Network Working Group
Request for Comments: 1509

J. Wray
Digital Equipment Corporation

September 1993

Generic Security Service API : C-bindings

Status of this Memo

This RFC specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document specifies C language bindings for the Generic Security Service Application Program Interface (GSS-API), which is described at a language-independent conceptual level in other documents.

The Generic Security Service Application Programming Interface (GSS-API) provides security services to its callers, and is intended for implementation atop alternative 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.

1. INTRODUCTION

The Generic Security Service Application Programming Interface [1] 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 GSSAPI:

(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 the local username under which it is running.

(b) A pair of communicating applications establish a joint security context using their credentials. The security context is a pair of GSSAPI data structures that contain shared state information, which is required in order that per-message security services may be provided. 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. This transfer of rights is termed delegation, and is achieved by creating a set of credentials, similar to those used by the originating application, but which may be used by the responder. To establish and maintain the shared information that makes up the security context, certain GSSAPI calls will return a token data structure, which is a cryptographically protected opaque data type. The caller of such a GSSAPI routine is responsible for transferring the token to the peer application, which should then pass it to a corresponding GSSAPI routine which will decode it and extract the information.

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

(i) integrity and data origin authentication, or

(ii) confidentiality, integrity and data origin authentication to application data, which are treated by GSSAPI as arbitrary octet-strings. The application transmitting a message that it wishes to protect will call the appropriate GSSAPI routine (sign or seal) to apply protection, specifying the appropriate security context, and send the result to the receiving application. The receiver will pass the received data to the corresponding decoding routine (verify or unseal) to remove the protection and validate the data.

(d) At the completion of a communications session (which may extend across several connections), the peer applications call GSSAPI routines to delete the security context. Multiple contexts may also be used (either successively or simultaneously) within a single communications association.

2. GSSAPI Routines

This section lists the functions performed by each of the GSSAPI routines and discusses their major parameters, describing how they are to be passed to the routines. The routines are listed in figure 4-1.

Figure 4-1 GSSAPI Routines

Routine	Function
gss_acquire_cred	Assume a global identity
gss_release_cred	Discard credentials
gss_init_sec_context	Initiate a security context with a peer application
gss_accept_sec_context	Accept a security context initiated by a peer application
gss_process_context_token	Process a token on a security context from a peer application
gss_delete_sec_context	Discard a security context
gss_context_time	Determine for how long a context will remain valid
gss_sign	Sign a message; integrity service
gss_verify	Check signature on a message
gss_seal	Sign (optionally encrypt) a message; confidentiality service
gss_unseal	Verify (optionally decrypt) message
gss_display_status	Convert an API status code to text
gss_indicate_mechs	Determine underlying authentication mechanism
gss_compare_name	Compare two internal-form names
gss_display_name	Convert opaque name to text

gss_import_name	Convert a textual name to internal-form
gss_release_name	Discard an internal-form name
gss_release_buffer	Discard a buffer
gss_release_oid_set	Discard a set of object identifiers
gss_inquire_cred	Determine information about a credential

Individual GSSAPI 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.

2.1. Data Types and Calling Conventions

The following conventions are used by the GSSAPI:

2.1.1. Structured data types

Wherever these GSSAPI C-bindings describe structured data, only fields that must be provided by all GSSAPI implementation are documented. Individual implementations may provide additional fields, either for internal use within GSSAPI routines, or for use by non-portable applications.

2.1.2. Integer types

GSSAPI defines 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 GSSAPI routine definitions. Individual GSSAPI implementations will include appropriate typedef definitions to map this type onto a built-in data type.

2.1.3. String and similar data

Many of the GSSAPI routines take arguments and return values that describe contiguous multiple-byte data. All such data is passed between the GSSAPI 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 passed to the application by a GSSAPI routine using the gss_buffer_t conventions is allocated by the GSSAPI 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.

2.1.3.1. Opaque data types

Certain multiple-word data items are considered opaque data types at the GSSAPI, because their internal structure has no significance either to the GSSAPI 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_seal (which is opaque to the GSSAPI). Opaque data is passed between the GSSAPI and the application using the gss_buffer_t datatype.

