```
Network Working Group                                          J. Linn
Request for Comments: 2078                     OpenVision Technologies
Category: Standards Track                                 January 1997
Obsoletes: 1508
```

Generic Security Service Application Program Interface, Version 2

Status of this Memo

Abstract

The Generic Security Service Application Program Interface (GSS-API), as defined in RFC-1508, provides security services to callers in a generic fashion, supportable with a range of underlying mechanisms and technologies and hence allowing source-level portability of applications to different environments. This specification defines GSS-API services and primitives at a level independent of underlying mechanism and programming language environment, and is to be complemented by other, related specifications:

documents defining specific parameter bindings for particular language environments

documents defining token formats, protocols, and procedures to be implemented in order to realize GSS-API services atop particular security mechanisms

This memo revises RFC-1508, making specific, incremental changes in response to implementation experience and liaison requests. It is intended, therefore, that this memo or a successor version thereto will become the basis for subsequent progression of the GSS-API specification on the standards track.

Table of Contents

1: GSS-API Characteristics and Concepts

GSS-API operates in the following paradigm. A typical GSS-API caller is itself a communications protocol, calling on GSS-API in order to protect its communications with authentication, integrity, and/or confidentiality security services. A GSS-API caller accepts tokens provided to it by its local GSS-API implementation and transfers the tokens to a peer on a remote system; that peer passes the received tokens to its local GSS-API implementation for processing. The security services available through GSS-API in this fashion are implementable (and have been implemented) over a range of underlying mechanisms based on secret-key and public-key cryptographic technologies.

The GSS-API separates the operations of initializing a security context between peers, achieving peer entity authentication (This security service definition, and other definitions used in this document, corresponds to that provided in International Standard ISO 7498-2-1988(E), Security Architecture.) (GSS_Init_sec_context() and GSS_Accept_sec_context() calls), from the operations of providing per-message data origin authentication and data integrity protection (GSS_GetMIC() and GSS_VerifyMIC() calls) for messages subsequently transferred in conjunction with that context. When establishing a security context, the GSS-API enables a context initiator to optionally permit its credentials to be delegated, meaning that the context acceptor may initiate further security contexts on behalf of the initiating caller. Per-message GSS_Wrap() and GSS_Unwrap() calls provide the data origin authentication and data integrity services which GSS_GetMIC() and GSS_VerifyMIC() offer, and also support selection of confidentiality services as a caller option. Additional calls provide supportive functions to the GSS-API's users.

The following paragraphs provide an example illustrating the dataflows involved in use of the GSS-API by a client and server in a mechanism-independent fashion, establishing a security context and transferring a protected message. The example assumes that credential acquisition has already been completed. The example assumes that the underlying authentication technology is capable of authenticating a client to a server using elements carried within a single token, and of authenticating the server to the client (mutual authentication) with a single returned token; this assumption holds for presently-documented CAT mechanisms but is not necessarily true for other cryptographic technologies and associated protocols.

The client calls GSS_Init_sec_context() to establish a security context to the server identified by targ_name, and elects to set the mutual_req_flag so that mutual authentication is performed in the course of context establishment. GSS_Init_sec_context() returns an output_token to be passed to the server, and indicates GSS_S_CONTINUE_NEEDED status pending completion of the mutual authentication sequence. Had mutual_req_flag not been set, the initial call to GSS_Init_sec_context() would have returned GSS_S_COMPLETE status. The client sends the output_token to the server.

The server passes the received token as the input_token parameter to GSS_Accept_sec_context(). GSS_Accept_sec_context indicates GSS_S_COMPLETE status, provides the client's authenticated identity in the src_name result, and provides an output_token to be passed to the client. The server sends the output_token to the client.

The client passes the received token as the input_token parameter to a successor call to GSS_Init_sec_context(), which processes data included in the token in order to achieve mutual authentication from the client's viewpoint. This call to GSS_Init_sec_context() returns GSS_S_COMPLETE status, indicating successful mutual authentication and the completion of context establishment for this example.

The client generates a data message and passes it to GSS_Wrap(). GSS_Wrap() performs data origin authentication, data integrity, and (optionally) confidentiality processing on the message and encapsulates the result into output_message, indicating GSS_S_COMPLETE status. The client sends the output_message to the server.

