.

gss_wrap calculates a cryptographic MIC of an application message, and places both the MIC and the message inside a single token. The Application should pass the token to the peer application, which presents it to gss_unwrap to extract the message and verify the MIC.

Either pair of routines may be capable of detecting out-of-sequence message delivery, or duplication of messages. Details of such mis-ordered messages are indicated through supplementary status bits in the major status code returned by gss_verify_mic or gss_unwrap. The relevant supplementary bits are:

GSS_S_DUPLICATE_TOKEN - The token is a duplicate of one that has already been received and processed. Only contexts that claim to provide replay detection may set this bit.

GSS_S_OLD_TOKEN - The token is too old to determine whether or not it is a duplicate. Contexts supporting out-of-sequence detection but not replay detection should always set this bit if GSS_S_UNSEQ_TOKEN is set; contexts that support replay detection should only set this bit if the

token is so old that it cannot be checked for duplication.

GSS_S_UNSEQ_TOKEN - A later token has already been processed.

GSS_S_GAP_TOKEN - An earlier token has not yet been received.

A mechanism need not maintain a list of all tokens that have been processed in order to support these status codes. A typical mechanism might retain information about only the most recent "N" tokens processed, allowing it to distinguish duplicates and missing tokens within the most recent "N" messages; the receipt of a token older than the most recent "N" would result in a GSS_S_OLD_TOKEN status.

6.4. Anonymous Authentication

In certain situations, an application may wish to initiate the authentication process to authenticate a peer, without revealing its own identity. As an example, consider an application providing access to a database containing medical information, and offering unrestricted access to the service. A client of such a service might wish to authenticate the service (in order to establish trust in any information retrieved from it), but might not wish the service to be able to obtain the client's identity (perhaps due to privacy concerns about the specific inquiries, or perhaps simply to avoid being placed on mailing-lists).

In normal use of the GSS-API, the initiator's identity is made available to the acceptor as a result of the context establishment process. However, context initiators may request that their identity not be revealed to the context acceptor. Many mechanisms do not support anonymous authentication, and for such mechanisms the request will not be honored. An authentication token will be still be generated, but the application is always informed if a requested service is unavailable, and has the option to abort context establishment if anonymity is valued above the other security services that would require a context to be established.

In addition to informing the application that a context is established anonymously (via the ret_flags outputs from gss_init_sec_context and gss_accept_sec_context), the optional src_name output from gss_accept_sec_context and gss_inquire_context will, for such contexts, return a reserved internal-form name, defined by the implementation.

When presented to gss_display_name, this reserved internal-form name will result in a printable name that is syntactically distinguishable from any valid principal name supported by the implementation, associated with a name-type object identifier with the value GSS_C_NT_ANONYMOUS, whose value us given in Appendix A. The printable form of an anonymous name should be chosen such that it implies anonymity, since this name may appear in, for example, audit logs. For example, the string "<anonymous>" might be a good choice, if no valid printable names supported by the implementation can begin with "<" and end with ">".

6.5. Confidentiality

If a context supports the confidentiality service, gss_wrap may be used to encrypt application messages. Messages are selectively encrypted, under the control of the conf_req_flag input parameter to gss_wrap.

6.6. Inter-process context transfer

GSS-API V2 provides routines (gss_export_sec_context and gss_import_sec_context) which allow a security context to be transferred between processes on a single machine. The most common use for such a feature is a client-server design where the server is implemented as a single process that accepts incoming security contexts, which then launches child processes to deal with the data on these contexts. In such a design, the child processes must have access to the security context data structure created within the parent by its call to gss_accept_sec_context so that they can use per-message protection services and delete the security context when the communication session ends.

Since the security context data structure is expected to contain sequencing information, it is impractical in general to share a context between processes. Thus GSS-API provides a call (gss_export_sec_context) that the process which currently owns the context can call to declare that it has no intention to use the context subsequently, and to create an inter-process token containing information needed by the adopting process to successfully import the context. After successful completion of gss_export_sec_context, the original security context is made inaccessible to the calling process by GSS-API, and any context handles referring to this context are no longer valid. The originating process transfers the inter-process token to the adopting process, which passes it to gss_import_sec_context, and a fresh gss_ctx_id_t is created such that it is functionally identical to the original context.

The inter-process token may contain sensitive data from the original security context (including cryptographic keys). Applications using inter-process tokens to transfer security contexts must take appropriate steps to protect these tokens in transit.

Implementations are not required to support the inter-process transfer of security contexts. The ability to transfer a security context is indicated when the context is created, by gss_init_sec_context or gss_accept_sec_context setting the GSS_C_TRANS_FLAG bit in their ret_flags parameter.

6.7. The use of incomplete contexts

Some mechanisms may allow the per-message services to be used before the context establishment process is complete. For example, a

mechanism may include sufficient information in its initial context-level token for the context acceptor to immediately decode messages protected with gss_wrap or gss_get_mic. For such a mechanism, the initiating application need not wait until subsequent context-level tokens have been sent and received before invoking the per-message protection services.

The ability of a context to provide per-message services in advance of complete context establishment is indicated by the setting of the GSS_C_PROT_READY_FLAG bit in the ret_flags parameter from gss_init_sec_context and gss_accept_sec_context. Applications wishing to use per-message protection services on partially-established contexts should check this flag before attempting to invoke gss_wrap or gss_get_mic.

7. GSS-API routine descriptions

In addition to the explicit major status codes documented here, the code GSS_S_FAILURE may be returned by any routine, indicating an implementation-specific or mechanism-specific error condition, further details of which are reported via the minor_status parameter.

7.1. gss_accept_sec_context

```
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,
  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)
```

Purpose:

Allows a remotely initiated security context between the application and a remote peer to be established. The routine may return a output_token which should be transferred to the peer application, where the peer application will present it to gss_init_sec_context. If no token need be sent, gss_accept_sec_context will indicate this by setting the length field of the output_token argument to zero. To complete the context establishment, one or more reply tokens may be required from the peer application; if so, gss_accept_sec_context will return a status flag of GSS_S_CONTINUE_NEEDED, in which case it should be called again when the reply token is received from the peer application, passing the token to gss_accept_sec_context via the input_token parameters.

Portable applications should be constructed to use the token length and return status to determine whether a token needs to be sent or waited for. Thus a typical portable caller should always invoke gss_accept_sec_context within a loop:

```
gss_ctx_id_t context_hdl = GSS_C_NO_CONTEXT;

do {
  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);
  if (GSS_ERROR(maj_stat)) {
    report_error(maj_stat, min_stat);
  };
  if (output_token->length != 0) {
    send_token_to_peer(output_token);
```

```
    gss_release_buffer(&min_stat, output_token);
  };
  if (GSS_ERROR(maj_stat)) {
    if (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);
```

Whenever the routine returns a major status that includes the value GSS_S_CONTINUE_NEEDED, the context is not fully established and the following restrictions apply to the output parameters:

The value returned via the time_rec parameter is undefined Unless the accompanying ret_flags parameter contains the bit GSS_C_PROT_READY_FLAG, indicating that per-message services may be applied in advance of a successful completion status, the value returned via the mech_type parameter may be undefined until the routine returns a major status value of GSS_S_COMPLETE.

The values of the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG,GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG,GSS_C_INTEG_FLAG and GSS_C_ANON_FLAG bits returned via the ret_flags parameter should contain the values that the implementation expects would be valid if context establishment were to succeed.

The values of the GSS_C_PROT_READY_FLAG and GSS_C_TRANS_FLAG bits within ret_flags should indicate the actual state at the time gss_accept_sec_context returns, whether or not the context is fully established.

Although this requires that GSS-API implementations set the GSS_C_PROT_READY_FLAG in the final ret_flags returned to a caller (i.e. when accompanied by a GSS_S_COMPLETE status code), applications should not rely on this behavior as the flag was not defined in Version 1 of the GSS-API. Instead, applications should be prepared to use per-message services after a successful context establishment, according to the GSS_C_INTEG_FLAG and GSS_C_CONF_FLAG values.

All other bits within the ret_flags argument should be set to zero. While the routine returns GSS_S_CONTINUE_NEEDED, the values returned via the ret_flags argument indicate the services that the implementation expects to be available from the established context.

If the initial call of gss_accept_sec_context() fails, the implementation should not create a context object, and should leave the value of the context_handle parameter set to GSS_C_NO_CONTEXT to indicate this. In the event of a failure on a subsequent call, the implementation is permitted to delete the "half-built" security context (in which case it should set the context_handle parameter to GSS_C_NO_CONTEXT), but the preferred behavior is to leave the

security context (and the context_handle parameter) untouched for the application to delete (using gss_delete_sec_context).

During context establishment, the informational status bits GSS_S_OLD_TOKEN and GSS_S_DUPLICATE_TOKEN indicate fatal errors, and GSS-API mechanisms should always return them in association with a routine error of GSS_S_FAILURE. This requirement for pairing did not exist in version 1 of the GSS-API specification, so applications that wish to run over version 1 implementations must special-case these codes.

Parameters:

```
   context_handle    gss_ctx_id_t, read/modify
                     context handle for new context.  Supply
                     GSS_C_NO_CONTEXT for first call; use value
                     returned in subsequent calls.  Once
                     gss_accept_sec_context() has returned a
                     value via this parameter, resources have
                     been assigned to the corresponding context,
                     and must be freed by the application after
                     use with a call to gss_delete_sec_context().


acceptor_cred_handle  gss_cred_id_t, read
                     Credential handle claimed by context
                     acceptor. Specify GSS_C_NO_CREDENTIAL to
                     accept the context as a default principal.
                     If GSS_C_NO_CREDENTIAL is specified, but
                     no default acceptor principal is defined,
                     GSS_S_NO_CRED will be returned.
```

input_token_buffer   buffer, opaque, read
token obtained from remote application.

input_chan_bindings   channel bindings, read, optional
Application-specified bindings. Allows application to securely bind channel identification information to the security context. If channel bindings are not used, specify GSS_C_NO_CHANNEL_BINDINGS.

src_name   gss_name_t, modify, optional
Authenticated name of context initiator. After use, this name should be deallocated by passing it to gss_release_name(). If not required, specify NULL.

mech_type   Object ID, modify, optional
Security mechanism used. The returned OID value will be a pointer into static storage, and should be treated as read-only by the caller (in particular, it does not need to be freed). If not required, specify NULL.

output_token   buffer, opaque, modify
Token to be passed to peer application. If the length field of the returned token buffer is 0, then no token need be passed to the peer application. If a non-zero length field is returned, the associated storage must be freed after use by the application with a call to gss_release_buffer().

ret_flags   bit-mask, modify, optional
Contains various independent flags, each of which indicates that the context supports a specific service option. If not needed, specify NULL. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically-ANDed with the ret_flags value to test whether a given option is

supported by the context. The flags are:

GSS_C_DELEG_FLAG

True - Delegated credentials are available via the delegated_cred_handle parameter

False - No credentials were delegated

GSS_C_MUTUAL_FLAG

True - Remote peer asked for mutual authentication

False - Remote peer did not ask for mutual authentication

GSS_C_REPLAY_FLAG

True - replay of protected messages will be detected

False - replayed messages will not be detected

GSS_C_SEQUENCE_FLAG

True - out-of-sequence protected messages will be detected

False - out-of-sequence messages will not be detected

GSS_C_CONF_FLAG

True - Confidentiality service may be invoked by calling the gss_wrap routine

False - No confidentiality service (via gss_wrap) available. gss_wrap will provide message encapsulation, data-origin authentication and integrity services only.