2.1.3.2. Character strings

Certain multiple-word data items may be regarded as simple ISO Latin-1 character strings. An example of this is the input_name_buffer parameter to gss_import_name. Some GSSAPI routines also return character strings. Character strings are passed between the application and the GSSAPI using the gss_buffer_t datatype, defined earlier.

2.1.4. Object Identifiers

Certain GSSAPI 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 GSSAPI caller to select an underlying security mechanism. 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 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- oganization(3) icd-ecma(12) member-company(2) dec(1011) cryptoAlgorithms(7) SPX(5)} meaning SPX (Digital's 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. GSSAPI implementations should provide constant gss_OID values to allow callers 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 2.1.10). Applications should treat gss_OID_desc values returned by GSSAPI routines as read-only. In particular, the application should not attempt to deallocate them. The gss_OID_desc datatype is equivalent to the X/Open OM_object_identifier datatype [2].

2.1.5. Object Identifier Sets

Certain GSSAPI procedures take parameters of the type gss_OID_set. This type represents one or more object identifiers (section 2.1.4). A gss_OID_set object has the following structure:

```
typedef struct gss_OID_set_desc_struct {
   int       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 GSSAPI, to request the use of specific mechanisms, and to indicate which mechanisms a given credential supports. Storage associated with gss_OID_set values returned to the application by the GSSAPI may be deallocated by the gss_release_oid_set routine.

2.1.6. Credentials

A credential handle is a caller-opaque atomic datum that identifies a GSSAPI credential data structure. It is represented by the caller-opaque type gss_cred_id_t, which may be implemented as either an arithmetic or a pointer type. Credentials describe a principal, and they give their holder the ability to act as that principal. The GSSAPI does not make the actual credentials available to applications; instead the credential handle is used to identify a particular credential, held internally by GSSAPI or underlying

mechanism. Thus the credential handle contains no security-relavent information, and requires no special protection by the application. Depending on the implementation, a given credential handle may refer to different credentials when presented to the GSSAPI by different callers. Individual GSSAPI implementations should define both the scope of a credential handle and the scope of a credential itself (which must be at least as wide as that of a handle). Possibilities for credential handle scope include the process that acquired the handle, the acquiring process and its children, or all processes sharing some local identification information (e.g., UID). If no handles exist by which a given credential may be reached, the GSSAPI may delete the credential.

Certain routines allow credential handle parameters to be omitted to indicate the use of a default credential. The mechanism by which a default credential is established and its scope should be defined by the individual GSSAPI implementation.

2.1.7. Contexts

The gss_ctx_id_t data type contains a caller-opaque atomic value that identifies one end of a GSSAPI security context. It may be implemented as either an arithmetic or a pointer type. Depending on the implementation, a given gss_ctx_id_t value may refer to different GSSAPI security contexts when presented to the GSSAPI by different callers. The security context holds state information about each end of a peer communication, including cryptographic state information. Individual GSSAPI implementations should define the scope of a context. Since no way is provided by which a new gss_ctx_id_t value may be obtained for an existing context, the scope of a context should be the same as the scope of a gss_ctx_id_t.

2.1.8. Authentication tokens

A token is a caller-opaque type that GSSAPI uses to maintain synchronization between the context data structures at each end of a GSSAPI security context. The token is a cryptographically protected bit-string, generated by the underlying mechanism at one end of a GSSAPI 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 GSSAPI and the application using the gss_buffer_t conventions.

2.1.9. Status values

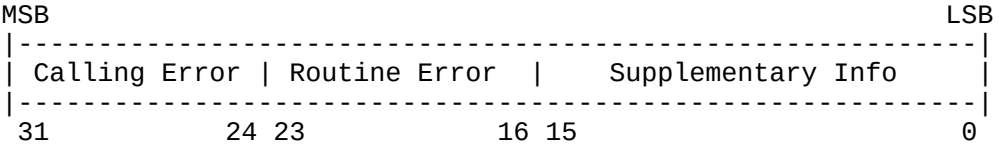
One or more status codes are returned by each GSSAPI routine. Two distinct sorts of status codes are returned. These are termed GSS status codes and Mechanism status codes.