The server passes the received message to GSS_Unwrap(). GSS_Unwrap() inverts the encapsulation performed by GSS_Wrap(), deciphers the message if the optional confidentiality feature was applied, and validates the data origin authentication and data integrity checking quantities. GSS_Unwrap() indicates successful validation by returning GSS_S_COMPLETE status along with the resultant output_message.

For purposes of this example, we assume that the server knows by out-of-band means that this context will have no further use after one protected message is transferred from client to server. Given this premise, the server now calls GSS_Delete_sec_context() to flush context-level information. Optionally, the server-side application may provide a token buffer to GSS_Delete_sec_context(), to receive a context_token to be transferred to the client in order to request that client-side context-level information be deleted.

If a context_token is transferred, the client passes the context_token to GSS_Process_context_token(), which returns GSS_S_COMPLETE status after deleting context-level information at the client system.

The GSS-API design assumes and addresses several basic goals, including:

Mechanism independence: The GSS-API defines an interface to cryptographically implemented strong authentication and other security services at a generic level which is independent of particular underlying mechanisms. For example, GSS-API-provided services can be implemented by secret-key technologies (e.g., Kerberos) or public-key approaches (e.g., X.509).

Protocol environment independence: The GSS-API is independent of the communications protocol suites with which it is employed, permitting use in a broad range of protocol environments. In appropriate environments, an intermediate implementation "veneer" which is oriented to a particular communication protocol (e.g., Remote Procedure Call (RPC)) may be interposed between applications which call that protocol and the GSS-API, thereby invoking GSS-API facilities in conjunction with that protocol’s communications invocations.

Protocol association independence: The GSS-API’s security context construct is independent of communications protocol association constructs. This characteristic allows a single GSS-API implementation to be utilized by a variety of invoking protocol modules on behalf of those modules' calling applications. GSS-API services can also be invoked directly by applications, wholly independent of protocol associations.

Suitability to a range of implementation placements: GSS-API clients are not constrained to reside within any Trusted Computing Base (TCB) perimeter defined on a system where the GSS-API is implemented; security services are specified in a manner suitable to both intra-TCB and extra-TCB callers.

1.1: GSS-API Constructs

This section describes the basic elements comprising the GSS-API.

1.1.1: Credentials

1.1.1.1: Credential Constructs and Concepts

Credentials provide the prerequisites which permit GSS-API peers to establish security contexts with each other. A caller may designate that the credential elements which are to be applied for context initiation or acceptance be selected by default. Alternately, those GSS-API callers which need to make explicit selection of particular credentials structures may make references to those credentials through GSS-API-provided credential handles ("cred_handles"). In all cases, callers' credential references are indirect, mediated by GSS-API implementations and not requiring callers to access the selected credential elements.

A single credential structure may be used to initiate outbound contexts and to accept inbound contexts. Callers needing to operate in only one of these modes may designate this fact when credentials are acquired for use, allowing underlying mechanisms to optimize their processing and storage requirements. The credential elements defined by a particular mechanism may contain multiple cryptographic keys, e.g., to enable authentication and message encryption to be performed with different algorithms.

A GSS-API credential structure may contain multiple credential elements, each containing mechanism-specific information for a particular underlying mechanism (mech_type), but the set of elements within a given credential structure represent a common entity. A credential structure's contents will vary depending on the set of mech_types supported by a particular GSS-API implementation. Each credential element identifies the data needed by its mechanism in order to establish contexts on behalf of a particular principal, and may contain separate credential references for use in context initiation and context acceptance. Multiple credential elements within a given credential having overlapping combinations of mechanism, usage mode, and validity period are not permitted.

Commonly, a single mech_type will be used for all security contexts established by a particular initiator to a particular target. A major motivation for supporting credential sets representing multiple mech_types is to allow initiators on systems which are equipped to handle multiple types to initiate contexts to targets on other systems which can accommodate only a subset of the set supported at the initiator’s system.

1.1.1.2: Credential Management

It is the responsibility of underlying system-specific mechanisms and OS functions below the GSS-API to ensure that the ability to acquire and use credentials associated with a given identity is constrained to appropriate processes within a system. This responsibility should be taken seriously by implementors, as the ability for an entity to utilize a principal’s credentials is equivalent to the entity’s ability to successfully assert that principal’s identity.

