GNU Generic Security Service (GSS) API Reference Manual

COLLABORATORS

	TITLE : GNU Generic Security Service (GSS) API Reference Manual	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 9, 2014	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

GNU Generic Security Service (GSS) API Reference Manual

GSS is an implementation of the Generic Security Service Application Program Interface (GSS-API). GSS-API is used by network servers to provide security services, e.g., to authenticate SMTP/IMAP clients against SMTP/IMAP servers. GSS consists of a library and a manual.

GSS is developed for the GNU/Linux system, but runs on over 20 platforms including most major Unix platforms and Windows, and many kind of devices including iPAQ handhelds and S/390 mainframes.

GSS is a GNU project, and is licensed under the GNU General Public License version 3 or later.

1.1 gss

gss —

Types and Values

#define	GSS_VERSION
#define	GSS_VERSION_MAJOR
#define	GSS_VERSION_MINOR
#define	GSS_VERSION_PATCH
#define	GSS_VERSION_NUMBER

Description

Functions

Types and Values

GSS_VERSION

```
# define GSS_VERSION "1.0.3"
```

Pre-processor symbol with a string that describe the header file version number. Used together with gss_check_version() to verify header file and run-time library consistency.

GSS_VERSION_MAJOR

```
# define GSS_VERSION_MAJOR 1
```

Pre-processor symbol with a decimal value that describe the major level of the header file version number. For example, when the header version is 1.2.3 this symbol will be 1.

GSS_VERSION_MINOR

```
# define GSS_VERSION_MINOR 0
```

Pre-processor symbol with a decimal value that describe the minor level of the header file version number. For example, when the header version is 1.2.3 this symbol will be 2.

GSS_VERSION_PATCH

```
# define GSS_VERSION_PATCH 3
```

Pre-processor symbol with a decimal value that describe the patch level of the header file version number. For example, when the header version is 1.2.3 this symbol will be 3.

GSS_VERSION_NUMBER

```
# define GSS_VERSION_NUMBER 0x010003
```

Pre-processor symbol with a hexadecimal value describing the header file version number. For example, when the header version is 1.2.3 this symbol will have the value 0x010203.

1.2 api

api —

Functions

#define	GSS_C_NO_NAME
#define	GSS_C_NO_BUFFER
#define	GSS_C_NO_OID
#define	GSS_C_NO_OID_SET
#define	GSS_C_NO_CONTEXT
#define	GSS_C_NO_CREDENTIAL
#define	GSS_C_NO_CHANNEL_BINDINGS
#define	GSS_CALLING_ERROR()
#define	GSS_ROUTINE_ERROR()
#define	GSS_SUPPLEMENTARY_INFO()
#define	GSS_ERROR()
#define	GSS_S_BAD_MECH
#define	GSS_S_BAD_NAME
#define	GSS_S_BAD_NAMETYPE
#define	GSS_S_BAD_BINDINGS
#define	GSS_S_BAD_STATUS
#define	GSS_S_BAD_SIG

#define	GSS_S_NO_CRED
#define	GSS_S_NO_CONTEXT
#define	GSS_S_DEFECTIVE_TOKEN
#define	GSS_S_DEFECTIVE_CREDENTIAL
#define	GSS_S_CREDENTIALS_EXPIRED
#define	GSS_S_CONTEXT_EXPIRED
#define	GSS_S_FAILURE
#define	GSS_S_BAD_QOP
#define	GSS_S_UNAUTHORIZED
#define	GSS_S_UNAVAILABLE
#define	GSS_S_DUPLICATE_ELEMENT
#define	GSS_S_NAME_NOT_MN
OM_uint32	gss_acquire_cred ()
OM_uint32	gss_release_cred ()
OM_uint32	gss_init_sec_context ()
OM_uint32	gss_accept_sec_context ()
OM_uint32	gss_process_context_token ()
OM_uint32	gss_delete_sec_context ()
OM_uint32	gss_context_time ()
OM_uint32	gss_get_mic ()
OM_uint32	gss_verify_mic ()
OM_uint32	gss_wrap ()
OM_uint32	gss_unwrap ()
OM_uint32	gss_display_status ()
OM_uint32	gss_indicate_mechs ()
OM_uint32	gss_compare_name ()
OM_uint32	gss_display_name ()
OM_uint32	gss_import_name ()
OM_uint32	gss_export_name ()
OM_uint32	gss_release_name ()
OM_uint32	gss_release_buffer ()
OM_uint32	gss_release_oid_set ()
OM_uint32	gss_inquire_cred ()
OM_uint32	gss_inquire_context ()
OM_uint32	gss_wrap_size_limit ()
OM_uint32	gss_add_cred ()
OM_uint32	gss_inquire_cred_by_mech ()
OM_uint32	gss_export_sec_context ()
OM_uint32	gss_import_sec_context ()
OM_uint32	gss_create_empty_oid_set ()
OM_uint32	gss_add_oid_set_member ()
OM_uint32	gss_test_oid_set_member ()
OM_uint32	gss_inquire_names_for_mech ()
OM_uint32	gss_inquire_mechs_for_name ()
OM_uint32	gss_canonicalize_name ()
OM_uint32	gss_duplicate_name ()
OM_uint32	gss_sign ()
OM_uint32	gss_verify ()
OM_uint32	gss_seal ()
OM_uint32	gss_unseal ()
OM_uint32	gss_inquire_saslname_for_mech ()
OM_uint32	gss_inquire_mech_for_saslname ()
int	gss_oid_equal ()
OM_uint32	gss_encapsulate_token ()
OM_uint32	gss_decapsulate_token ()