2.1.9.1. GSS status codes

GSSAPI routines return GSS status codes as their OM_uint32 function value. These codes indicate errors that are independent of the underlying mechanism 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 GSSAPI specification) or calling errors (errors that are specific to these 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:



Hence if a GSSAPI 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_SIG	6	A token had an invalid signature
GSS_S_NO_CRED	7	No credentials were supplied
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)

Table 5-3 Supplementary Status Bits

Name	Bit Number	Meaning
GSS_S_CONTINUE_NEEDED	0 (LSB)	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

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.

A GSSAPI 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 routine status values only.

2.1.9.2. Mechanism-specific status codes

GSSAPI 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 GSSAPI routine, even if it returns a calling error or one of the generic API errors indicated above as fatal, although 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 set set by the routine, even in the event of an error, although in such cases the GSSAPI 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.

The GSS 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 status code will provide more details about the error.

2.1.10. Names

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

Two distinct representations are defined for names:

(a) A printable form, for presentation to a user

(b) An internal form, for presentation at the API

The syntax of a printable name is defined by the GSSAPI implementation, and may be dependent on local system configuration, or on individual user preference. The internal form provides a canonical representation of the name that is independent of configuration.

A given GSSAPI implementation may support names drawn from multiple namespaces. In such an implementation, the internal form of the name must include fields that identify the namespace from which the name is drawn. The namespace from which a printable name is drawn is specified by an accompanying object identifier.

Routines (gss_import_name and gss_display_name) are provided to convert names between their printable representations and the 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 preferred syntax for each supported name-type.

Comparison of internal-form names is accomplished via the gss_compare_names routine. This removes the need for the application program to understand the syntaxes of the various printable names that a given GSSAPI implementation may support.

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

2.1.11. Channel Bindings

GSSAPI supports the use of user-specified tags to identify a given context to the peer application. These tags are used to identify the particular communications channel that carries the context. Channel bindings are communicated to the GSSAPI 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	DARPA Internet address type
GSS_C_AF_IMPLINK	ARPAnet IMP address type (eg IP)
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 (eg X.25)
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	X25
GSS_C_AF_NULLADDR	No address specified

Note that these 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.

Conceptually, the GSSAPI concatenates the initiator_addrtype, initiator_address, acceptor_addrtype, acceptor_address and application_data to form an octet string. The mechanism signs this octet string, and binds the signature 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

signature is calculated in the same way. The calculated signature is compared with that found in the token, and if the signatures 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 signature); 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.

2.1.12. 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.

2.1.12.1. 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.

2.1.12.2. Integer types (input)

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

2.1.12.3. Integer types (output)

Specify NULL as the value for the pointer.

2.1.12.4. Pointer types

Specify NULL as the value.

2.1.12.5. Object IDs

Specify GSS_C_NULL_OID as the value.

2.1.12.6. Object ID Sets

Specify GSS_C_NULL_OID_SET as the value.

2.1.12.7. Credentials

Specify GSS_C_NO_CREDENTIAL to use the default credential handle.

2.1.12.8. Channel Bindings

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

3. GSSAPI routine descriptions

2.1. gss_acquire_cred