Once a set of GSS-API credentials is established, the transferability of that credentials set to other processes or analogous constructs within a system is a local matter, not defined by the GSS-API. An example local policy would be one in which any credentials received as a result of login to a given user account, or of delegation of rights to that account, are accessible by, or transferable to, processes running under that account.

The credential establishment process (particularly when performed on behalf of users rather than server processes) is likely to require access to passwords or other quantities which should be protected locally and exposed for the shortest time possible. As a result, it will often be appropriate for preliminary credential establishment to be performed through local means at user login time, with the result(s) cached for subsequent reference. These preliminary credentials would be set aside (in a system-specific fashion) for subsequent use, either:

> to be accessed by an invocation of the GSS-API GSS_Acquire_cred() call, returning an explicit handle to reference that credential
>
> to comprise default credential elements to be installed, and to be used when default credential behavior is requested on behalf of a process

1.1.1.3: Default Credential Resolution

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

GSS_Init_sec_context:

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

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

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

(iv) A user-configurable default identity should be used.

GSS_Accept_sec_context:

(i) If there is only a single authorized principal identity capable of accepting security contexts, then that principal shall be used, otherwise

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

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

(iv) A user-configurable default identity shall be used.

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

1.1.2: Tokens

Tokens are data elements transferred between GSS-API callers, and are divided into two classes. Context-level tokens are exchanged in order to establish and manage a security context between peers. Per-message tokens relate to an established context and are exchanged to provide protective security services (i.e., data origin authentication, integrity, and optional confidentiality) for corresponding data messages.

The first context-level token obtained from GSS_Init_sec_context() is required to indicate at its very beginning a globally-interpretable mechanism identifier, i.e., an Object Identifier (OID) of the security mechanism. The remaining part of this token as well as the whole content of all other tokens are specific to the particular underlying mechanism used to support the GSS-API. Section 3 of this document provides, for designers of GSS-API support mechanisms, the description of the header of the first context-level token which is then followed by mechanism-specific information.

Tokens’ contents are opaque from the viewpoint of GSS-API callers. They are generated within the GSS-API implementation at an end system, provided to a GSS-API caller to be transferred to the peer GSS-API caller at a remote end system, and processed by the GSS-API implementation at that remote end system. Tokens may be output by GSS-API calls (and should be transferred to GSS-API peers) whether or not the calls’ status indicators indicate successful completion. Token transfer may take place in an in-band manner, integrated into the same protocol stream used by the GSS-API callers for other data transfers, or in an out-of-band manner across a logically separate channel.

Different GSS-API tokens are used for different purposes (e.g., context initiation, context acceptance, protected message data on an established context), and it is the responsibility of a GSS-API caller receiving tokens to distinguish their types, associate them with corresponding security contexts, and pass them to appropriate GSS-API processing routines. Depending on the caller protocol environment, this distinction may be accomplished in several ways.

The following examples illustrate means through which tokens’ types may be distinguished:

- implicit tagging based on state information (e.g., all tokens on a new association are considered to be context establishment tokens until context establishment is completed, at which point all tokens are considered to be wrapped data objects for that context),

- explicit tagging at the caller protocol level,

- a hybrid of these approaches.

Commonly, the encapsulated data within a token includes internal mechanism-specific tagging information, enabling mechanism-level processing modules to distinguish tokens used within the mechanism for different purposes. Such internal mechanism-level tagging is recommended to mechanism designers, and enables mechanisms to determine whether a caller has passed a particular token for processing by an inappropriate GSS-API routine.

Development of GSS-API support primitives based on a particular underlying cryptographic technique and protocol (i.e., conformant to a specific GSS-API mechanism definition) does not necessarily imply that GSS-API callers using that GSS-API mechanism will be able to interoperate with peers invoking the same technique and protocol outside the GSS-API paradigm, or with peers implementing a different GSS-API mechanism based on the same underlying technology. The format of GSS-API tokens defined in conjunction with a particular mechanism, and the techniques used to integrate those tokens into callers’ protocols, may not be interoperable with the tokens used by non-GSS-API callers of the same underlying technique.