GSS_C_INTEG_FLAG

True - Integrity service may be invoked by calling either gss_get_mic or gss_wrap routines.

False - Per-message integrity service

unavailable.

GSS_C_ANON_FLAG

True - The initiator does not wish to be authenticated; the src_name parameter (if requested) contains an anonymous internal name.

False - The initiator has been authenticated normally.

```
                     GSS_C_PROT_READY_FLAG
                     True - Protection services (as specified
                            by the states of the GSS_C_CONF_FLAG
                            and GSS_C_INTEG_FLAG) are available
                            if the accompanying major status
                            return value is either GSS_S_COMPLETE
                            or GSS_S_CONTINUE_NEEDED.
                     False - Protection services (as specified
                             by the states of the GSS_C_CONF_FLAG
                             and GSS_C_INTEG_FLAG) are available
                             only if the accompanying major status
                             return value is GSS_S_COMPLETE.
                     GSS_C_TRANS_FLAG
                     True - The resultant security context may
                            be transferred to other processes via
                            a call to gss_export_sec_context().
                     False - The security context is not
                             transferrable.
                     All other bits should be set to zero.

time_rec             Integer, modify, optional
                     number of seconds for which the context
                     will remain valid. Specify NULL if
                     not required.

delegated_cred_handle
                     gss_cred_id_t, modify, optional
                     credential handle for credentials received
                     from context initiator.  Only valid if
                     deleg_flag in ret_flags is true, in which
                     case an explicit credential handle
                     (i.e. not GSS_C_NO_CREDENTIAL) will be
                     returned; if deleg_flag is false,
                     gss_accept_context() will set this parameter
                     to GSS_C_NO_CREDENTIAL.  If a credential
                     handle is returned, the associated resources
                     must be released by the application after use
                     with a call to gss_release_cred().  Specify
                     NULL if not required.

minor_status         Integer, modify
                     Mechanism specific status code.


Function value:  GSS status code

GSS_S_COMPLETE    Successful completion
```

GSS_S_CONTINUE_NEEDED Indicates that a token from the peer application is required to complete the context, and that gss_accept_sec_context must be called again with that token.

GSS_S_DEFECTIVE_TOKEN Indicates that consistency checks performed on the input_token failed.

GSS_S_DEFECTIVE_CREDENTIAL Indicates that consistency checks performed on the credential failed.

GSS_S_NO_CRED The supplied credentials were not valid for context acceptance, or the credential handle did not reference any credentials.

GSS_S_CREDENTIALS_EXPIRED The referenced credentials have expired.

GSS_S_BAD_BINDINGS The input_token contains different channel bindings to those specified via the input_chan_bindings parameter.

GSS_S_NO_CONTEXT Indicates that the supplied context handle did not refer to a valid context.

GSS_S_BAD_SIG The input_token contains an invalid MIC.

GSS_S_OLD_TOKEN The input_token was too old. This is a fatal error during context establishment.

GSS_S_DUPLICATE_TOKEN The input_token is valid, but is a duplicate of a token already processed. This is a fatal error during context establishment.

GSS_S_BAD_MECH The received token specified a mechanism that is not supported by the implementation or the provided credential.

7.2. 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)
```

Purpose:

Allows an application to acquire a handle for a pre-existing credential by name. GSS-API implementations must impose a local access-control policy on callers of this routine to prevent unauthorized callers from acquiring credentials to which they are not entitled. This routine is not intended to provide a "login to the network" function, as such a function would involve the creation of new credentials rather than merely acquiring a handle to existing credentials. Such functions, if required, should be defined in implementation-specific extensions to the API.

If desired_name is GSS_C_NO_NAME, the call is interpreted as a request for a credential handle that will invoke default behavior when passed to gss_init_sec_context() (if cred_usage is GSS_C_INITIATE or GSS_C_BOTH) or gss_accept_sec_context() (if cred_usage is GSS_C_ACCEPT or GSS_C_BOTH).

Mechanisms should honor the desired_mechs parameter, and return a credential that is suitable to use only with the requested mechanisms. An exception to this is the case where one underlying credential element can be shared by multiple mechanisms; in this case it is permissible for an implementation to indicate all mechanisms with which the credential element may be used. If desired_mechs is an empty set, behavior is undefined.

This routine is expected to be used primarily by context acceptors, since implementations are likely to provide mechanism-specific ways of obtaining GSS-API initiator credentials from the system login process. Some implementations may therefore not support the acquisition of GSS_C_INITIATE or GSS_C_BOTH credentials via gss_acquire_cred for any name other than GSS_C_NO_NAME, or a name produced by applying either gss_inquire_cred to a valid credential, or gss_inquire_context to an active context.

If credential acquisition is time-consuming for a mechanism, the mechanism may choose to delay the actual acquisition until the credential is required (e.g. by gss_init_sec_context or gss_accept_sec_context). Such mechanism-specific implementation decisions should be invisible to the calling application; thus a call of gss_inquire_cred immediately following the call of gss_acquire_cred must return valid credential data, and may therefore incur the overhead of a deferred credential acquisition.

Parameters:

```
desired_name      gss_name_t, read
                  Name of principal whose credential
                  should be acquired

time_req          Integer, read, optional
                  number of seconds that credentials
                  should remain valid. Specify GSS_C_INDEFINITE
                  to request that the credentials have the maximum
                  permitted lifetime.

desired_mechs     Set of Object IDs, read, optional
                  set of underlying security mechanisms that
                  may be used.  GSS_C_NO_OID_SET may be used
                  to obtain an implementation-specific default.

cred_usage        gss_cred_usage_t, read
                  GSS_C_BOTH - Credentials may be used
                     either to initiate or accept
                     security contexts.
                  GSS_C_INITIATE - Credentials will only be
                     used to initiate security contexts.
                  GSS_C_ACCEPT - Credentials will only be used to
```

accept security contexts.

output_cred_handle gss_cred_id_t, modify
The returned credential handle. Resources
associated with this credential handle must
be released by the application after use
with a call to gss_release_cred().

actual_mechs Set of Object IDs, modify, optional
The set of mechanisms for which the
credential is valid. Storage associated
with the returned OID-set must be released by
the application after use with a call to
gss_release_oid_set(). Specify NULL if not
required.

time_rec Integer, modify, optional
Actual number of seconds for which the
returned credentials will remain valid. If the
implementation does not support expiration of
credentials, the value GSS_C_INDEFINITE will
be returned. Specify NULL if not required

minor_status Integer, modify
Mechanism specific status code.

Function value: GSS status code

GSS_S_COMPLETE Successful completion

GSS_S_BAD_MECH Unavailable mechanism requested

GSS_S_BAD_NAMETYPE Type contained within desired_name parameter
is not supported

GSS_S_BAD_NAME Value supplied for desired_name parameter is ill
formed.

GSS_S_CREDENTIALS_EXPIRED The credentials could not be acquired
Because they have expired.

GSS_S_NO_CRED No credentials were found for the specified name.

7.3. 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)
```

Purpose:

Adds a credential-element to a credential. The credential-element is identified by the name of the principal to which it refers. GSS-API implementations must impose a local access-control policy on callers of this routine to prevent unauthorized callers from acquiring credential-elements to which they are not entitled. This routine is not intended to provide a "login to the network" function, as such a function would involve the creation of new mechanism-specific authentication data, rather than merely acquiring a GSS-API handle to existing data. Such functions, if required, should be defined in implementation-specific extensions to the API.

If desired_name is GSS_C_NO_NAME, the call is interpreted as a request to add a credential element that will invoke default behavior when passed to gss_init_sec_context() (if cred_usage is GSS_C_INITIATE or GSS_C_BOTH) or gss_accept_sec_context() (if cred_usage is GSS_C_ACCEPT or GSS_C_BOTH).

This routine is expected to be used primarily by context acceptors, since implementations are likely to provide mechanism-specific ways of obtaining GSS-API initiator credentials from the system login process. Some implementations may therefore not support the acquisition of GSS_C_INITIATE or GSS_C_BOTH credentials via gss_acquire_cred for any name other than GSS_C_NO_NAME, or a name produced by applying either gss_inquire_cred to a valid credential, or gss_inquire_context to an active context.

If credential acquisition is time-consuming for a mechanism, the mechanism may choose to delay the actual acquisition until the credential is required (e.g. by gss_init_sec_context or gss_accept_sec_context). Such mechanism-specific implementation decisions should be invisible to the calling application; thus a call of gss_inquire_cred immediately following the call of gss_add_cred must return valid credential data, and may therefore incur the overhead of a deferred credential acquisition.

This routine can be used to either compose a new credential containing all credential-elements of the original in addition to the newly-acquire credential-element, or to add the new credential-element to an existing credential. If NULL is specified for the output_cred_handle parameter argument, the new credential-element will be added to the credential identified by input_cred_handle; if a valid pointer is specified for the output_cred_handle parameter, a new credential handle will be created.

If GSS_C_NO_CREDENTIAL is specified as the input_cred_handle, gss_add_cred will compose a credential (and set the output_cred_handle parameter accordingly) based on default behavior. That is, the call will have the same effect as if the application had first made a call to gss_acquire_cred(), specifying the same usage and passing GSS_C_NO_NAME as the desired_name parameter to obtain an explicit credential handle embodying default behavior, passed this credential handle to gss_add_cred(), and finally called gss_release_cred() on the first credential handle.