```
OM_uint32  gss_acquire_cred (
                OM_uint32 *     minor_status,
                gss_name_t      desired_name,
                OM_uint32       time_req,
                gss_OID_set     desired_mechs,
                int             cred_usage,
                gss_cred_id_t * output_cred_handle,
                gss_OID_set *   actual_mechs,
                 OM_int32 *      time_rec)
```

Purpose:

Allows an application to acquire a handle for a pre-existing credential by name. GSSAPI 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 result in 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 credential acquisition is time-consuming for a mechanism, the mechanism may chooses 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
number of seconds that credentials
should remain valid

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

cred_usage integer, 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.

actual_mechs Set of Object IDs, modify, optional
The set of mechanisms for which the
credential is valid. 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_FAILURE Unspecified failure. The minor_status parameter contains more detailed information

3.2. gss_release_cred

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

Purpose:

Informs GSSAPI that the specified credential handle is no longer required by the process. When all processes have released a credential, it will be deleted.

Parameters:

cred_handle	gss_cred_id_t, modify, optional buffer containing opaque credential handle. If GSS_C_NO_CREDENTIAL is supplied, the default credential will be released
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.

3.3. gss_init_sec_context

```
OM_uint32  gss_init_sec_context (
                OM_uint32 *     minor_status,
                gss_cred_id_t   claimant_cred_handle,
                gss_ctx_id_t *  context_handle,
                gss_name_t      target_name,
                gss_OID         mech_type,
                int             req_flags,
                int             time_req,
                gss_channel_bindings_t
                                input_chan_bindings,
                gss_buffer_t    input_token
                gss_OID *       actual_mech_type,
                gss_buffer_t    output_token,
                int *           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 as GSS_C_NO_BUFFER. 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 indicating 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_init_sec_context via the input_token parameters.

The values returned via the ret_flags and time_rec parameters are not defined unless the routine returns GSS_S_COMPLETE.

Parameters:

claimant_cred_handle	gss_cred_id_t, read, optional handle for credentials claimed. Supply GSS_C_NO_CREDENTIAL to use default credentials.
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.

target_name gss_name_t, read
Name of target

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

req_flags bit-mask, read
Contains four 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 signed or sealed messages
False - Don't attempt to detect replayed messages
GSS_C_SEQUENCE_FLAG
True - Enable detection of out-of-sequence signed or sealed messages
False - Don't attempt to detect out-of-sequence messages

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

input_chan_bindings channel bindings, read
Application-specified bindings. Allows application to securely bind channel identification information to the security context.

input_token buffer, opaque, read, optional (see text)
Token received from peer application.
Supply GSS_C_NO_BUFFER on initial call.

actual_mech_type OID, modify
actual mechanism used.

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.

ret_flags bit-mask, modify
Contains six independent flags, each of which indicates that the context supports a specific service option. 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 - Remote peer has been asked to authenticated itself
False - Remote peer has not been asked to authenticate itself
GSS_C_REPLAY_FLAG
True - replay of signed or sealed messages will be detected
False - replayed messages will not be detected
GSS_C_SEQUENCE_FLAG
True - out-of-sequence signed or sealed 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 seal routine
False - No confidentiality service (via seal) available. seal 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_sign or gss_seal
routines.
False - Per-message integrity service
unavailable.

time_rec integer, modify, optional
number of seconds for which the context
will remain valid. If the implementation does
not support credential expiration, 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_CONTINUE_NEEDED Indicates that a token from the peer
application is required to complete thecontext, 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 signature, or a
signature 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_FAILURE Failure. See minor_status for more information