1.1.3: Security Contexts

Security contexts are established between peers, using credentials established locally in conjunction with each peer or received by peers via delegation. Multiple contexts may exist simultaneously between a pair of peers, using the same or different sets of credentials. Coexistence of multiple contexts using different credentials allows graceful rollover when credentials expire. Distinction among multiple contexts based on the same credentials serves applications by distinguishing different message streams in a security sense.

The GSS-API is independent of underlying protocols and addressing structure, and depends on its callers to transport GSS-API-provided data elements. As a result of these factors, it is a caller responsibility to parse communicated messages, separating GSS-API-related data elements from caller-provided data. The GSS-API is independent of connection vs. connectionless orientation of the underlying communications service.

No correlation between security context and communications protocol association is dictated. (The optional channel binding facility, discussed in Section 1.1.6 of this document, represents an intentional exception to this rule, supporting additional protection features within GSS-API supporting mechanisms.) This separation allows the GSS-API to be used in a wide range of communications environments, and also simplifies the calling sequences of the individual calls. In many cases (depending on underlying security protocol, associated mechanism, and availability of cached information), the state information required for context setup can be sent concurrently with initial signed user data, without interposing additional message exchanges.

1.1.4: Mechanism Types

In order to successfully establish a security context with a target peer, it is necessary to identify an appropriate underlying mechanism type (mech_type) which both initiator and target peers support. The definition of a mechanism embodies not only the use of a particular cryptographic technology (or a hybrid or choice among alternative cryptographic technologies), but also definition of the syntax and semantics of data element exchanges which that mechanism will employ in order to support security services.

It is recommended that callers initiating contexts specify the "default" mech_type value, allowing system-specific functions within or invoked by the GSS-API implementation to select the appropriate mech_type, but callers may direct that a particular mech_type be employed when necessary.

The means for identifying a shared mech_type to establish a security context with a peer will vary in different environments and circumstances; examples include (but are not limited to):

use of a fixed mech_type, defined by configuration, within an environment

syntactic convention on a target-specific basis, through examination of a target's name

lookup of a target's name in a naming service or other database in order to identify mech_types supported by that target explicit negotiation between GSS-API callers in advance of security context setup

When transferred between GSS-API peers, mech_type specifiers (per Section 3, represented as Object Identifiers (OIDs)) serve to qualify the interpretation of associated tokens. (The structure and encoding of Object Identifiers is defined in ISO/IEC 8824, "Specification of Abstract Syntax Notation One (ASN.1)" and in ISO/IEC 8825, "Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)".) Use of hierarchically structured OIDs serves to preclude ambiguous interpretation of mech_type specifiers. The OID representing the DASS MechType, for example, is 1.3.12.2.1011.7.5, and that of the Kerberos V5 mechanism, once advanced to the level of Proposed Standard, will be 1.2.840.113554.1.2.2.

1.1.5: Naming

The GSS-API avoids prescribing naming structures, treating the names which are transferred across the interface in order to initiate and accept security contexts as opaque objects. This approach supports the GSS-API's goal of implementability atop a range of underlying security mechanisms, recognizing the fact that different mechanisms process and authenticate names which are presented in different forms. Generalized services offering translation functions among arbitrary sets of naming environments are outside the scope of the GSS-API; availability and use of local conversion functions to translate among the naming formats supported within a given end system is anticipated.

Different classes of name representations are used in conjunction with different GSS-API parameters:

- Internal form (denoted in this document by INTERNAL NAME), opaque to callers and defined by individual GSS-API implementations. GSS-API implementations supporting multiple namespace types must maintain internal tags to disambiguate the interpretation of particular names. A Mechanism Name (MN) is a special case of INTERNAL NAME, guaranteed to contain elements corresponding to one and only one mechanism; calls which are guaranteed to emit MNs or which require MNs as input are so identified within this specification.

- Contiguous string ("flat") form (denoted in this document by OCTET STRING); accompanied by OID tags identifying the namespace to which they correspond. Depending on tag value, flat names may or may not be printable strings for direct acceptance from and presentation to users. Tagging of flat names allows GSS-API callers and underlying GSS-API mechanisms to disambiguate name types and to determine whether an associated name’s type is one which they are capable of processing, avoiding aliasing problems which could result from misinterpreting a name of one type as a name of another type.