If GSS_C_NO_CREDENTIAL is specified as the input_cred_handle parameter, a non-NULL output_cred_handle must be supplied.

Parameters:

minor_status   Integer, modify
Mechanism specific status code.

input_cred_handle gss_cred_id_t, read, optional
The credential to which a credential-element will be added. If GSS_C_NO_CREDENTIAL is specified, the routine will compose the new credential based on default behavior (see description above). Note that, while the credential-handle is not modified by gss_add_cred(), the underlying credential will be modified if output_credential_handle is NULL.

desired_name   gss_name_t, read.
Name of principal whose credential should be acquired.

desired_mech   Object ID, read
Underlying security mechanism with which the credential may be used.

cred_usage   gss_cred_usage_t, read
GSS_C_BOTH - Credential may be used either to initiate or accept security contexts.

GSS_C_INITIATE - Credential will only be used to initiate security contexts.
GSS_C_ACCEPT - Credential will only be used to accept security contexts.

initiator_time_req Integer, read, optional

number of seconds that the credential should remain valid for initiating security contexts. This argument is ignored if the composed credentials are of type GSS_C_ACCEPT. Specify GSS_C_INDEFINITE to request that the credentials have the maximum permitted initiator lifetime.

acceptor_time_req Integer, read, optional
number of seconds that the credential should remain valid for accepting security contexts. This argument is ignored if the composed credentials are of type GSS_C_INITIATE. Specify GSS_C_INDEFINITE to request that the credentials have the maximum permitted initiator lifetime.

output_cred_handle gss_cred_id_t, modify, optional
The returned credential handle, containing the new credential-element and all the credential-elements from input_cred_handle. If a valid pointer to a gss_cred_id_t is supplied for this parameter, gss_add_cred creates a new credential handle containing all credential-elements from the input_cred_handle and the newly acquired credential-element; if NULL is specified for this parameter, the newly acquired credential-element will be added to the credential identified by input_cred_handle.

The resources associated with any credential handle returned via this parameter must be released by the application after use with a call to gss_release_cred().

actual_mechs Set of Object IDs, modify, optional
The complete set of mechanisms for which

the new credential is valid. Storage for the returned OID-set must be freed by the application after use with a call to gss_release_oid_set(). Specify NULL if not required.

initiator_time_rec Integer, modify, optional
Actual number of seconds for which the returned credentials will remain valid for initiating contexts using the specified mechanism. If the implementation or mechanism does not support expiration of credentials, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required

acceptor_time_rec Integer, modify, optional
Actual number of seconds for which the returned credentials will remain valid for accepting security contexts using the specified mechanism. If the implementation or mechanism does not support expiration of credentials, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required

Function value: GSS status code

GSS_S_COMPLETE Successful completion

GSS_S_BAD_MECH Unavailable mechanism requested

GSS_S_BAD_NAMETYPE Type contained within desired_name parameter is not supported

GSS_S_BAD_NAME Value supplied for desired_name parameter is ill-formed.

GSS_S_DUPLICATE_ELEMENT The credential already contains an element for the requested mechanism with overlapping usage and validity period.

GSS_S_CREDENTIALS_EXPIRED The required credentials could not be added because they have expired.

GSS_S_NO_CRED No credentials were found for the specified name.

7.4. 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)
```

Purpose:

Add an Object Identifier to an Object Identifier set. This routine is intended for use in conjunction with gss_create_empty_oid_set when constructing a set of mechanism OIDs for input to gss_acquire_cred. The oid_set parameter must refer to an OID-set that was created by GSS-API (e.g. a set returned by gss_create_empty_oid_set()). GSS-API creates a copy of the member_oid and inserts this copy into the set, expanding the storage allocated to the OID-set's elements array if necessary. The routine may add the new member OID anywhere within the elements array, and implementations should verify that the new member_oid is not already contained within the elements array; if the member_oid is already present, the oid_set should remain unchanged.

Parameters:

```
   minor_status      Integer, modify
                     Mechanism specific status code

   member_oid        Object ID, read
                     The object identifier to copied into
                     the set.

   oid_set           Set of Object ID, modify
                     The set in which the object identifier
                     should be inserted.
```

Function value: GSS status code

```
   GSS_S_COMPLETE    Successful completion
```

7.5. gss_canonicalize_name

```
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)
```

Purpose:

Generate a canonical mechanism name (MN) from an arbitrary internal name. The mechanism name is the name that would be returned to a context acceptor on successful authentication of a context where the initiator used the input_name in a successful call to gss_acquire_cred, specifying an OID set containing <mech_type> as its only member, followed by a call to gss_init_sec_context, specifying <mech_type> as the authentication mechanism.

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code

input_name   gss_name_t, read
                  The name for which a canonical form is
                  desired

mech_type   Object ID, read
                  The authentication mechanism for which the
                  canonical form of the name is desired. The
                  desired mechanism must be specified explicitly;
                  no default is provided.

output_name   gss_name_t, modify
                  The resultant canonical name. Storage
                  associated with this name must be freed by
                  the application after use with a call to
                  gss_release_name().

Function value: GSS status code

GSS_S_COMPLETE   Successful completion.

GSS_S_BAD_MECH   The identified mechanism is not supported.

GSS_S_BAD_NAMETYPE The provided internal name contains no elements
                  that could be processed by the specified
                  mechanism.

GSS_S_BAD_NAME   The provided internal name was ill-formed.

7.6. gss_compare_name

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

Purpose:

Allows an application to compare two internal-form names to determine whether they refer to the same entity.

If either name presented to gss_compare_name denotes an anonymous principal, the routines should indicate that the two names do not refer to the same identity.

Parameters:

```
minor_status      Integer, modify
                  Mechanism specific status code.

name1             gss_name_t, read
                  internal-form name

name2             gss_name_t, read
                  internal-form name

name_equal        boolean, modify
                  non-zero - names refer to same entity
                  zero - names refer to different entities
                        (strictly, the names are not known
                        to refer to the same identity).

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion

GSS_S_BAD_NAMETYPE The two names were of incomparable types.

GSS_S_BAD_NAME    One or both of name1 or name2 was ill-formed.
```

7.7. gss_context_time

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

Purpose:

Determines the number of seconds for which the specified context will remain valid.

Parameters:

minor_status Integer, modify
Implementation specific status code.

context_handle gss_ctx_id_t, read
Identifies the context to be interrogated.

time_rec Integer, modify
Number of seconds that the context will remain valid. If the context has already expired, zero will be returned.

Function value: GSS status code

GSS_S_COMPLETE Successful completion

GSS_S_CONTEXT_EXPIRED The context has already expired

GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context

7.8. gss_create_empty_oid_set

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

Purpose:

Create an object-identifier set containing no object identifiers, to which members may be subsequently added using the gss_add_oid_set_member() routine. These routines are intended to be used to construct sets of mechanism object identifiers, for input to gss_acquire_cred.

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code

oid_set   Set of Object IDs, modify
                  The empty object identifier set.
                  The routine will allocate the
                  gss_OID_set_desc object, which the
                  application must free after use with
                  a call to gss_release_oid_set().

Function value: GSS status code

GSS_S_COMPLETE   Successful completion

7.9. 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)
```

Purpose:

Delete a security context. gss_delete_sec_context will delete the local data structures associated with the specified security context, and may generate an output_token, which when passed to the peer gss_process_context_token will instruct it to do likewise. If no token is required by the mechanism, the GSS-API should set the length field of the output_token (if provided) to zero. No further security services may be obtained using the context specified by context_handle.

In addition to deleting established security contexts, gss_delete_sec_context must also be able to delete "half-built" security contexts resulting from an incomplete sequence of gss_init_sec_context()/gss_accept_sec_context() calls.

The output_token parameter is retained for compatibility with version 1 **of the GSS-API.** It is recommended that both peer applications invoke gss_delete_sec_context passing the value GSS_C_NO_BUFFER for the output_token parameter, indicating that no token is required, and that gss_delete_sec_context should simply delete local context data structures. If the application does pass a valid buffer to gss_delete_sec_context, mechanisms are encouraged to return a zero-length token, indicating that no peer action is necessary, and that no token should be transferred by the application.

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code.

context_handle   gss_ctx_id_t, modify
                  context handle identifying context to delete.
                  After deleting the context, the GSS-API will set
                  this context handle to GSS_C_NO_CONTEXT.

output_token   buffer, opaque, modify, optional
                  token to be sent to remote application to
                  instruct it to also delete the context. It
                  is recommended that applications specify
                  GSS_C_NO_BUFFER for this parameter, requesting
                  local deletion only. If a buffer parameter is
                  provided by the application, the mechanism may
                  return a token in it; mechanisms that implement
                  only local deletion should set the length field of
                  this token to zero to indicate to the application
                  that no token is to be sent to the peer.

Function value: GSS status code

GSS_S_COMPLETE   Successful completion

GSS_S_NO_CONTEXT No valid context was supplied

7.10.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)
```

Purpose:

Allows an application to obtain a textual representation of an opaque internal-form name for display purposes. The syntax of a printable name is defined by the GSS-API implementation.

If input_name denotes an anonymous principal, the implementation should return the gss_OID value GSS_C_NT_ANONYMOUS as the output_name_type, and a textual name that is syntactically distinct from all valid supported printable names in output_name_buffer.

If input_name was created by a call to gss_import_name, specifying GSS_C_NO_OID as the name-type, implementations that employ lazy conversion between name types may return GSS_C_NO_OID via the output_name_type parameter.

Parameters:

minor_status	Integer, modify Mechanism specific status code.
input_name	gss_name_t, read name to be displayed
output_name_buffer	buffer, character-string, modify buffer to receive textual name string. The application must free storage associated with this name after use with a call to gss_release_buffer().
output_name_type	Object ID, modify, optional The type of the returned name. The returned gss_OID will be a pointer into static storage, and should be treated as read-only by the caller (in particular, the application should not attempt to free it). Specify NULL if not required.

Function value:	GSS status code
GSS_S_COMPLETE	Successful completion
GSS_S_BAD_NAME	input_name was ill-formed

7.11.gss_display_status

```
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)
```

Purpose:

Allows an application to obtain a textual representation of a GSS-API status code, for display to the user or for logging purposes. Since some status values may indicate multiple conditions, applications may need to call gss_display_status multiple times, each call generating a single text string. The message_context parameter is used by gss_display_status to store state information about which error messages have already been extracted from a given status_value; message_context must be initialized to 0 by the application prior to the first call, and gss_display_status will return a non-zero value in this parameter if there are further messages to extract.

The message_context parameter contains all state information required by gss_display_status in order to extract further messages from the status_value; even when a non-zero value is returned in this parameter, the application is not required to call gss_display_status again unless subsequent messages are desired. The following code extracts all messages from a given status code and prints them to 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;
```

```
do {

  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);

} while (message_context != 0);
```

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code.

status_value   Integer, read
                  Status value to be converted

status_type   Integer, read
                  GSS_C_GSS_CODE - status_value is a GSS status
                  code

GSS_C_MECH_CODE - status_value is a mechanism
                  status code

mech_type   Object ID, read, optional
                  Underlying mechanism (used to interpret a
                  minor status value) Supply GSS_C_NO_OID to
                  obtain the system default.

message_context   Integer, read/modify
                  Should be initialized to zero by the
                  application prior to the first call.
                  On return from gss_display_status(),

a non-zero status_value parameter indicates that additional messages may be extracted from the status code via subsequent calls to gss_display_status(), passing the same status_value, status_type, mech_type, and message_context parameters.

status_string   buffer, character string, modify
textual interpretation of the status_value. Storage associated with this parameter must be freed by the application after use with a call to gss_release_buffer().