3.4. gss_accept_sec_context

```
OM_uint32  gss_accept_sec_context (
                OM_uint32 *     minor_status,
                gss_ctx_id_t *  context_handle,
                gss_cred_id_t   verifier_cred_handle,
                gss_buffer_t    input_token_buffer
                gss_channel_bindings_t
                                input_chan_bindings,
                gss_name_t *    src_name,
                gss_OID *       mech_type,
                gss_buffer_t    output_token,
                int *           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.

The values returned via the src_name, ret_flags, time_rec, and delegated_cred_handle parameters are not defined unless the routine returns GSS_S_COMPLETE.

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.

verifier_cred_handle gss_cred_id_t, read, optional
Credential handle claimed by context acceptor.
Specify GSS_C_NO_CREDENTIAL to use default credentials. If GSS_C_NO_CREDENTIAL is specified, but the caller has no default credentials established, an implementation-defined default credential may be used.

input_token_buffer buffer, opaque, read
token obtained from remote application

input_chan_bindings channel bindings, read
Application-specified bindings. Allows application to securely bind channel identification information to the security context.

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
Security mechanism used. The returned OID value will be a pointer into static storage, and should be treated as read-only by the caller.

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.

ret_flags bit-mask, modify
Contains six independent flags, each of which indicates that the context supports a specific service option. 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 signed or sealed messages will be detected
- False - replayed messages will not be detected

GSS_C_SEQUENCE_FLAG
- True - out-of-sequence signed or sealed 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 seal routine
- False - No confidentiality service (via seal) available. seal 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_sign or gss_seal routines.
- False - Per-message integrity service unavailable.

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
credential handle for credentials received from context initiator. Only valid if deleg_flag in ret_flags is true.

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 signature.

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_FAILURE Failure. See minor_status for more information.

3.5. gss_process_context_token

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

Purpose:

Provides a way to pass a token to the security service. Usually, tokens are associated either with context establishment (when they would be passed to gss_init_sec_context or gss_accept_sec_context) or with per-message security service (when they would be passed to gss_verify or gss_unseal). Occasionally, tokens may be received at other times, and gss_process_context_token allows such tokens to be passed to the underlying security service for processing. At present, such additional tokens may only be generated by gss_delete_sec_context. GSSAPI implementation may use this service to implement deletion of the security context.

Parameters:

context_handle	gss_ctx_id_t, read context handle of context on which token is to be processed
token_buffer	buffer, opaque, read pointer to first byte of token to process
minor_status	integer, modify Implementation specific status code.

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_FAILURE	Failure. See minor_status for more information
GSS_S_NO_CONTEXT	The context_handle did not refer to a valid context

3.6. 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 generate an output_token, which when passed to the peer gss_process_context_token will instruct it to do likewise. No further security services may be obtained using the context specified by context_handle.

Parameters:

minor_status	integer, modify Mechanism specific status code.
context_handle	gss_ctx_id_t, modify context handle identifying context to delete.
output_token	buffer, opaque, modify token to be sent to remote application to instruct it to also delete the context

Function value:

GSS status code:

GSS_S_COMPLETE	Successful completion
GSS_S_FAILURE	Failure, see minor_status for more information
GSS_S_NO_CONTEXT	No valid context was supplied

3.7. gss_context_time

```
OM_uint32  gss_context_time (
               OM_uint32 *     minor_status,
               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_CREDENTIALS_EXPIRED	The context is recognized, but associated credentials have expired
GSS_S_NO_CONTEXT	The context_handle parameter did not identify a valid context

3.8. gss_sign

```
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    msg_token)
```

Purpose:

Generates a cryptographic signature for the supplied message, and places the signature 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.

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 integer, 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_sign will return a major_status of GSS_S_FAILURE.

message_buffer buffer, opaque, read
message to be signed

msg_token buffer, opaque, modify
buffer to receive token

Function value:

GSS status code:

GSS_S_COMPLETE Successful completion

GSS_S_CONTEXT_EXPIRED The context has already expired

GSS_S_CREDENTIALS_EXPIRED The context is recognized, but associated credentials have expired

GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context

GSS_S_FAILURE Failure. See minor_status for more information.

3.9. gss_verify

```
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)
```

Purpose:

Verifies that a cryptographic signature, 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.

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	integer, modify quality of protection gained from signature

Function value:

GSS status code:

GSS_S_COMPLETE	Successful completion
GSS_S_DEFECTIVE_TOKEN	The token failed consistency checks
GSS_S_BAD_SIG	The signature was incorrect
GSS_S_DUPLICATE_TOKEN	The token was valid, and contained a correct signature for the message, but it had already been processed
GSS_S_OLD_TOKEN	The token was valid, and contained a correct signature for the message, but it is too old
GSS_S_UNSEQ_TOKEN	The token was valid, and contained a correct signature for the message, but has been verified out of sequence; an earlier token has been signed or sealed by the remote application, but not yet been processed locally.
GSS_S_CONTEXT_EXPIRED	The context has already expired
GSS_S_CREDENTIALS_EXPIRED	The context is recognized, but associated credentials have expired

GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context

GSS_S_FAILURE Failure. See minor_status for more information.

3.10. gss_seal

```
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)
```

Purpose:

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

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
True - Both confidentiality and integrity services are requested
False - Only integrity service is requested

qop_req integer, 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_seal will return a major_status of GSS_S_FAILURE.

input_message_buffer buffer, opaque, read
message to be sealed

conf_state boolean, modify
True - Confidentiality, data origin authentication and integrity services have been applied
False - Integrity and data origin services only has been applied.

output_message_buffer buffer, opaque, modify
buffer to receive sealed message

Function value:

GSS status code:

GSS_S_COMPLETE Successful completion

GSS_S_CONTEXT_EXPIRED The context has already expired

GSS_S_CREDENTIALS_EXPIRED The context is recognized, but associated credentials have expired

GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context

GSS_S_FAILURE Failure. See minor_status for more information.