Types and Values

typedef	gss_ctx_id_t
typedef	gss_cred_id_t
typedef	gss_name_t
typedef	gss_uint32
typedef	OM_uint32
typedef	gss_qop_t
typedef	gss_cred_usage_t
#define	GSS_C_DELEG_FLAG
#define	GSS_C_MUTUAL_FLAG
#define	GSS_C_REPLAY_FLAG
#define	GSS_C_SEQUENCE_FLAG
#define	GSS_C_CONF_FLAG
#define	GSS_C_INTEG_FLAG
#define	GSS_C_ANON_FLAG
#define	GSS_C_PROT_READY_FLAG
#define	GSS_C_TRANS_FLAG
#define	GSS_C_BOTH
#define	GSS_C_INITIATE
#define	GSS_C_ACCEPT
#define	GSS_C_GSS_CODE
#define	GSS_C_MECH_CODE
#define	GSS_C_AF_UNSPEC
#define	GSS_C_AF_LOCAL
#define	GSS_C_AF_INET
#define	GSS_C_AF_IMPLINK
#define	GSS_C_AF_PUP
#define	GSS_C_AF_CHAOS
#define	GSS_C_AF_NS
#define	GSS_C_AF_NBS
#define	GSS_C_AF_ECMA
#define	GSS_C_AF_DATAKIT
#define	GSS_C_AF_CCITT
#define	GSS_C_AF_SNA
#define	GSS_C_AF_DECnet
#define	GSS_C_AF_DLI
#define	GSS_C_AF_LAT
#define	GSS_C_AF_HYLINK
#define	GSS_C_AF_APPLETALK
#define	GSS_C_AF_BSC
#define	GSS_C_AF_DSS
#define	GSS_C_AF_OSI
#define	GSS_C_AF_X25
#define	GSS_C_AF_NULLADDR
#define	GSS_C_EMPTY_BUFFER
#define	GSS_C_NULL_OID
#define	GSS_C_NULL_OID_SET
#define	GSS_C_QOP_DEFAULT
#define	GSS_C_INDEFINITE
extern gss_OID	GSS_C_NT_USER_NAME
extern gss_OID	GSS_C_NT_MACHINE_UID_NAME
extern gss_OID	GSS_C_NT_STRING_UID_NAME
extern gss_OID	GSS_C_NT_HOSTBASED_SERVICE_X
extern gss_OID	GSS_C_NT_HOSTBASED_SERVICE
extern gss_OID	GSS_C_NT_ANONYMOUS

extern gss_OID	GSS_C_NT_EXPORT_NAME
#define	GSS_S_COMPLETE
#define	GSS_C_CALLING_ERROR_OFFSET
#define	GSS_C_ROUTINE_ERROR_OFFSET
#define	GSS_C_SUPPLEMENTARY_OFFSET
#define	GSS_C_CALLING_ERROR_MASK
#define	GSS_C_ROUTINE_ERROR_MASK
#define	GSS_C_SUPPLEMENTARY_MASK
#define	GSS_S_CALL_INACCESSIBLE_READ
#define	GSS_S_CALL_INACCESSIBLE_WRITE
#define	GSS_S_CALL_BAD_STRUCTURE
#define	GSS_S_BAD_MIC
#define	GSS_S_CONTINUE_NEEDED
#define	GSS_S_DUPLICATE_TOKEN
#define	GSS_S_OLD_TOKEN
#define	GSS_S_UNSEQ_TOKEN
#define	GSS_S_GAP_TOKEN
typedef	gss_const_buffer_t
typedef	gss_const_ctx_id_t
typedef	gss_const_cred_id_t
typedef	gss_const_name_t
typedef	gss_const_OID
typedef	gss_const_OID_set