Function value: GSS status code

GSS_S_COMPLETE   Successful completion

GSS_S_BAD_MECH   Indicates that translation in accordance with an unsupported mechanism type was requested

GSS_S_BAD_STATUS   The status value was not recognized, or the status type was neither GSS_C_GSS_CODE nor GSS_C_MECH_CODE.

7.12.gss_duplicate_name

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

Purpose:

Create an exact duplicate of the existing internal name src_name. The new dest_name will be independent of src_name (i.e. src_name and dest_name must both be released, and the release of one shall not affect the validity of the other).

Parameters:

minor_status   Integer, modify
Mechanism specific status code.

src_name   gss_name_t, read

```
                  internal name to be duplicated.

dest_name         gss_name_t, modify
                  The resultant copy of <src_name>.
                  Storage associated with this name must
                  be freed by the application after use
                  with a call to gss_release_name().

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion

GSS_S_BAD_NAME    The src_name parameter was ill-formed.
```

7.13.gss_export_name

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

Purpose:

To produce a canonical contiguous string representation of a
mechanism name (MN), suitable for direct comparison (e.g. with

memcmp) for use in authorization functions (e.g. matching entries in an access-control list).
The <input_name> parameter must specify a valid MN (i.e. an internal name generated by gss_accept_sec_context or by gss_canonicalize_name).

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code

input_name   gss_name_t, read
                  The MN to be exported

exported_name   gss_buffer_t, octet-string, modify
                  The canonical contiguous string form of <input_name>. Storage associated with this string must freed by the application after use with gss_release_buffer().

Function value: GSS status code

GSS_S_COMPLETE   Successful completion

GSS_S_NAME_NOT_MN The provided internal name was not a mechanism name.

GSS_S_BAD_NAME   The provided internal name was ill-formed.

GSS_S_BAD_NAMETYPE The internal name was of a type not supported by the GSS-API implementation.

7.14.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)
```

Purpose:

Provided to support the sharing of work between multiple processes. This routine will typically be used by the context-acceptor, in an application where a single process receives incoming connection requests and accepts security contexts over them, then passes the established context to one or more other processes for message exchange. gss_export_sec_context() deactivates the security context for the calling process and creates an interprocess token which, when passed to gss_import_sec_context in another process, will re-activate the context in the second process. Only a single instantiation of a given context may be active at any one time; a subsequent attempt by a context exporter to access the exported security context will fail.

The implementation may constrain the set of processes by which the interprocess token may be imported, either as a function of local security policy, or as a result of implementation decisions. For example, some implementations may constrain contexts to be passed only between processes that run under the same account, or which are part of the same process group.

The interprocess token may contain security-sensitive information (for example cryptographic keys). While mechanisms are encouraged to either avoid placing such sensitive information within interprocess tokens, or to encrypt the token before returning it to the application, in a typical object-library GSS-API implementation this may not be possible. Thus the application must take care to protect the interprocess token, and ensure that any process to which the token is transferred is trustworthy.

If creation of the interprocess token is succesful, the implementation shall deallocate all process-wide resources associated with the security context, and set the context_handle to GSS_C_NO_CONTEXT. In the event of an error that makes it impossible to complete the export of the security context, the implementation must not return an interprocess token, and should strive to leave the security context referenced by the context_handle parameter untouched. If this is impossible, it is permissible for the implementation to delete the security context, providing it also sets the context_handle parameter to GSS_C_NO_CONTEXT.

Parameters:

minor_status	Integer, modify Mechanism specific status code

context_handle gss_ctx_id_t, modify
context handle identifying the context to

transfer.

interprocess_token buffer, opaque, modify
token to be transferred to target process.
Storage associated with this token must be
freed by the application after use with a
call to gss_release_buffer().

Function value: GSS status code

GSS_S_COMPLETE Successful completion

GSS_S_CONTEXT_EXPIRED The context has expired

GSS_S_NO_CONTEXT The context was invalid

GSS_S_UNAVAILABLE The operation is not supported.

7.15.gss_get_mic

```
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       msg_token)
```

Purpose:

Generates a cryptographic MIC for the supplied message, and places the MIC in a token for transfer to the peer application. The qop_req parameter allows a choice between several cryptographic algorithms, if supported by the chosen mechanism.

Since some application-level protocols may wish to use tokens emitted by gss_wrap() to provide "secure framing", implementations must support derivation of MICs from zero-length messages.

Parameters:

```
minor_status      Integer, modify
                  Implementation specific status code.

context_handle    gss_ctx_id_t, read
                  identifies the context on which the message
                  will be sent

qop_req           gss_qop_t, read, optional
                  Specifies requested quality of protection.
                  Callers are encouraged, on portability grounds,
```

to accept the default quality of protection offered by the chosen mechanism, which may be requested by specifying GSS_C_QOP_DEFAULT for this parameter. If an unsupported protection strength is requested, gss_get_mic will return a major_status of GSS_S_BAD_QOP.

message_buffer buffer, opaque, read
message to be protected

msg_token buffer, opaque, modify
buffer to receive token. The application must free storage associated with this buffer after use with a call to gss_release_buffer().

Function value: GSS status code

GSS_S_COMPLETE Successful completion

GSS_S_CONTEXT_EXPIRED The context has already expired

GSS_S_NO_CONTEXT The context_handle parameter did not identify

a valid context

GSS_S_BAD_QOP The specified QOP is not supported by the mechanism.

7.16.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)
```

Purpose:

Convert a contiguous string name to internal form. In general, the internal name returned (via the <output_name> parameter) will not be an MN; the exception to this is if the <input_name_type> indicates that the contiguous string provided via the <input_name_buffer> parameter is of type GSS_C_NT_EXPORT_NAME, in which case the returned internal name will be an MN for the mechanism that exported the name.

Parameters:

```
minor_status      Integer, modify
                  Mechanism specific status code

input_name_buffer  buffer, octet-string, read
                  buffer containing contiguous string name to convert

input_name_type   Object ID, read, optional
                  Object ID specifying type of printable
                  name.  Applications may specify either
                  GSS_C_NO_OID to use a mechanism-specific
                  default printable syntax, or an OID recognized
                  by the GSS-API implementation to name a
                  specific namespace.

output_name       gss_name_t, modify
                  returned name in internal form.  Storage
                  associated with this name must be freed
                  by the application after use with a call
                  to gss_release_name().

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion

GSS_S_BAD_NAMETYPE The input_name_type was unrecognized

GSS_S_BAD_NAME    The input_name parameter could not be interpreted
                  as a name of the specified type

GSS_S_BAD_MECH    The input name-type was GSS_C_NT_EXPORT_NAME,
                  but the mechanism contained within the
                  input-name is not supported
```

7.17.gss_import_sec_context

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

Purpose:

Allows a process to import a security context established by another process. A given interprocess token may be imported only once. See gss_export_sec_context.

Parameters:

```
minor_status      Integer, modify
                  Mechanism specific status code

interprocess_token  buffer, opaque, modify
                    token received from exporting process

context_handle    gss_ctx_id_t, modify
                  context handle of newly reactivated context.
                  Resources associated with this context handle
                  must be released by the application after use
                  with a call to gss_delete_sec_context().

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion.

GSS_S_NO_CONTEXT  The token did not contain a valid context
reference.

GSS_S_DEFECTIVE_TOKEN The token was invalid.

GSS_S_UNAVAILABLE The operation is unavailable.

GSS_S_UNAUTHORIZED Local policy prevents the import of this context
                   by the current process.
```

7.18.gss_indicate_mechs

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

Purpose:

Allows an application to determine which underlying security mechanisms are available.

Parameters:

```
minor_status      Integer, modify
                  Mechanism specific status code.

mech_set          set of Object IDs, modify
                  set of implementation-supported mechanisms.
                  The returned gss_OID_set value will be a
                  dynamically-allocated OID set, that should
                  be released by the caller after use with a
                  call to gss_release_oid_set().

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion
```

7.19.gss_init_sec_context

```
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 )
```

Purpose:

Initiates the establishment of a security context between the application and a remote peer. Initially, the input_token parameter should be specified either as GSS_C_NO_BUFFER, or as a pointer to a gss_buffer_desc object whose length field contains the value zero. The routine may return a output_token which should be transferred to the peer application, where the peer application will present it to gss_accept_sec_context. If no token need be sent, gss_init_sec_context will indicate this by setting the length field of the output_token argument to zero. To complete the context establishment, one or more reply tokens may be required from the peer application; if so, gss_init_sec_context will return a status containing the supplementary information bit GSS_S_CONTINUE_NEEDED. In this case, gss_init_sec_context should be called again when the reply token is received from the peer application, passing the reply token to gss_init_sec_context via the input_token parameters.