- The GSS-API Exported Name Object, a special case of flat name designated by a reserved OID value, carries a canonicalized form of a name suitable for binary comparisons.

In addition to providing means for names to be tagged with types, this specification defines primitives to support a level of naming environment independence for certain calling applications. To provide basic services oriented towards the requirements of callers which need not themselves interpret the internal syntax and semantics of names, GSS-API calls for name comparison (GSS_Compare_name()), human-readable display (GSS_Display_name()), input conversion (GSS_Import_name()), internal name deallocation (GSS_Release_name()), and internal name duplication (GSS_Duplicate_name()) functions are defined. (It is anticipated that these proposed GSS-API calls will be implemented in many end systems based on system-specific name manipulation primitives already extant within those end systems; inclusion within the GSS-API is intended to offer GSS-API callers a portable means to perform specific operations, supportive of authorization and audit requirements, on authenticated names.)

GSS_Import_name() implementations can, where appropriate, support more than one printable syntax corresponding to a given namespace (e.g., alternative printable representations for X.500 Distinguished Names), allowing flexibility for their callers to select among alternative representations. GSS_Display_name() implementations output a printable syntax selected as appropriate to their operational environments; this selection is a local matter. Callers desiring portability across alternative printable syntaxes should refrain from implementing comparisons based on printable name forms and should instead use the GSS_Compare_name() call to determine whether or not one internal-format name matches another.

The GSS_Canonicalize_name() and GSS_Export_name() calls enable callers to acquire and process Exported Name Objects, canonicalized and translated in accordance with the procedures of a particular GSS-API mechanism. Exported Name Objects can, in turn, be input to GSS_Import_name(), yielding equivalent MNs. These facilities are designed specifically to enable efficient storage and comparison of names (e.g., for use in access control lists).

The following diagram illustrates the intended dataflow among name-related GSS-API processing routines.

```
                        GSS-API library defaults
                               |
                               |
                               V                         text, for
   text -------------->  internal_name (IN) -----------> display only
         import_name()          /          display_name()
                               /
                              /
                             /
    accept_sec_context()    /
          |                /
          |               /
          |              /  canonicalize_name()
          |             /
          |            /
          |           /
          |          /
          |         /
          |        |
          V        V     <---------------------
    single mechanism        import_name()         exported name: flat
    internal_name (MN)                            binary "blob" usable
                         ---------------------->  for access control
                            export_name()
```

1.1.6: Channel Bindings

The GSS-API accommodates the concept of caller-provided channel binding ("chan_binding") information. Channel bindings are used to strengthen the quality with which peer entity authentication is provided during context establishment, by limiting the scope within which an intercepted context establishment token can be reused by an attacker. Specifically, they enable GSS-API callers to bind the establishment of a security context to relevant characteristics (e.g., addresses, transformed representations of encryption keys) of the underlying communications channel, of protection mechanisms applied to that communications channel, and to application-specific data.

The caller initiating a security context must determine the appropriate channel binding values to provide as input to the GSS_Init_sec_context() call, and consistent values must be provided to GSS_Accept_sec_context() by the context’s target, in order for both peers’ GSS-API mechanisms to validate that received tokens possess correct channel-related characteristics. Use or non-use of the GSS-API channel binding facility is a caller option. GSS-API mechanisms can operate in an environment where NULL channel bindings are presented; mechanism implementors are encouraged, but not required, to make use of caller-provided channel binding data within their mechanisms. Callers should not assume that underlying mechanisms provide confidentiality protection for channel binding information.

When non-NULL channel bindings are provided by callers, certain mechanisms can offer enhanced security value by interpreting the bindings' content (rather than simply representing those bindings, or integrity check values computed on them, within tokens) and will therefore depend on presentation of specific data in a defined format. To this end, agreements among mechanism implementors are defining conventional interpretations for the contents of channel binding arguments, including address specifiers (with content dependent on communications protocol environment) for context initiators and acceptors. (These conventions are being incorporated in GSS-API mechanism specifications and into the GSS-API C language bindings specification.) In order for GSS-API callers to be portable across multiple mechanisms and achieve the full security functionality which each mechanism can provide, it is strongly recommended that GSS-API callers provide channel bindings consistent with these conventions and those of the networking environment in which they operate.