Description

Functions

GSS_C_NO_NAME

```
#define GSS_C_NO_NAME ((gss_name_t) 0)
```

GSS_C_NO_BUFFER

```
#define GSS_C_NO_BUFFER ((gss_buffer_t) 0)
```

GSS_C_NO_OID

```
#define GSS_C_NO_OID ((gss_OID) 0)
```

GSS_C_NO_OID_SET

```
#define GSS_C_NO_OID_SET ((gss_OID_set) 0)
```

GSS_C_NO_CONTEXT

```
#define GSS_C_NO_CONTEXT ((gss_ctx_id_t) 0)
```

GSS_C_NO_CREDENTIAL

```
#define GSS_C_NO_CREDENTIAL ((gss_cred_id_t) 0)
```

GSS_C_NO_CHANNEL_BINDINGS

```
#define GSS_C_NO_CHANNEL_BINDINGS ((gss_channel_bindings_t) 0)
```

GSS_CALLING_ERROR()

```
#define             GSS_CALLING_ERROR(x)
```

GSS_ROUTINE_ERROR()

```
#define             GSS_ROUTINE_ERROR(x)
```

GSS_SUPPLEMENTARY_INFO()

```
#define             GSS_SUPPLEMENTARY_INFO(x)
```

GSS_ERROR()

```
#define             GSS_ERROR(x)
```

GSS_S_BAD_MECH

```
#define GSS_S_BAD_MECH              (1ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_BAD_NAME

```
#define GSS_S_BAD_NAME              (2ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_BAD_NAMETYPE

```
#define GSS_S_BAD_NAMETYPE          (3ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_BAD_BINDINGS

```
#define GSS_S_BAD_BINDINGS          (4ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_BAD_STATUS

```
#define GSS_S_BAD_STATUS            (5ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_BAD_SIG

```
#define GSS_S_BAD_SIG                (6ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_NO_CRED

```
#define GSS_S_NO_CRED                (7ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_NO_CONTEXT

```
#define GSS_S_NO_CONTEXT             (8ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_DEFECTIVE_TOKEN

```
#define GSS_S_DEFECTIVE_TOKEN        (9ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_DEFECTIVE_CREDENTIAL

```
#define GSS_S_DEFECTIVE_CREDENTIAL (10ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_CREDENTIALS_EXPIRED

```
#define GSS_S_CREDENTIALS_EXPIRED  (11ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_CONTEXT_EXPIRED

```
#define GSS_S_CONTEXT_EXPIRED        (12ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_FAILURE

```
#define GSS_S_FAILURE                (13ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_BAD_QOP

```
#define GSS_S_BAD_QOP                (14ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_UNAUTHORIZED

```
#define GSS_S_UNAUTHORIZED           (15ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_UNAVAILABLE

```
#define GSS_S_UNAVAILABLE            (16ul << GSS_C_ROUTINE_ERROR_OFFSET)
```

GSS_S_DUPLICATE_ELEMENT

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

GSS_S_NAME_NOT_MN

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

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);
```

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

minor_status	(integer, modify) Mechanism specific status code.	
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.	

Returns

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.

gss_release_cred ()

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

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

Parameters

minor_status	(Integer, modify) Mechanism specific status code.	
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.	

Returns

GSS_S_COMPLETE: Successful completion.

GSS_S_NO_CRED: Credentials could not be accessed.

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);
```

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(m
{

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; };

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(m
{

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.

The `req_flags` values:

`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.

The `ret_flags` values:

`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 transferable.

All other bits should be set to zero.

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. See below for the flags.	
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) 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. See below for the flags.	

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.	

Returns

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.

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,
                        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);
```

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; };

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.

The `ret_flags` values:

`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 transferable.

All other bits should be set to zero.

Parameters

minor_status	(Integer, modify) Mechanism specific status code.	
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. See below for the flags.	
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_sec_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.

Returns

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.

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);
```

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 acceptor's 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.	

Returns

`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.

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);
```

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.	

Returns

`GSS_S_COMPLETE`: Successful completion.

`GSS_S_NO_CONTEXT`: No valid context was supplied.

gss_context_time ()

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

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.	

Returns

`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_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 message_token);
```

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) Mechanism 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.	
message_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().	

Returns

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.

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);
```

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.	

Returns

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.

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);
```

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().	

Returns

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.

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);
```

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.	

Returns

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.

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);
```

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);

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().	

Returns

`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_C

gss_indicate_mechs ()

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

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().	

Returns

GSS_S_COMPLETE: Successful completion.

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);
```

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).	

Returns

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.

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);
```

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.	

Returns

`GSS_S_COMPLETE`: Successful completion.

`GSS_S_BAD_NAME`: *`input_name`* was ill-formed.

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);
```

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().	

Returns

`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.

gss_export_name ()

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

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().	

Returns

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.

gss_release_name ()

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

Free GSSAPI-allocated storage associated with an internal-form name. The name is set 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.	

Returns

GSS_S_COMPLETE: Successful completion.

GSS_S_BAD_NAME: The name parameter did not contain a valid name.

gss_release_buffer ()

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

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.	