Portable applications should be constructed to use the token length and return status to determine whether a token needs to be sent or waited for. Thus a typical portable caller should always invoke gss_init_sec_context within a loop:

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

while (!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);
  if (GSS_ERROR(maj_stat)) {
    report_error(maj_stat, min_stat);
  };

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

    if (context_hdl != GSS_C_NO_CONTEXT)
      gss_delete_sec_context(&min_stat,
                             &context_hdl,
                             GSS_C_NO_BUFFER);
    break;
  };
  if (maj_stat & GSS_S_CONTINUE_NEEDED) {
    receive_token_from_peer(input_token);
  } else {
    context_established = 1;
  };
```

```
};
```

Whenever the routine returns a major status that includes the value GSS_S_CONTINUE_NEEDED, the context is not fully established and the following restrictions apply to the output parameters:

The value returned via the time_rec parameter is undefined Unless the accompanying ret_flags parameter contains the bit GSS_C_PROT_READY_FLAG, indicating that per-message services may be applied in advance of a successful completion status, the value returned via the actual_mech_type parameter is undefined until the routine returns a major status value of GSS_S_COMPLETE.

The values of the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG and GSS_C_ANON_FLAG bits returned via the ret_flags parameter should contain the values that the implementation expects would be valid if context establishment were to succeed. In particular, if the application has requested a service such as delegation or anonymous authentication via the req_flags argument, and such a service is unavailable from the underlying mechanism, gss_init_sec_context should generate a token that will not provide the service, and indicate via the ret_flags argument that the service will not be supported. The application may choose to abort the context establishment by calling gss_delete_sec_context (if it cannot continue in the absence

of the service), or it may choose to transmit the token and continue context establishment (if the service was merely desired but not mandatory).

The values of the GSS_C_PROT_READY_FLAG and GSS_C_TRANS_FLAG bits within ret_flags should indicate the actual state at the time gss_init_sec_context returns, whether or not the context is fully established.

GSS-API implementations that support per-message protection are encouraged to set the GSS_C_PROT_READY_FLAG in the final ret_flags returned to a caller (i.e. when accompanied by a GSS_S_COMPLETE status code). However, applications should not rely on this behavior as the flag was not defined in Version 1 of the GSS-API. Instead, applications should determine what per-message services are available after a successful context establishment according to the GSS_C_INTEG_FLAG and GSS_C_CONF_FLAG values.

All other bits within the ret_flags argument should be set to zero.

If the initial call of gss_init_sec_context() fails, the implementation should not create a context object, and should leave the value of the context_handle parameter set to GSS_C_NO_CONTEXT to indicate this. In the event of a failure on a subsequent call, the implementation is permitted to delete the "half-built" security context (in which case it should set the context_handle parameter to GSS_C_NO_CONTEXT), but the preferred behavior is to leave the security context untouched for the application to delete (using gss_delete_sec_context).

During context establishment, the informational status bits GSS_S_OLD_TOKEN and GSS_S_DUPLICATE_TOKEN indicate fatal errors, and GSS-API mechanisms should always return them in association with a routine error of GSS_S_FAILURE. This requirement for pairing did not exist in version 1 of the GSS-API specification, so applications that wish to run over version 1 implementations must special-case these codes.

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code.

initiator_cred_handle gss_cred_id_t, read, optional
                       handle for credentials claimed. Supply
                       GSS_C_NO_CREDENTIAL to act as a default
                       initiator principal. If no default
                       initiator is defined, the function will
                       return GSS_S_NO_CRED.

context_handle   gss_ctx_id_t, read/modify
                  context handle for new context. Supply

GSS_C_NO_CONTEXT for first call; use value returned by first call in continuation calls. Resources associated with this context-handle must be released by the application after use with a call to gss_delete_sec_context().

target_name gss_name_t, read
Name of target

mech_type OID, read, optional
Object ID of desired mechanism. Supply
GSS_C_NO_OID to obtain an implementation
specific default

req_flags bit-mask, read
Contains various independent flags, each of
which requests that the context support a
specific service option. Symbolic
names are provided for each flag, and the
symbolic names corresponding to the required
flags should be logically-ORed
together to form the bit-mask value. The
flags are:

GSS_C_DELEG_FLAG
True - Delegate credentials to remote peer
False - Don't delegate

GSS_C_MUTUAL_FLAG
True - Request that remote peer
authenticate itself
False - Authenticate self to remote peer
only

GSS_C_REPLAY_FLAG
True - Enable replay detection for
messages protected with gss_wrap
or gss_get_mic
False - Don't attempt to detect
replayed messages

GSS_C_SEQUENCE_FLAG
True - Enable detection of out-of-sequence
protected messages
False - Don't attempt to detect
out-of-sequence messages

GSS_C_CONF_FLAG
True - Request that confidentiality service
be made available (via gss_wrap)
False - No per-message confidentiality service
is required.

GSS_C_INTEG_FLAG
True - Request that integrity service be

```
                           made available (via gss_wrap or
                           gss_get_mic)
                    False - No per-message integrity service
                            is required.

                  GSS_C_ANON_FLAG
                    True - Do not reveal the initiator's
                           identity to the acceptor.
                    False - Authenticate normally.

time_req          Integer, read, optional
                  Desired number of seconds for which context
                  should remain valid.  Supply 0 to request a
                  default validity period.

input_chan_bindings  channel bindings, read, optional
                     Application-specified bindings.  Allows
                     application to securely bind channel
                     identification information to the security
                     context.  Specify GSS_C_NO_CHANNEL_BINDINGS
                     if channel bindings are not used.

input_token       buffer, opaque, read, optional (see text)
                  Token received from peer application.
                  Supply GSS_C_NO_BUFFER, or a pointer to
                  a buffer containing the value GSS_C_EMPTY_BUFFER
                  on initial call.

actual_mech_type  OID, modify, optional
                  Actual mechanism used.  The OID returned via
                  this parameter will be a pointer to static
                  storage that should be treated as read-only;
                  In particular the application should not attempt
                  to free it.  Specify NULL if not required.

output_token      buffer, opaque, modify
                  token to be sent to peer application.  If
                  the length field of the returned buffer is
                  zero, no token need be sent to the peer
                  application.  Storage associated with this
                  buffer must be freed by the application
                  after use with a call to gss_release_buffer().

ret_flags         bit-mask, modify, optional
                  Contains various independent flags, each of which
                  indicates that the context supports a specific
```

service option. Specify NULL if not required. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically-ANDed with the ret_flags value to test whether a given option is supported by the context. The flags are:

GSS_C_DELEG_FLAG

True - Credentials were delegated to the remote peer
False - No credentials were delegated

GSS_C_MUTUAL_FLAG
True - The remote peer has authenticated itself.
False - Remote peer has not authenticated itself.

GSS_C_REPLAY_FLAG
True - replay of protected messages will be detected
False - replayed messages will not be detected

GSS_C_SEQUENCE_FLAG
True - out-of-sequence protected messages will be detected
False - out-of-sequence messages will not be detected

GSS_C_CONF_FLAG
True - Confidentiality service may be invoked by calling gss_wrap routine
False - No confidentiality service (via gss_wrap) available. gss_wrap will provide message encapsulation, data-origin authentication and integrity services only.

GSS_C_INTEG_FLAG
True - Integrity service may be invoked by calling either gss_get_mic or gss_wrap routines.

False - Per-message integrity service unavailable.

GSS_C_ANON_FLAG

True - The initiator's identity has not been revealed, and will not be revealed if any emitted token is passed to the acceptor.

False - The initiator's identity has been or will be authenticated normally.

GSS_C_PROT_READY_FLAG

True - Protection services (as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG) are available for use if the accompanying major status return value is either GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED.

False - Protection services (as specified by the states of the GSS_C_CONF_FLAG

and GSS_C_INTEG_FLAG) are available only if the accompanying major status return value is GSS_S_COMPLETE.

GSS_C_TRANS_FLAG

True - The resultant security context may be transferred to other processes via a call to gss_export_sec_context().

False - The security context is not transferrable.

All other bits should be set to zero.

time_rec Integer, modify, optional
number of seconds for which the context will remain valid. If the implementation does not support context expiration, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required.

Function value: GSS status code

GSS_S_COMPLETE Successful completion

GSS_S_CONTINUE_NEEDED Indicates that a token from the peer application is required to complete the context, and that gss_init_sec_context must be called again with that token.

GSS_S_DEFECTIVE_TOKEN Indicates that consistency checks performed on the input_token failed

GSS_S_DEFECTIVE_CREDENTIAL Indicates that consistency checks performed on the credential failed.

GSS_S_NO_CRED The supplied credentials were not valid for context initiation, or the credential handle did not reference any credentials.

GSS_S_CREDENTIALS_EXPIRED The referenced credentials have expired

GSS_S_BAD_BINDINGS The input_token contains different channel bindings to those specified via the input_chan_bindings parameter

GSS_S_BAD_SIG The input_token contains an invalid MIC, or a MIC that could not be verified

GSS_S_OLD_TOKEN The input_token was too old. This is a fatal error during context establishment

GSS_S_DUPLICATE_TOKEN The input_token is valid, but is a duplicate of a token already processed. This is a fatal error during context establishment.

GSS_S_NO_CONTEXT Indicates that the supplied context handle did not refer to a valid context

GSS_S_BAD_NAMETYPE The provided target_name parameter contained an invalid or unsupported type of name

GSS_S_BAD_NAME The provided target_name parameter was ill-formed.

GSS_S_BAD_MECH The specified mechanism is not supported by the provided credential, or is unrecognized by the implementation.

7.20.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 )
```

Purpose:

Obtains information about a security context. The caller must already have obtained a handle that refers to the context, although the context need not be fully established.

Parameters:

```
minor_status      Integer, modify
                  Mechanism specific status code

context_handle    gss_ctx_id_t, read
                  A handle that refers to the security context.

src_name          gss_name_t, modify, optional
                  The name of the context initiator.
                  If the context was established using anonymous
                  authentication, and if the application invoking
                  gss_inquire_context is the context acceptor,
                  an anonymous name will be returned.  Storage
                  associated with this name must be freed by the
                  application after use with a call to
                  gss_release_name().  Specify NULL if not
                  required.

targ_name         gss_name_t, modify, optional
                  The name of the context acceptor.
                  Storage associated with this name must be
                  freed by the application after use with a call
                  to gss_release_name().  If the context acceptor
                  did not authenticate itself, and if the initiator
                  did not specify a target name in its call to
                  gss_init_sec_contect(), the value GSS_C_NO_NAME
                  will be returned.  Specify NULL if not required.

lifetime_rec      Integer, modify, optional
                  The number of seconds for which the context
                  will remain valid.  If the context has
                  expired, this parameter will be set to zero.
                  If the implementation does not support
                  context expiration, the value
                  GSS_C_INDEFINITE will be returned.  Specify
```

NULL if not required.

mech_type gss_OID, modify, optional
The security mechanism providing the context. The returned OID will be a pointer to static storage that should be treated as read-only by the application; in particular the application should not attempt to free it. Specify NULL if not required.

ctx_flags bit-mask, modify, optional
Contains various independent flags, each of which indicates that the context supports (or is expected to support, if ctx_open is false) a specific service option. If not needed, specify NULL. Symbolic names are provided for each flag, and the symbolic names corresponding to the required flags should be logically-ANDed with the ret_flags value to test whether a given option is supported by the context. The flags are:

GSS_C_DELEG_FLAG
True - Credentials were delegated from the initiator to the acceptor.
False - No credentials were delegated

GSS_C_MUTUAL_FLAG
True - The acceptor was authenticated to the initiator
False - The acceptor did not authenticate itself.