1.2: GSS-API Features and Issues

This section describes aspects of GSS-API operations, of the security services which the GSS-API provides, and provides commentary on design issues.

1.2.1: Status Reporting

Each GSS-API call provides two status return values. Major_status values provide a mechanism-independent indication of call status (e.g., GSS_S_COMPLETE, GSS_S_FAILURE, GSS_S_CONTINUE_NEEDED), sufficient to drive normal control flow within the caller in a generic fashion. Table 1 summarizes the defined major_status return codes in tabular fashion.

Table 1: GSS-API Major Status Codes

FATAL ERROR CODES

GSS_S_BAD_BINDINGS	channel binding mismatch
GSS_S_BAD_MECH	unsupported mechanism requested
GSS_S_BAD_NAME	invalid name provided
GSS_S_BAD_NAMETYPE	name of unsupported type provided
GSS_S_BAD_STATUS	invalid input status selector
GSS_S_BAD_SIG	token had invalid integrity check
GSS_S_CONTEXT_EXPIRED	specified security context expired
GSS_S_CREDENTIALS_EXPIRED	expired credentials detected
GSS_S_DEFECTIVE_CREDENTIAL	defective credential detected
GSS_S_DEFECTIVE_TOKEN	defective token detected
GSS_S_FAILURE	failure, unspecified at GSS-API level
GSS_S_NO_CONTEXT	no valid security context specified
GSS_S_NO_CRED	no valid credentials provided
GSS_S_BAD_QOP	unsupported QOP value
GSS_S_UNAUTHORIZED	operation unauthorized
GSS_S_UNAVAILABLE	operation unavailable
GSS_S_DUPLICATE_ELEMENT	duplicate credential element requested
GSS_S_NAME_NOT_MN	name contains multi-mechanism elements

INFORMATORY STATUS CODES

GSS_S_COMPLETE	normal completion
GSS_S_CONTINUE_NEEDED	continuation call to routine required
GSS_S_DUPLICATE_TOKEN	duplicate per-message token detected
GSS_S_OLD_TOKEN	timed-out per-message token detected
GSS_S_UNSEQ_TOKEN	reordered (early) per-message token detected
GSS_S_GAP_TOKEN	skipped predecessor token(s) detected

Minor_status provides more detailed status information which may include status codes specific to the underlying security mechanism. Minor_status values are not specified in this document.

GSS_S_CONTINUE_NEEDED major_status returns, and optional message outputs, are provided in GSS_Init_sec_context() and GSS_Accept_sec_context() calls so that different mechanisms' employment of different numbers of messages within their authentication sequences need not be reflected in separate code paths within calling applications. Instead, such cases are accommodated with sequences of continuation calls to GSS_Init_sec_context() and GSS_Accept_sec_context(). The same mechanism is used to encapsulate mutual authentication within the GSS-API's context initiation calls.

For mech_types which require interactions with third-party servers in order to establish a security context, GSS-API context establishment calls may block pending completion of such third-party interactions.

On the other hand, no GSS-API calls pend on serialized interactions with GSS-API peer entities. As a result, local GSS-API status returns cannot reflect unpredictable or asynchronous exceptions occurring at remote peers, and reflection of such status information is a caller responsibility outside the GSS-API.

1.2.2: Per-Message Security Service Availability

When a context is established, two flags are returned to indicate the set of per-message protection security services which will be available on the context:

the integ_avail flag indicates whether per-message integrity and data origin authentication services are available

the conf_avail flag indicates whether per-message confidentiality services are available, and will never be returned TRUE unless the integ_avail flag is also returned TRUE

GSS-API callers desiring per-message security services should check the values of these flags at context establishment time, and must be aware that a returned FALSE value for integ_avail means that invocation of GSS_GetMIC() or GSS_Wrap() primitives on the associated context will apply no cryptographic protection to user data messages.

The GSS-API per-message integrity and data origin authentication services provide assurance to a receiving caller that protection was applied to a message by the caller's peer on the security context, corresponding to the entity named at context initiation. The GSS-API per-message confidentiality service provides assurance to a sending caller that the message's content is protected from access by entities other than the context's named peer.