Returns

GSS_S_COMPLETE: Successful completion.

gss_release_oid_set ()

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

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.

The gss_OID_set parameter is set 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.	

Returns

GSS_S_COMPLETE: Successful completion.

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);
```

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.	

Returns

`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.

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);
```

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.

The `ctx_flags` values:

`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 transferable.

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_context(), 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. See below for the flags.	
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.	

Returns

GSS_S_COMPLETE: Successful completion.

GSS_S_NO_CONTEXT: The referenced context could not be accessed.

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);
```

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.

Returns

`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.

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);
```

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 text). 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	

Returns

`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.

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);
```

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.	

Returns

`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.

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);
```

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 successful, 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().	

Returns

`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.

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);
```

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().

Returns

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.

gss_create_empty_oid_set ()

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

Create an object-identifier set containing no object identifiers, to which members may be subsequently added using the gss_add_oid_set_m 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().

Returns

GSS_S_COMPLETE: Successful completion.

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);
```

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.	

Returns

GSS_S_COMPLETE: Successful completion.

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);
```

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.	

Returns

GSS_S_COMPLETE: Successful completion.

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);
```

Returns the set of nametypes supported by the specified mechanism.

Parameters

minor_status	(Integer, modify) Mechanism 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().	

Returns

GSS_S_COMPLETE: Successful completion.

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);
```

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 prefilter for a call to a subsequent mechanism-specific routine.

Parameters

minor_status	(Integer, modify) Mechanism 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().	

Returns

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.

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);
```

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().	

Returns

GSS_S_COMPLETE: Successful completion.

gss_duplicate_name ()

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

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().	

Returns

GSS_S_COMPLETE: Successful completion.

GSS_S_BAD_NAME: The src_name parameter was ill-formed.

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 message_token);
```

Returns

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);
```

Returns

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);
```

Returns

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);
```

Returns

gss_inquire_saslname_for_mech ()

```
OM_uint32
gss_inquire_saslname_for_mech (OM_uint32 *minor_status,
                               const gss_OID desired_mech,
                               gss_buffer_t sasl_mech_name,
                               gss_buffer_t mech_name,
                               gss_buffer_t mech_description);
```

Output the SASL mechanism name of a GSS-API mechanism. It also returns a name and description of the mechanism in a user friendly form.

Parameters

minor_status	(Integer, modify) Mechanism specific status code.	
desired_mech	(OID, read) Identifies the GSS-API mechanism to query.	
sasl_mech_name	(buffer, character-string, modify, optional) Buffer to receive SASL mechanism name. The application must free storage associated with this name after use with a call to gss_release_buffer().	

mech_name	(buffer, character-string, modify, optional) Buffer to receive human readable mechanism name. The application must free storage associated with this name after use with a call to gss_release_buffer().	
mech_description	(buffer, character-string, modify, optional) Buffer to receive description of mechanism. The application must free storage associated with this name after use with a call to gss_release_buffer().	

Returns

GSS_S_COMPLETE: Successful completion.

GSS_S_BAD_MECH: The *desired_mech* OID is unsupported.

gss_inquire_mech_for_saslname ()

```
OM_uint32
gss_inquire_mech_for_saslname (OM_uint32 *minor_status,
                               const gss_buffer_t sasl_mech_name,
                               gss_OID *mech_type);
```

Output GSS-API mechanism OID of mechanism associated with given *sasl_mech_name* .

Parameters

minor_status	(Integer, modify) Mechanism specific status code.	
sasl_mech_name	(buffer, character-string, read) Buffer with SASL mechanism name.	
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.	

Returns

GSS_S_COMPLETE: Successful completion.

GSS_S_BAD_MECH: There is no GSS-API mechanism known as *sasl_mech_name* .

gss_oid_equal ()

```
int
gss_oid_equal (gss_const_OID first_oid,
               gss_const_OID second_oid);
```

Compare two OIDs for equality. The comparison is "deep", i.e., the actual byte sequences of the OIDs are compared instead of just the pointer equality. This function is standardized in RFC 6339.

Parameters

first_oid	(Object ID, read) First Object identifier.	
second_oid	(Object ID, read) First Object identifier.	

Returns

Returns boolean value true when the two OIDs are equal, otherwise false.

gss_encapsulate_token ()

```
OM_uint32
gss_encapsulate_token (gss_const_buffer_t input_token,
                       gss_const_OID token_oid,
                       gss_buffer_t output_token);
```

Add the mechanism-independent token header to GSS-API context token data. This is used for the initial token of a GSS-API context establishment sequence. It incorporates an identifier of the mechanism type to be used on that context, and enables tokens to be interpreted unambiguously at GSS-API peers. See further section 3.1 of RFC 2743. This function is standardized in RFC 6339.