GSS_C_REPLAY_FLAG
True - replay of protected messages will be detected
False - replayed messages will not be detected

GSS_C_SEQUENCE_FLAG
True - out-of-sequence protected messages will be detected
False - out-of-sequence messages will not be detected

GSS_C_CONF_FLAG

True - Confidentiality service may be invoked by calling gss_wrap routine

False - No confidentiality service (via gss_wrap) available. gss_wrap will provide message encapsulation, data-origin authentication and integrity services only.

GSS_C_INTEG_FLAG

True - Integrity service may be invoked by calling either gss_get_mic or gss_wrap routines.

False - Per-message integrity service unavailable.

GSS_C_ANON_FLAG

True - The initiator's identity will not be revealed to the acceptor. The src_name parameter (if requested) contains an anonymous internal name.

False - The initiator has been authenticated normally.

GSS_C_PROT_READY_FLAG

True - Protection services (as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG) are available for use.

False - Protection services (as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG) are available only if the context is fully established (i.e. if the open parameter is non-zero).

GSS_C_TRANS_FLAG

True - The resultant security context may be transferred to other processes via a call to gss_export_sec_context().

False - The security context is not transferrable.

locally_initiated Boolean, modify

```
                  Non-zero if the invoking application is the
                  context initiator.
                  Specify NULL if not required.

open              Boolean, modify
                  Non-zero if the context is fully established;
                  Zero if a context-establishment token
                  is expected from the peer application.
                  Specify NULL if not required.

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion

GSS_S_NO_CONTEXT  The referenced context could not be accessed.
```

7.21.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 )
```

Purpose:

Obtains information about a credential.

Parameters:

```
minor_status      Integer, modify
                  Mechanism specific status code

cred_handle       gss_cred_id_t, read
                  A handle that refers to the target credential.
                  Specify GSS_C_NO_CREDENTIAL to inquire about
                  the default initiator principal.

name              gss_name_t, modify, optional
                  The name whose identity the credential asserts.
```

Storage associated with this name should be freed by the application after use with a call to gss_release_name(). Specify NULL if not required.

lifetime Integer, modify, optional
The number of seconds for which the credential will remain valid. If the credential has expired, this parameter will be set to zero. If the implementation does not support credential expiration, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required.

cred_usage gss_cred_usage_t, modify, optional
How the credential may be used. One of the following:
GSS_C_INITIATE
GSS_C_ACCEPT
GSS_C_BOTH
Specify NULL if not required.

mechanisms gss_OID_set, modify, optional
Set of mechanisms supported by the credential. Storage associated with this OID set must be freed by the application after use with a call to gss_release_oid_set(). Specify NULL if not required.

Function value: GSS status code

GSS_S_COMPLETE Successful completion

GSS_S_NO_CRED The referenced credentials could not be accessed.

GSS_S_DEFECTIVE_CREDENTIAL The referenced credentials were invalid.

GSS_S_CREDENTIALS_EXPIRED The referenced credentials have expired.
If the lifetime parameter was not passed as NULL, it will be set to 0.

7.22.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 )
```

Purpose:

Obtains per-mechanism information about a credential.

Parameters:

minor_status   Integer, modify
Mechanism specific status code

cred_handle   gss_cred_id_t, read
A handle that refers to the target credential. Specify GSS_C_NO_CREDENTIAL to inquire about the default initiator principal.

mech_type   gss_OID, read
The mechanism for which information should be returned.

name   gss_name_t, modify, optional
The name whose identity the credential asserts. Storage associated with this name must be freed by the application after use with a call to gss_release_name(). Specify NULL if not required.

initiator_lifetime Integer, modify, optional
The number of seconds for which the credential will remain capable of initiating security contexts under the specified mechanism. If the credential can no longer be used to initiate contexts, or if the credential usage for this mechanism is GSS_C_ACCEPT, this parameter will be set to zero. If the implementation does not support expiration of initiator credentials, the value GSS_C_INDEFINITE will be returned. Specify NULL if not required.

acceptor_lifetime Integer, modify, optional
The number of seconds for which the credential will remain capable of accepting security contexts under the specified mechanism. If the credential can no longer be used to accept contexts, or if the credential usage for this mechanism is

```
                  GSS_C_INITIATE, this parameter will be set to zero.

                  If the implementation does not support expiration
                  of acceptor credentials, the value GSS_C_INDEFINITE
                  will be returned.  Specify NULL if not required.

cred_usage        gss_cred_usage_t, modify, optional
                  How the credential may be used with the specified
                  mechanism.  One of the following:
                    GSS_C_INITIATE
                    GSS_C_ACCEPT
                    GSS_C_BOTH
                  Specify NULL if not required.

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion

GSS_S_NO_CRED     The referenced credentials could not be accessed.

GSS_S_DEFECTIVE_CREDENTIAL The referenced credentials were invalid.

GSS_S_CREDENTIALS_EXPIRED The referenced credentials have expired.
                 If the lifetime parameter was not passed as NULL,
                 it will be set to 0.
```

7.23.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 )
```

Purpose:

Returns the set of mechanisms supported by the GSS-API implementation that may be able to process the specified name.

Each mechanism returned will recognize at least one element within the name. It is permissible for this routine to be implemented within a mechanism-independent GSS-API layer, using the type information contained within the presented name, and based on registration information provided by individual mechanism implementations. This means that the returned mech_types set may indicate that a particular mechanism will understand the name when in fact it would refuse to accept the name as input to gss_canonicalize_name, gss_init_sec_context, gss_acquire_cred or gss_add_cred (due to some property of the specific name, as opposed to the name type). Thus this routine should be used only as a pre-filter for a call to a subsequent mechanism-specific routine.

Parameters:

minor_status   Integer, modify
                  Implementation specific status code.

input_name   gss_name_t, read
                  The name to which the inquiry relates.

mech_types   gss_OID_set, modify
                  Set of mechanisms that may support the
                  specified name. The returned OID set
                  must be freed by the caller after use
                  with a call to gss_release_oid_set().

```
Function value:   GSS status code

GSS_S_COMPLETE    Successful completion

GSS_S_BAD_NAME    The input_name parameter was ill-formed.

GSS_S_BAD_NAMETYPE The input_name parameter contained an invalid or
                   unsupported type of name
```

7.24.gss_inquire_names_for_mech

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

Purpose:

Returns the set of nametypes supported by the specified mechanism.

Parameters:

```
minor_status      Integer, modify
                  Implementation specific status code.

mechanism         gss_OID, read
                  The mechanism to be interrogated.

name_types        gss_OID_set, modify
                  Set of name-types supported by the specified
                  mechanism.  The returned OID set must be
                  freed by the application after use with a
                  call to gss_release_oid_set().

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion
```

7.25.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)
```

Purpose:

Provides a way to pass an asynchronous token to the security service. Most context-level tokens are emitted and processed synchronously by gss_init_sec_context and gss_accept_sec_context, and the application is informed as to whether further tokens are expected by the GSS_C_CONTINUE_NEEDED major status bit. Occasionally, a mechanism may need to emit a context-level token at a point when the peer entity is not expecting a token. For example, the initiator's final call to gss_init_sec_context may emit a token and return a status of GSS_S_COMPLETE, but the acceptorÆs call to gss_accept_sec_context may fail. The acceptor's mechanism may wish to send a token containing an error indication to the initiator, but the initiator is not expecting a token at this point, believing that the context is fully established. Gss_process_context_token provides a way to pass such a token to the mechanism at any time.

Parameters:

minor_status   Integer, modify
                  Implementation specific status code.

context_handle   gss_ctx_id_t, read
                  context handle of context on which token is to
                  be processed

token_buffer   buffer, opaque, read
                  token to process

Function value: GSS status code

GSS_S_COMPLETE   Successful completion

GSS_S_DEFECTIVE_TOKEN Indicates that consistency checks performed
                  on the token failed

GSS_S_NO_CONTEXT The context_handle did not refer to a valid context

7.26.gss_release_buffer

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

Purpose:

Free storage associated with a buffer. The storage must have been allocated by a GSS-API routine. In addition to freeing the associated storage, the routine will zero the length field in the descriptor to which the buffer parameter refers, and implementations are encouraged to additionally set the pointer field in the descriptor to NULL. Any buffer object returned by a GSS-API routine may be passed to gss_release_buffer (even if there is no storage associated with the buffer).

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code

buffer   buffer, modify
                  The storage associated with the buffer will be deleted. The gss_buffer_desc object will not be freed, but its length field will be zeroed.

Function value: GSS status code

GSS_S_COMPLETE   Successful completion

7.27.gss_release_cred

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

Purpose:

Informs GSS-API that the specified credential handle is no longer required by the application, and frees associated resources. Implementations are encouraged to set the cred_handle to GSS_C_NO_CREDENTIAL on successful completion of this call.

Parameters:

cred_handle — gss_cred_id_t, modify, optional
Opaque handle identifying credential to be released. If GSS_C_NO_CREDENTIAL is supplied, the routine will complete successfully, but will do nothing.

minor_status — Integer, modify
Mechanism specific status code.

Function value: GSS status code

GSS_S_COMPLETE — Successful completion

GSS_S_NO_CRED — Credentials could not be accessed.

7.28.gss_release_name

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

Purpose:

Free GSSAPI-allocated storage associated with an internal-form name. Implementations are encouraged to set the name to GSS_C_NO_NAME on successful completion of this call.

Parameters:

```
minor_status      Integer, modify
                  Mechanism specific status code

name              gss_name_t, modify
                  The name to be deleted

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion

GSS_S_BAD_NAME    The name parameter did not contain a valid name
```

7.29.gss_release_oid_set

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

Purpose:

Free storage associated with a GSSAPI-generated gss_OID_set object. The set parameter must refer to an OID-set that was returned from a GSS-API routine. gss_release_oid_set() will free the storage associated with each individual member OID, the OID set's elements array, and the gss_OID_set_desc.