3.11. gss_unseal

```
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)
```

Purpose:

Converts a previously sealed message back to a usable form, verifying the embedded signature. 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.

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
sealed message

output_message_buffer buffer, opaque, modify
buffer to receive unsealed message

conf_state boolean, modify
True - Confidentiality and integrity protection were used
False - Inteegrity service only was used

qop_state integer, modify
quality of protection gained from signature

Function value:

GSS status code:

GSS_S_COMPLETE Successful completion

GSS_S_DEFECTIVE_TOKEN The token failed consistency checks

GSS_S_BAD_SIG The signature was incorrect

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

GSS_S_OLD_TOKEN The token was valid, and contained a correct signature for the message, but it is too old

GSS_S_UNSEQ_TOKEN The token was valid, and contained a correct signature for the message, but has been verified out of sequence; an earlier token has been signed or sealed by the remote application, but not yet been processed locally.

GSS_S_CONTEXT_EXPIRED The context has already expired

GSS_S_CREDENTIALS_EXPIRED The context is recognized, but associated credentials have expired

GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context

GSS_S_FAILURE Failure. See minor_status for more information.

3.12. gss_display_status

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

Purpose:

Allows an application to obtain a textual representation of a GSSAPI status code, for display to the user or for logging purposes. Since some status values may indicate multiple errors, applications may need to call gss_display_status multiple times, each call generating a single text string. The message_context parameter is used to indicate which error message should be extracted from a given status_value; message_context should be initialized to 0, and gss_display_status will return a non-zero value if there are further messages to extract.

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_NULL_OID to obtain the system default.
message_context	integer, read/modify Should be initialized to zero by caller

on first call. If further messages are contained in the status_value parameter, message_context will be non-zero on return, and this value should be passed back to subsequent calls, along with the same status_value, status_type and mech_type parameters.

status_string buffer, character string, modify
textual interpretation of the status_value

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.

3.13. 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 pointer into static storage, and should be treated as read-only by the caller.

Function value:

GSS status code:

GSS_S_COMPLETE	Successful completion

3.14. gss_compare_name

```
OM_uint32  gss_compare_name (
               OM_uint32 *     minor_status,
               gss_name_t      name1,
               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.

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 True - names refer to same entity False - 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 type contained within either name1 or name2 was unrecognized, or the names were of incomparable types.
GSS_S_BAD_NAME	One or both of name1 or name2 was ill-formed

3.15. gss_display_name

```
OM_uint32  gss_display_name (
               OM_uint32 *     minor_status,
               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 GSSAPI implementation.

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
output_name_type	Object ID, modify 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

Function value:

GSS status code:

GSS_S_COMPLETE	Successful completion
GSS_S_BAD_NAMETYPE	The type of input_name was not recognized
GSS_S_BAD_NAME	input_name was ill-formed

3.16. gss_import_name

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

Purpose:

Convert a printable name to internal form.

Parameters:

minor_status	integer, modify	Mechanism specific status code
input_name_buffer	buffer, character-string, read	buffer containing printable name to convert
input_name_type	Object ID, read, optional	Object Id specifying type of printable name. Applications may specify either GSS_C_NULL_OID to use a local system-specific printable syntax, or an OID registered by the GSSAPI implementation to name a particular namespace.
output_name	gss_name_t, modify	returned name in internal form

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

3.17. 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.

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

3.18. gss_release_buffer

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

Purpose:

Free storage associated with a buffer format name. The storage must have been allocated by a GSSAPI routine. In addition to freeing the associated storage, the routine will zero the length field in the buffer parameter.