Parameters

input_token	(buffer, opaque, read) Buffer with GSS-API context token data.	
token_oid	(Object ID, read) Object identifier of token.	
output_token	(buffer, opaque, modify) Encapsulated token data; caller must release with gss_release_buffer().	

Returns

GSS_S_COMPLETE: Indicates successful completion, and that output parameters holds correct information.

GSS_S_FAILURE: Indicates that encapsulation failed for reasons unspecified at the GSS-API level.

gss_decapsulate_token ()

```
OM_uint32
```

```
gss_decapsulate_token (gss_const_buffer_t input_token,
                       gss_const_OID token_oid,
                       gss_buffer_t output_token);
```

Remove the mechanism-independent token header from an initial GSS-API context token. Unwrap a buffer in the mechanism-independent token format. This is the reverse of gss_encapsulate_token(). The translation is loss-less, all data is preserved as is. This function is standardized in RFC 6339.

Parameters

input_token	(buffer, opaque, read) Buffer with GSS-API context token.	
token_oid	(Object ID, read) Expected object identifier of token.	
output_token	(buffer, opaque, modify) Decapsulated token data; caller must release with gss_release_buffer().	

Returns

GSS_S_COMPLETE: Indicates successful completion, and that output parameters holds correct information.

GSS_S_DEFECTIVE_TOKEN: Means that the token failed consistency checks (e.g., OID mismatch or ASN.1 DER length errors).

GSS_S_FAILURE: Indicates that decapsulation failed for reasons unspecified at the GSS-API level.