Implementations are encouraged to set the gss_OID_set parameter to GSS_C_NO_OID_SET on successful completion of this routine.

Parameters:

minor_status	Integer, modify Mechanism specific status code
set	Set of Object IDs, modify The storage associated with the gss_OID_set will be deleted.

Function value:	GSS status code
GSS_S_COMPLETE	Successful completion

7.30.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)
```

Purpose:

Interrogate an Object Identifier set to determine whether a specified Object Identifier is a member. This routine is intended to be used with OID sets returned by gss_indicate_mechs(), gss_acquire_cred(), and gss_inquire_cred(), but will also work with user-generated sets.

Parameters:

```
minor_status      Integer, modify
                  Mechanism specific status code

member            Object ID, read
                  The object identifier whose presence
                  is to be tested.

set               Set of Object ID, read
                  The Object Identifier set.
```

```
present           Boolean, modify
                  non-zero if the specified OID is a member
                  of the set, zero if not.

Function value:   GSS status code

GSS_S_COMPLETE    Successful completion
```

7.31.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)
```

Purpose:

Converts a message previously protected by gss_wrap back to a usable form, verifying the embedded MIC. The conf_state parameter indicates whether the message was encrypted; the qop_state parameter indicates the strength of protection that was used to provide the confidentiality and integrity services.

Since some application-level protocols may wish to use tokens emitted by gss_wrap() to provide "secure framing", implementations must support the wrapping and unwrapping of zero-length messages.

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code.

context_handle   gss_ctx_id_t, read
                  Identifies the context on which the message arrived

input_message_buffer buffer, opaque, read
                  protected message

output_message_buffer buffer, opaque, modify
                  Buffer to receive unwrapped message. Storage associated with this buffer must be freed by the application after use use with a call to gss_release_buffer().

conf_state   boolean, modify, optional
                  Non-zero - Confidentiality and integrity
                             protection were used
                  Zero - Integrity service only was used
                  Specify NULL if not required

qop_state   gss_qop_t, modify, optional
                  Quality of protection provided.
                  Specify NULL if not required

Function value: GSS status code

GSS_S_COMPLETE   Successful completion

GSS_S_DEFECTIVE_TOKEN The token failed consistency checks

GSS_S_BAD_SIG   The MIC was incorrect

GSS_S_DUPLICATE_TOKEN The token was valid, and contained a correct MIC for the message, but it had already been processed

GSS_S_OLD_TOKEN The token was valid, and contained a correct MIC for the message, but it is too old to check for duplication.

GSS_S_UNSEQ_TOKEN The token was valid, and contained a correct MIC for the message, but has been verified out of sequence; a later token has already been received.

GSS_S_GAP_TOKEN The token was valid, and contained a correct MIC for the message, but has been verified out of sequence; an earlier expected token has not yet been received.

GSS_S_CONTEXT_EXPIRED The context has already expired

GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context

7.32.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)
```

Purpose:

Verifies that a cryptographic MIC, contained in the token parameter, fits the supplied message. The qop_state parameter allows a message recipient to determine the strength of protection that was applied to the message.

Since some application-level protocols may wish to use tokens emitted by gss_wrap() to provide "secure framing", implementations must support the calculation and verification of MICs over zero-length messages.

Parameters:

```
minor_status      Integer, modify
                  Mechanism specific status code.

context_handle    gss_ctx_id_t, read
                  Identifies the context on which the message
                  arrived

message_buffer    buffer, opaque, read
                  Message to be verified

token_buffer      buffer, opaque, read
                  Token associated with message

qop_state         gss_qop_t, modify, optional
                  quality of protection gained from MIC
                  Specify NULL if not required
```

```
Function value:   GSS status code

GSS_S_COMPLETE    Successful completion

GSS_S_DEFECTIVE_TOKEN The token failed consistency checks

GSS_S_BAD_SIG     The MIC was incorrect

GSS_S_DUPLICATE_TOKEN The token was valid, and contained a correct
                  MIC for the message, but it had already been
```

processed

GSS_S_OLD_TOKEN The token was valid, and contained a correct MIC for the message, but it is too old to check for duplication.

GSS_S_UNSEQ_TOKEN The token was valid, and contained a correct MIC for the message, but has been verified out of sequence; a later token has already been received.

GSS_S_GAP_TOKEN The token was valid, and contained a correct MIC for the message, but has been verified out of sequence; an earlier expected token has not yet been received.

GSS_S_CONTEXT_EXPIRED The context has already expired

GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context

7.33.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 )
```

Purpose:

Attaches a cryptographic MIC and optionally encrypts the specified input_message. The output_message contains both the MIC and the message. The qop_req parameter allows a choice between several cryptographic algorithms, if supported by the chosen mechanism.

Since some application-level protocols may wish to use tokens emitted by gss_wrap() to provide "secure framing", implementations must support the wrapping of zero-length messages.

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code.

context_handle   gss_ctx_id_t, read
                  Identifies the context on which the message
                  will be sent

conf_req_flag   boolean, read
Non-zero - Both confidentiality and integrity
services are requested
Zero - Only integrity service is requested

qop_req   gss_qop_t, read, optional
Specifies required quality of protection. A
mechanism-specific default may be requested by
setting qop_req to GSS_C_QOP_DEFAULT. If an
unsupported protection strength is requested,
gss_wrap will return a major_status of
GSS_S_BAD_QOP.

input_message_buffer buffer, opaque, read
Message to be protected

conf_state   boolean, modify, optional
Non-zero - Confidentiality, data origin
authentication and integrity
services have been applied
Zero - Integrity and data origin services only
has been applied.
Specify NULL if not required

output_message_buffer buffer, opaque, modify
Buffer to receive protected message.
Storage associated with this message must
be freed by the application after use with
a call to gss_release_buffer().

Function value: GSS status code

GSS_S_COMPLETE   Successful completion

GSS_S_CONTEXT_EXPIRED The context has already expired

GSS_S_NO_CONTEXT The context_handle parameter did not identify a
valid context

GSS_S_BAD_QOP   The specified QOP is not supported by the
mechanism.

7.34.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)
```

Purpose:

Allows an application to determine the maximum message size that, if presented to gss_wrap with the same conf_req_flag and qop_req parameters, will result in an output token containing no more than req_output_size bytes.

This call is intended for use by applications that communicate over protocols that impose a maximum message size. It enables the application to fragment messages prior to applying protection.

GSS-API implementations are recommended but not required to detect invalid QOP values when gss_wrap_size_limit() is called. This routine guarantees only a maximum message size, not the availability of specific QOP values for message protection.

Successful completion of this call does not guarantee that gss_wrap will be able to protect a message of length max_input_size bytes, since this ability may depend on the availability of system resources at the time that gss_wrap is called. However, if the implementation itself imposes an upper limit on the length of messages that may be processed by gss_wrap, the implementation should not return a value via max_input_bytes that is greater than this length.

Parameters:

minor_status   Integer, modify
                  Mechanism specific status code

context_handle   gss_ctx_id_t, read
                  A handle that refers to the security over which the messages will be sent.

conf_req_flag   Boolean, read
                  Indicates whether gss_wrap will be asked to apply confidentiality protection in addition to integrity protection. See the routine description for gss_wrap for more details.

qop_req   gss_qop_t, read
                  Indicates the level of protection that gss_wrap will be asked to provide. See the routine description for gss_wrap for more details.

req_output_size Integer, read
The desired maximum size for tokens emitted by gss_wrap.

max_input_size Integer, modify
The maximum input message size that may be presented to gss_wrap in order to guarantee that the emitted token shall be no larger than req_output_size bytes.

Function value: GSS status code

GSS_S_COMPLETE Successful completion

GSS_S_NO_CONTEXT The referenced context could not be accessed.

GSS_S_CONTEXT_EXPIRED The context has expired.

GSS_S_BAD_QOP The specified QOP is not supported by the mechanism.

8. SECURITY CONSIDERATIONS

This document specifies a service interface for security facilities and services; as such, security considerations appear throughout the specification. Nonetheless, it is appropriate to summarize certain specific points relevant to GSS-API implementors and calling applications. Usage of the GSS-API interface does not in itself provide security services or assurance; instead, these attributes are dependent on the underlying mechanism(s) which support a GSS-API implementation. Callers must be attentive to the requests made to GSS-API calls and to the status indicators returned by GSS-API, as these specify the security service characteristics which GSS-API will provide. When the interprocess context transfer facility is used, appropriate local controls should be applied to constrain access to interprocess tokens and to the sensitive data which they contain.

APPENDIX A. GSS-API C header file gssapi.h

C-language GSS-API implementations should include a copy of the following header-file.

```
#ifndef GSSAPI_H_
#define GSSAPI_H_



/*
 * First, include stddef.h to get size_t defined.
 */
#include <stddef.h>

/*
 * If the platform supports the xom.h header file, it should be
 * included here.
 */
#include <xom.h>


/*
 * Now define the three implementation-dependent types.
 */
```

```
typedef <platform-specific> gss_ctx_id_t;
typedef <platform-specific> gss_cred_id_t;
typedef <platform-specific> gss_name_t;

/*
 * The following type must be defined as the smallest natural
 * unsigned integer supported by the platform that has at least
 * 32 bits of precision.
 */
typedef <platform-specific> gss_uint32;
```

```
#ifdef OM_STRING
/*
 * We have included the xom.h header file.  Verify that OM_uint32
 * is defined correctly.
 */

#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

/*
 * We can't use X/Open definitions, so roll our own.
 */

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;

/*
 * For now, define a QOP-type as an OM_uint32
 */
typedef OM_uint32 gss_qop_t;

typedef int gss_cred_usage_t;

/*
 * Flag bits for context-level services.
 */
```

```
#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


/*
 * Credential usage options
 */
#define GSS_C_BOTH     0
#define GSS_C_INITIATE 1
#define GSS_C_ACCEPT   2

/*
 * Status code types for gss_display_status
 */
```

```
#define GSS_C_GSS_CODE  1
#define GSS_C_MECH_CODE 2

/*
 * The constant definitions for channel-bindings address families
 */
#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

/*
 * Various Null values
 */
#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}

/*
 * Some alternate names for a couple of the above
 * values.  These are defined for V1 compatibility.
 */
#define GSS_C_NULL_OID GSS_C_NO_OID
#define GSS_C_NULL_OID_SET GSS_C_NO_OID_SET