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

3.19. gss_release_oid_set

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

Purpose:

Free storage associated with a gss_OID_set object. The storage must have been allocated by a GSSAPI 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

3.20. gss_inquire_cred

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

Purpose:

Obtains information about a credential. The caller must already have obtained a handle that refers to the 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 credential.
name	gss_name_t, modify	The name whose identity the credential asserts. Specify NULL if not required.
lifetime	Integer, modify	

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 — Integer, modify
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
Set of mechanisms supported by the credential. 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.

```
#ifndef GSSAPI_H_
#define GSSAPI_H_

/*
 * First, define the platform-dependent types.
 */
typedef <platform-specific> OM_uint32;
typedef <platform-specific> gss_ctx_id_t;
typedef <platform-specific> gss_cred_id_t;
typedef <platform-specific> gss_name_t;
```

```
  /*
   * Note that a platform supporting the xom.h X/Open header file
   * may make use of that header for the definitions of OM_uint32
   * and the structure to which gss_OID_desc equates.
   */

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

  typedef struct gss_OID_set_desc_struct  {
        int     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;


  /*
   * Six independent flags each of which indicates that a context
   * supports a specific service option.
   */
  #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


  /*
   * 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;

  #define GSS_C_NO_BUFFER ((gss_buffer_t) 0)
  #define GSS_C_NULL_OID ((gss_OID) 0)
  #define GSS_C_NULL_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}

  /*
   * Define the default Quality of Protection for per-message
   * services.  Note that an implementation that offers multiple
   * levels of QOP may either reserve a value (for example zero,
   * as assumed here) to mean "default protection", or alternatively
   * may simply equate GSS_C_QOP_DEFAULT to a specific explicit QOP
   * value.
```

```
   */
  #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 0xffffffffl


  /* 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
   */
  #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) \
    ((GSS_CALLING_ERROR(x) != 0) || (GSS_ROUTINE_ERROR(x) != 0))


  /*
   * 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_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)

  /*
   * 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))


  /*
   * Finally, function prototypes for the GSSAPI routines.
   */

  OM_uint32 gss_acquire_cred
             (OM_uint32*,       /* minor_status */
              gss_name_t,       /* desired_name */
              OM_uint32,        /* time_req */
              gss_OID_set,      /* desired_mechs */
              int,              /* 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 */
             gss_cred_id_t,    /* claimant_cred_handle */
             gss_ctx_id_t*,    /* context_handle */
             gss_name_t,       /* target_name */
             gss_OID,          /* mech_type */
             int,              /* req_flags */
             OM_uint32,        /* time_req */
             gss_channel_bindings_t,
                               /* input_chan_bindings */
             gss_buffer_t,     /* input_token */
             gss_OID*,         /* actual_mech_type */
             gss_buffer_t,     /* output_token */
             int*,             /* ret_flags */
             OM_uint32*        /* time_rec */
            );

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

  OM_uint32 gss_process_context_token
            (OM_uint32*,       /* minor_status */
             gss_ctx_id_t,     /* context_handle */
             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 */
             gss_ctx_id_t,     /* context_handle */
             OM_uint32*        /* time_rec */
            );

  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_uitn32 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 */
            );
```

```
  OM_uint32 gss_display_status
            (OM_uint32*,       /* minor_status */
             OM_uint32,        /* status_value */
             int,              /* status_type */
             gss_OID,          /* mech_type */
             int*,             /* 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 */
             gss_name_t,       /* name1 */
             gss_name_t,       /* name2 */
             int*              /* name_equal */
            );

  OM_uint32 gss_display_name,
            (OM_uint32*,      /* minor_status */
             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 */
             gss_buffer_t,     /* input_name_buffer */
             gss_OID,          /* input_name_type */
             gss_name_t*       /* output_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 */
             gss_cred_id_t,    /* cred_handle */
             gss_name_t *,     /* name */
             OM_uint32 *,      /* lifetime */
             int *,            /* cred_usage */
             gss_OID_set *     /* mechanisms */
            );



  #endif /* GSSAPI_H_ */
```

References

[1] Linn, J., "Generic Security Service Application Program Interface", RFC 1508, Geer Zolot Associate, September 1993.

[2] "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.

Security Considerations

Security issues are discussed throughout this memo.

Author's Address

John Wray
Digital Equipment Corporation
550 King Street, LKG2-2/AA6
Littleton, MA 01460
USA

Phone: +1-508-486-5210
EMail: Wray@tuxedo.enet.dec.com