Types and Values

gss_ctx_id_t

```
typedef struct gss_ctx_id_struct *gss_ctx_id_t;
```

gss_cred_id_t

```
typedef struct gss_cred_id_struct *gss_cred_id_t;
```

gss_name_t

```
typedef struct gss_name_struct *gss_name_t;
```

gss_uint32

```
typedef unsigned short gss_uint32;
```

OM_uint32

```
typedef gss_uint32 OM_uint32;
```

gss_qop_t

```
typedef OM_uint32 gss_qop_t;
```

gss_cred_usage_t

```
typedef int gss_cred_usage_t;
```

GSS_C_DELEG_FLAG

```
#define GSS_C_DELEG_FLAG      1
```

GSS_C_MUTUAL_FLAG

```
#define GSS_C_MUTUAL_FLAG     2
```

GSS_C_REPLAY_FLAG

```
#define GSS_C_REPLAY_FLAG     4
```

GSS_C_SEQUENCE_FLAG

```
#define GSS_C_SEQUENCE_FLAG   8
```

GSS_C_CONF_FLAG

```
#define GSS_C_CONF_FLAG       16
```

GSS_C_INTEG_FLAG

```
#define GSS_C_INTEG_FLAG      32
```

GSS_C_ANON_FLAG

```
#define GSS_C_ANON_FLAG       64
```

GSS_C_PROT_READY_FLAG

```
#define GSS_C_PROT_READY_FLAG 128
```

GSS_C_TRANS_FLAG

```
#define GSS_C_TRANS_FLAG      256
```

GSS_C_BOTH

```
#define GSS_C_BOTH      0
```

GSS_C_INITIATE

```
#define GSS_C_INITIATE 1
```

GSS_C_ACCEPT

```
#define GSS_C_ACCEPT    2
```

GSS_C_GSS_CODE

```
#define GSS_C_GSS_CODE  1
```

GSS_C_MECH_CODE

```
#define GSS_C_MECH_CODE 2
```

GSS_C_AF_UNSPEC

```
#define GSS_C_AF_UNSPEC     0
```

GSS_C_AF_LOCAL

```
#define GSS_C_AF_LOCAL      1
```

GSS_C_AF_INET

```
#define GSS_C_AF_INET       2
```

GSS_C_AF_IMPLINK

```
#define GSS_C_AF_IMPLINK    3
```

GSS_C_AF_PUP

```
#define GSS_C_AF_PUP        4
```

GSS_C_AF_CHAOS

```
#define GSS_C_AF_CHAOS      5
```

GSS_C_AF_NS

```
#define GSS_C_AF_NS          6
```

GSS_C_AF_NBS

```
#define GSS_C_AF_NBS         7
```

GSS_C_AF_ECMA

```
#define GSS_C_AF_ECMA        8
```

GSS_C_AF_DATAKIT

```
#define GSS_C_AF_DATAKIT     9
```

GSS_C_AF_CCITT

```
#define GSS_C_AF_CCITT       10
```

GSS_C_AF_SNA

```
#define GSS_C_AF_SNA         11
```

GSS_C_AF_DECnet

```
#define GSS_C_AF_DECnet      12
```

GSS_C_AF_DLI

```
#define GSS_C_AF_DLI         13
```

GSS_C_AF_LAT

```
#define GSS_C_AF_LAT         14
```

GSS_C_AF_HYLINK

```
#define GSS_C_AF_HYLINK      15
```

GSS_C_AF_APPLETALK

```
#define GSS_C_AF_APPLETALK   16
```

GSS_C_AF_BSC

```
#define GSS_C_AF_BSC        17
```

GSS_C_AF_DSS

```
#define GSS_C_AF_DSS        18
```

GSS_C_AF_OSI

```
#define GSS_C_AF_OSI        19
```

GSS_C_AF_X25

```
#define GSS_C_AF_X25        21
```

GSS_C_AF_NULLADDR

```
#define GSS_C_AF_NULLADDR   255
```

GSS_C_EMPTY_BUFFER

```
#define GSS_C_EMPTY_BUFFER {0, NULL}
```

GSS_C_NULL_OID

```
#define GSS_C_NULL_OID GSS_C_NO_OID
```

GSS_C_NULL_OID_SET

```
#define GSS_C_NULL_OID_SET GSS_C_NO_OID_SET
```

GSS_C_QOP_DEFAULT

```
#define GSS_C_QOP_DEFAULT 0
```

GSS_C_INDEFINITE

```
#define GSS_C_INDEFINITE 0xfffffffful
```

GSS_C_NT_USER_NAME

```
extern gss_OID GSS_C_NT_USER_NAME;
```

GSS_C_NT_MACHINE_UID_NAME

```
extern gss_OID GSS_C_NT_MACHINE_UID_NAME;
```

GSS_C_NT_STRING_UID_NAME

```
extern gss_OID GSS_C_NT_STRING_UID_NAME;
```

GSS_C_NT_HOSTBASED_SERVICE_X

```
extern gss_OID GSS_C_NT_HOSTBASED_SERVICE_X;
```

GSS_C_NT_HOSTBASED_SERVICE

```
extern gss_OID GSS_C_NT_HOSTBASED_SERVICE;
```

GSS_C_NT_ANONYMOUS

```
extern gss_OID GSS_C_NT_ANONYMOUS;
```

GSS_C_NT_EXPORT_NAME

```
extern gss_OID GSS_C_NT_EXPORT_NAME;
```

GSS_S_COMPLETE

```
#define GSS_S_COMPLETE 0
```

GSS_C_CALLING_ERROR_OFFSET

```
#define GSS_C_CALLING_ERROR_OFFSET 24
```

GSS_C_ROUTINE_ERROR_OFFSET

```
#define GSS_C_ROUTINE_ERROR_OFFSET 16
```

GSS_C_SUPPLEMENTARY_OFFSET

```
#define GSS_C_SUPPLEMENTARY_OFFSET 0
```

GSS_C_CALLING_ERROR_MASK

```
#define GSS_C_CALLING_ERROR_MASK 0377ul
```

GSS_C_ROUTINE_ERROR_MASK

```
#define GSS_C_ROUTINE_ERROR_MASK 0377ul
```

GSS_C_SUPPLEMENTARY_MASK

```
#define GSS_C_SUPPLEMENTARY_MASK 0177777ul
```

GSS_S_CALL_INACCESSIBLE_READ

```
#define GSS_S_CALL_INACCESSIBLE_READ~(1ul << GSS_C_CALLING_ERROR_OFFSET)
```

GSS_S_CALL_INACCESSIBLE_WRITE

```
#define GSS_S_CALL_INACCESSIBLE_WRITE~(2ul << GSS_C_CALLING_ERROR_OFFSET)
```

GSS_S_CALL_BAD_STRUCTURE

```
#define GSS_S_CALL_BAD_STRUCTURE~(3ul << GSS_C_CALLING_ERROR_OFFSET)
```

GSS_S_BAD_MIC

```
#define GSS_S_BAD_MIC GSS_S_BAD_SIG
```

GSS_S_CONTINUE_NEEDED

```
#define GSS_S_CONTINUE_NEEDED~(1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 0))
```

GSS_S_DUPLICATE_TOKEN

```
#define GSS_S_DUPLICATE_TOKEN~(1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 1))
```

GSS_S_OLD_TOKEN

```
#define GSS_S_OLD_TOKEN    (1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 2))
```

GSS_S_UNSEQ_TOKEN

```
#define GSS_S_UNSEQ_TOKEN~(1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 3))
```

GSS_S_GAP_TOKEN

```
#define GSS_S_GAP_TOKEN    (1ul << (GSS_C_SUPPLEMENTARY_OFFSET + 4))
```

gss_const_buffer_t

```
typedef const gss_buffer_desc *gss_const_buffer_t;
```

gss_const_ctx_id_t

```
typedef const struct gss_ctx_id_struct *gss_const_ctx_id_t;
```

gss_const_cred_id_t

```
typedef const struct gss_cred_id_struct *gss_const_cred_id_t;
```

gss_const_name_t

```
typedef const struct gss_name_struct *gss_const_name_t;
```

gss_const_OID

```
typedef const gss_OID_desc *gss_const_OID;
```

gss_const_OID_set

```
typedef const gss_OID_set_desc *gss_const_OID_set;
```

1.3 ext

ext —

Functions

const char *	gss_check_version ()
int	gss_userok ()