/*
 * Define the default Quality of Protection for per-message
 * services.  Note that an implementation that offers multiple
 * levels of QOP may define GSS_C_QOP_DEFAULT to be either zero
 * (as done here) to mean "default protection", or to a specific
 * explicit QOP value.  However, a value of 0 should always be
 * interpreted by a GSS-API implementation as a request for the
 * default protection level.
 */
#define GSS_C_QOP_DEFAULT 0

/*
 * Expiration time of 2^32-1 seconds means infinite lifetime for a
 * credential or security context
 */
#define GSS_C_INDEFINITE 0xfffffffful

/*
 * The implementation must reserve static storage for a
 * gss_OID_desc object containing the value
 * {10, (void *)"\x2a\x86\x48\x86\xf7\x12"
 * "\x01\x02\x01\x01"},
 * corresponding to an object-identifier value of
 * {iso(1) member-body(2) United States(840) mit(113554)
 * infosys(1) gssapi(2) generic(1) user_name(1)}.  The constant
 * GSS_C_NT_USER_NAME should be initialized to point
 * to that gss_OID_desc.
 */
extern gss_OID GSS_C_NT_USER_NAME;

/*
 * The implementation must reserve static storage for a
 * gss_OID_desc object containing the value
 * {10, (void *)"\x2a\x86\x48\x86\xf7\x12"
 *              "\x01\x02\x01\x02"},
 * corresponding to an object-identifier value of
 * {iso(1) member-body(2) United States(840) mit(113554)
 * infosys(1) gssapi(2) generic(1) machine_uid_name(2)}.
 * The constant GSS_C_NT_MACHINE_UID_NAME should be
 * initialized to point to that gss_OID_desc.
 */
extern gss_OID GSS_C_NT_MACHINE_UID_NAME;

/*
 * The implementation must reserve static storage for a
 * gss_OID_desc object containing the value
 * {10, (void *)"\x2a\x86\x48\x86\xf7\x12"
 *               "\x01\x02\x01\x03"},
 * corresponding to an object-identifier value of
 * {iso(1) member-body(2) United States(840) mit(113554)
 * infosys(1) gssapi(2) generic(1) string_uid_name(3)}.
 * The constant GSS_C_NT_STRING_UID_NAME should be
 * initialized to point to that gss_OID_desc.
 */
extern gss_OID GSS_C_NT_STRING_UID_NAME;

/*
 * The implementation must reserve static storage for a
 * gss_OID_desc object containing the value
 * {6, (void *)"\x2b\x06\x01\x05\x06\x02"},
 * corresponding to an object-identifier value of
 * {iso(1) org(3) dod(6) internet(1) security(5)
 * nametypes(6) gss-host-based-services(2)).  The constant
 * GSS_C_NT_HOSTBASED_SERVICE_X should be initialized to point
 * to that gss_OID_desc.  This is a deprecated OID value, and
 * implementations wishing to support hostbased-service names
 * should instead use the GSS_C_NT_HOSTBASED_SERVICE OID,
 * defined below, to identify such names;
 * GSS_C_NT_HOSTBASED_SERVICE_X should be accepted a synonym
 * for GSS_C_NT_HOSTBASED_SERVICE when presented as an input
 * parameter, but should not be emitted by GSS-API
 * implementations
 */
extern gss_OID GSS_C_NT_HOSTBASED_SERVICE_X;

/*
 * The implementation must reserve static storage for a
 * gss_OID_desc object containing the value
 * {10, (void *)"\x2a\x86\x48\x86\xf7\x12"
 *               "\x01\x02\x01\x04"}, corresponding to an
 * object-identifier value of {iso(1) member-body(2)
 * Unites States(840) mit(113554) infosys(1) gssapi(2)
 * generic(1) service_name(4)}.  The constant
 * GSS_C_NT_HOSTBASED_SERVICE should be initialized
 * to point to that gss_OID_desc.
 */
extern gss_OID GSS_C_NT_HOSTBASED_SERVICE;

/*
```

```
 * The implementation must reserve static storage for a
 * gss_OID_desc object containing the value
 * {6, (void *)"\x2b\x06\01\x05\x06\x03"},
 * corresponding to an object identifier value of
 * {1(iso), 3(org), 6(dod), 1(internet), 5(security),
 * 6(nametypes), 3(gss-anonymous-name)}.  The constant
```

```
 * and GSS_C_NT_ANONYMOUS should be initialized to point
 * to that gss_OID_desc.
 */
extern gss_OID GSS_C_NT_ANONYMOUS;


/*
 * The implementation must reserve static storage for a
 * gss_OID_desc object containing the value
 * {6, (void *)"\x2b\x06\x01\x05\x06\x04"},
 * corresponding to an object-identifier value of
 * {1(iso), 3(org), 6(dod), 1(internet), 5(security),
 * 6(nametypes), 4(gss-api-exported-name)}.  The constant
 * GSS_C_NT_EXPORT_NAME should be initialized to point
 * to that gss_OID_desc.
 */
extern gss_OID GSS_C_NT_EXPORT_NAME;


/* Major status codes */

#define GSS_S_COMPLETE 0

/*
 * Some "helper" definitions to make the status code macros obvious.
 */
#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

/*
 * The macros that test status codes for error conditions.
 * Note that the GSS_ERROR() macro has changed slightly from
 * the V1 GSS-API so that it now evaluates its argument
 * only once.
 */
#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)))

/*
 * Now the actual status code definitions
 */

/*
 * Calling errors:
```

```
 */
#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)

/*
 * Routine errors:
 */
#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)

/*
 * Supplementary info bits:
 */
#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))

/*
 * Finally, function prototypes for the GSS-API routines.
 */

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 */
              );

/*
 * The following routines are obsolete variants of gss_get_mic,
 * gss_verify_mic, gss_wrap and gss_unwrap.  They should be
 * provided by GSS-API V2 implementations for backwards
 * compatibility with V1 applications.  Distinct entrypoints
 * (as opposed to #defines) should be provided, both to allow
 * GSS-API V1 applications to link against GSS-API V2
implementations,
 * and to retain the slight parameter type differences between the
 * obsolete versions of these routines and their current forms.
 */

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_ */
```

APPENDIX B. Additional constraints for application binary portability

The purpose of this C-bindings document is to encourage source-level portability of applications across GSS-API implementations on different platforms and atop different mechanisms. Additional goals that have not been explicitly addressed by this document are link-time and run-time portability.

Link-time portability provides the ability to compile an application against one implementation of GSS-API, and then link it against a different implementation on the same platform. It is a stricter requirement than source-level portability.

Run-time portability differs from link-time portability only on those platforms that implement dynamically loadable GSS-API implementations, but do not offer load-time symbol resolution. On such platforms, run-time portability is a stricter requirement than link-time portability, and will typically include the precise placement of the various GSS-API routines within library entrypoint vectors.

Individual platforms will impose their own rules that must be followed to achieve link-time (and run-time, if different) portability. In order to ensure either form of binary portability, an ABI specification must be written for GSS-API implementations on that platform. However, it is recognized that there are some issues that are likely to be common to all such ABI specifications. This appendix is intended to be a repository for such common issues, and contains some suggestions that individual ABI specifications may choose to reference. Since machine architectures vary greatly, it may not be possible or desirable to follow these suggestions on all platforms.

B.1. Pointers

While ANSI-C provides a single pointer type for each declared type, plus a single (void *) type, some platforms (notably those using segmented memory architectures) augment this with various modified pointer types (e.g. far pointers, near pointers). These language bindings assume ANSI-C, and thus do not address such non-standard implementations. GSS-API implementations for such platforms must choose an appropriate memory model, and should use it consistently throughout. For example, if a memory model is chosen that requires the use of far pointers when passing routine parameters, then far

pointers should also be used within the structures defined by GSS-API.

B.2. Internal structure alignment

GSS-API defines several data-structures containing differently-sized fields. An ABI specification should include a detailed description of how the fields of such structures are aligned, and if there is any internal padding in these data structures. The use of compiler defaults for the platform is recommended.

B.3. Handle types

The C bindings specify that the gss_cred_id_t and gss_ctx_id_t types should be implemented as either pointer or arithmetic types, and that if pointer types are used, care should be taken to ensure that two handles may be compared with the == operator. Note that ANSI-C does not guarantee that two pointer values may be compared with the == operator unless either the two pointers point to members of a single array, or at least one of the pointers contains a NULL value.

For binary portability, additional constraints are required. The following is an attempt at defining platform-independent constraints.

The size of the handle type must be the same as sizeof(void *), using the appropriate memory model.

The == operator for the chosen type must be a simple bit-wise comparison. That is, for two in-memory handle objects h1 and h2, the boolean value of the expression

```
(h1 == h2)
```

should always be the same as the boolean value of the expression

```
(memcmp(&h1, &h2, sizeof(h1)) == 0)
```

The actual use of the type (void *) for handle types is discouraged, not for binary portability reasons, but since it effectively disables much of the compile-time type-checking that the compiler can otherwise perform, and is therefore not "programmer-friendly". If a pointer implementation is desired, and if the platform's implementation of pointers permits, the handles should be implemented as pointers to distinct implementation-defined types.

B.4. The gss_name_t type

The gss_name_t type, representing the internal name object, should be implemented as a pointer type. The use of the (void *) type is discouraged as it does not allow the compiler to perform strong type-checking. However, the pointer type chosen should be of the same size as the (void *) type. Provided this rule is obeyed, ABI specifications need not further constrain the implementation of gss_name_t objects.

B.5. The int and size_t types

Some platforms may support differently sized implementations of the "int" and "size_t" types, perhaps chosen through compiler switches, and perhaps dependent on memory model. An ABI specification for such a platform should include required implementations for these types. It is recommended that the default implementation (for the chosen memory model, if appropriate) is chosen.

B.6. Procedure-calling conventions

Some platforms support a variety of different binary conventions for calling procedures. Such conventions cover things like the format of the stack frame, the order in which the routine parameters are pushed onto the stack, whether or not a parameter count is pushed onto the stack, whether some argument(s) or return values are to be passed in registers, and whether the called routine or the caller is responsible for removing the stack frame on return. For such platforms, an ABI specification should specify which calling convention is to be used for GSS-API implementations.

REFERENCES

[GSSAPI]   J. Linn, "Generic Security Service
Application Program Interface, Version 2",
RFC2078.

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

AUTHOR'S ADDRESS

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