Types and Values

extern gss_OID_desc	GSS_C_NT_USER_NAME_static
extern gss_OID_desc	GSS_C_NT_MACHINE_UID_NAME_static
extern gss_OID_desc	GSS_C_NT_STRING_UID_NAME_static
extern gss_OID_desc	GSS_C_NT_HOSTBASED_SERVICE_X_static
extern gss_OID_desc	GSS_C_NT_HOSTBASED_SERVICE_static
extern gss_OID_desc	GSS_C_NT_ANONYMOUS_static
extern gss_OID_desc	GSS_C_NT_EXPORT_NAME_static

Description

Functions

gss_check_version ()

```
const char~*
gss_check_version (const char *req_version);
```

Check that the version of the library is at minimum the one given as a string in *req_version* .

WARNING: This function is a GNU GSS specific extension, and is not part of the official GSS API.

Parameters

req_version	version string to compare with, or NULL	

Returns

The actual version string of the library; NULL if the condition is not met. If NULL is passed to this function no check is done and only the version string is returned.

gss_userok ()

```
int
gss_userok (const gss_name_t name,
            const char *username);
```

Compare the username against the output from gss_export_name() invoked on *name* , after removing the leading OID. This answers the question whether the particular mechanism would authenticate them as the same principal

WARNING: This function is a GNU GSS specific extension, and is not part of the official GSS API.

Parameters

name	(gss_name_t, read) Name to be compared.	
username	Zero terminated string with username.	

Returns

Returns 0 if the names match, non-0 otherwise.

Types and Values

GSS_C_NT_USER_NAME_static

```
extern gss_OID_desc GSS_C_NT_USER_NAME_static;
```

GSS_C_NT_MACHINE_UID_NAME_static

```
extern gss_OID_desc GSS_C_NT_MACHINE_UID_NAME_static;
```

GSS_C_NT_STRING_UID_NAME_static

```
extern gss_OID_desc GSS_C_NT_STRING_UID_NAME_static;
```

GSS_C_NT_HOSTBASED_SERVICE_X_static

```
extern gss_OID_desc GSS_C_NT_HOSTBASED_SERVICE_X_static;
```

GSS_C_NT_HOSTBASED_SERVICE_static

```
extern gss_OID_desc GSS_C_NT_HOSTBASED_SERVICE_static;
```

GSS_C_NT_ANONYMOUS_static

```
extern gss_OID_desc GSS_C_NT_ANONYMOUS_static;
```

GSS_C_NT_EXPORT_NAME_static

```
extern gss_OID_desc GSS_C_NT_EXPORT_NAME_static;
```

1.4 krb5

krb5 —

Types and Values

#define	GSS_KRB5_S_G_BAD_SERVICE_NAME
#define	GSS_KRB5_S_G_BAD_STRING_UID
#define	GSS_KRB5_S_G_NOUSER
#define	GSS_KRB5_S_G_VALIDATE_FAILED
#define	GSS_KRB5_S_G_BUFFER_ALLOC
#define	GSS_KRB5_S_G_BAD_MSG_CTX
#define	GSS_KRB5_S_G_WRONG_SIZE
#define	GSS_KRB5_S_G_BAD_USAGE
#define	GSS_KRB5_S_G_UNKNOWN_QOP
#define	GSS_KRB5_S_KG_CCACHE_NOMATCH
#define	GSS_KRB5_S_KG_KEYTAB_NOMATCH
#define	GSS_KRB5_S_KG_TGT_MISSING
#define	GSS_KRB5_S_KG_NO_SUBKEY
#define	GSS_KRB5_S_KG_CONTEXT_ESTABLISHED
#define	GSS_KRB5_S_KG_BAD_SIGN_TYPE
#define	GSS_KRB5_S_KG_BAD_LENGTH

#define	GSS_KRB5_S_KG_CTX_INCOMPLETE
extern gss_OID	GSS_KRB5_NT_USER_NAME
extern gss_OID	GSS_KRB5_NT_HOSTBASED_SERVICE_NAME
extern gss_OID	GSS_KRB5_NT_PRINCIPAL_NAME
extern gss_OID	GSS_KRB5_NT_MACHINE_UID_NAME
extern gss_OID	GSS_KRB5_NT_STRING_UID_NAME

Description

Functions

Types and Values

GSS_KRB5_S_G_BAD_SERVICE_NAME

```
#define GSS_KRB5_S_G_BAD_SERVICE_NAME 1
```

GSS_KRB5_S_G_BAD_STRING_UID

```
#define GSS_KRB5_S_G_BAD_STRING_UID 2
```

GSS_KRB5_S_G_NOUSER

```
#define GSS_KRB5_S_G_NOUSER 3
```

GSS_KRB5_S_G_VALIDATE_FAILED

```
#define GSS_KRB5_S_G_VALIDATE_FAILED 4
```

GSS_KRB5_S_G_BUFFER_ALLOC

```
#define GSS_KRB5_S_G_BUFFER_ALLOC 5
```

GSS_KRB5_S_G_BAD_MSG_CTX

```
#define GSS_KRB5_S_G_BAD_MSG_CTX 6
```

GSS_KRB5_S_G_WRONG_SIZE

```
#define GSS_KRB5_S_G_WRONG_SIZE 7
```

GSS_KRB5_S_G_BAD_USAGE

```
#define GSS_KRB5_S_G_BAD_USAGE 8
```

GSS_KRB5_S_G_UNKNOWN_QOP

```
#define GSS_KRB5_S_G_UNKNOWN_QOP 9
```

GSS_KRB5_S_KG_CCACHE_NOMATCH

```
#define GSS_KRB5_S_KG_CCACHE_NOMATCH 10
```

GSS_KRB5_S_KG_KEYTAB_NOMATCH

```
#define GSS_KRB5_S_KG_KEYTAB_NOMATCH 11
```

GSS_KRB5_S_KG_TGT_MISSING

```
#define GSS_KRB5_S_KG_TGT_MISSING 12
```

GSS_KRB5_S_KG_NO_SUBKEY

```
#define GSS_KRB5_S_KG_NO_SUBKEY 13
```

GSS_KRB5_S_KG_CONTEXT_ESTABLISHED

```
#define GSS_KRB5_S_KG_CONTEXT_ESTABLISHED 14
```

GSS_KRB5_S_KG_BAD_SIGN_TYPE

```
#define GSS_KRB5_S_KG_BAD_SIGN_TYPE 15
```

GSS_KRB5_S_KG_BAD_LENGTH

```
#define GSS_KRB5_S_KG_BAD_LENGTH 16
```

GSS_KRB5_S_KG_CTX_INCOMPLETE

```
#define GSS_KRB5_S_KG_CTX_INCOMPLETE 17
```

GSS_KRB5_NT_USER_NAME

```
extern gss_OID GSS_KRB5_NT_USER_NAME;
```

GSS_KRB5_NT_HOSTBASED_SERVICE_NAME

```
extern gss_OID GSS_KRB5_NT_HOSTBASED_SERVICE_NAME;
```

GSS_KRB5_NT_PRINCIPAL_NAME

```
extern gss_OID GSS_KRB5_NT_PRINCIPAL_NAME;
```

GSS_KRB5_NT_MACHINE_UID_NAME

```
extern gss_OID GSS_KRB5_NT_MACHINE_UID_NAME;
```

GSS_KRB5_NT_STRING_UID_NAME

```
extern gss_OID GSS_KRB5_NT_STRING_UID_NAME;
```

1.5 krb5-ext

krb5-ext —

Types and Values

extern gss_OID	GSS_KRB5
extern gss_OID_desc	GSS_KRB5_static
extern gss_OID_desc	GSS_KRB5_NT_USER_NAME_static
extern gss_OID_desc	GSS_KRB5_NT_HOSTBASED_SERVICE_NAME_static
extern gss_OID_desc	GSS_KRB5_NT_PRINCIPAL_NAME_static
extern gss_OID_desc	GSS_KRB5_NT_MACHINE_UID_NAME_static
extern gss_OID_desc	GSS_KRB5_NT_STRING_UID_NAME_static

Description

Functions

Types and Values

GSS_KRB5

```
extern gss_OID GSS_KRB5;
```

GSS_KRB5_static

```
extern gss_OID_desc GSS_KRB5_static;
```

GSS_KRB5_NT_USER_NAME_static

```
extern gss_OID_desc GSS_KRB5_NT_USER_NAME_static;
```

GSS_KRB5_NT_HOSTBASED_SERVICE_NAME_static

```
extern gss_OID_desc GSS_KRB5_NT_HOSTBASED_SERVICE_NAME_static;
```

GSS_KRB5_NT_PRINCIPAL_NAME_static

```
extern gss_OID_desc GSS_KRB5_NT_PRINCIPAL_NAME_static;
```

GSS_KRB5_NT_MACHINE_UID_NAME_static

```
extern gss_OID_desc GSS_KRB5_NT_MACHINE_UID_NAME_static;
```

GSS_KRB5_NT_STRING_UID_NAME_static

```
extern gss_OID_desc GSS_KRB5_NT_STRING_UID_NAME_static;
```

Chapter 2

Index

O