The GSS-API per-message protection service primitives, as the category name implies, are oriented to operation at the granularity of protocol data units. They perform cryptographic operations on the data units, transfer cryptographic control information in tokens, and, in the case of GSS_Wrap(), encapsulate the protected data unit. As such, these primitives are not oriented to efficient data protection for stream-paradigm protocols (e.g., Telnet) if cryptography must be applied on an octet-by-octet basis.

1.2.3: Per-Message Replay Detection and Sequencing

Certain underlying mech_types offer support for replay detection and/or sequencing of messages transferred on the contexts they support. These optionally-selectable protection features are distinct from replay detection and sequencing features applied to the context establishment operation itself; the presence or absence of context-level replay or sequencing features is wholly a function of the underlying mech_type's capabilities, and is not selected or omitted as a caller option.

The caller initiating a context provides flags (replay_det_req_flag and sequence_req_flag) to specify whether the use of per-message replay detection and sequencing features is desired on the context being established. The GSS-API implementation at the initiator system can determine whether these features are supported (and whether they are optionally selectable) as a function of mech_type, without need for bilateral negotiation with the target. When enabled, these features provide recipients with indicators as a result of GSS-API processing of incoming messages, identifying whether those messages were detected as duplicates or out-of-sequence. Detection of such events does not prevent a suspect message from being provided to a recipient; the appropriate course of action on a suspect message is a matter of caller policy.

The semantics of the replay detection and sequencing services applied to received messages, as visible across the interface which the GSS-API provides to its clients, are as follows:

When replay_det_state is TRUE, the possible major_status returns for well-formed and correctly signed messages are as follows:

1. GSS_S_COMPLETE indicates that the message was within the window (of time or sequence space) allowing replay events to be detected, and that the message was not a replay of a previously-processed message within that window.

2. GSS_S_DUPLICATE_TOKEN indicates that the cryptographic checkvalue on the received message was correct, but that the message was recognized as a duplicate of a previously-processed message.

3. GSS_S_OLD_TOKEN indicates that the cryptographic checkvalue on the received message was correct, but that the message is too old to be checked for duplication.

When sequence_state is TRUE, the possible major_status returns for well-formed and correctly signed messages are as follows:

1. GSS_S_COMPLETE indicates that the message was within the window (of time or sequence space) allowing replay events to be detected, that the message was not a replay of a previously-processed message within that window, and that no predecessor sequenced messages are missing relative to the last received message (if any) processed on the context with a correct cryptographic checkvalue.

2. GSS_S_DUPLICATE_TOKEN indicates that the integrity check value on the received message was correct, but that the message was recognized as a duplicate of a previously-processed message.

3. GSS_S_OLD_TOKEN indicates that the integrity check value on the received message was correct, but that the token is too old to be checked for duplication.

4. GSS_S_UNSEQ_TOKEN indicates that the cryptographic checkvalue on the received message was correct, but that it is earlier in a sequenced stream than a message already processed on the context. [Note: Mechanisms can be architected to provide a stricter form of sequencing service, delivering particular messages to recipients only after all predecessor messages in an ordered stream have been delivered. This type of support is incompatible with the GSS-API paradigm in which recipients receive all messages, whether in order or not, and provide them (one at a time, without intra-GSS-API message buffering) to GSS-API routines for validation. GSS-API facilities provide supportive functions, aiding clients to achieve strict message stream integrity in an efficient manner in conjunction with sequencing provisions in communications protocols, but the GSS-API does not offer this level of message stream integrity service by itself.]

5. GSS_S_GAP_TOKEN indicates that the cryptographic checkvalue on the received message was correct, but that one or more predecessor sequenced messages have not been successfully processed relative to the last received message (if any) processed on the context with a correct cryptographic checkvalue.

As the message stream integrity features (especially sequencing) may interfere with certain applications' intended communications paradigms, and since support for such features is likely to be resource intensive, it is highly recommended that mech_types supporting these features allow them to be activated selectively on initiator request when a context is established. A context initiator and target are provided with corresponding indicators (replay_det_state and sequence_state), signifying whether these features are active on a given context.

An example mech_type supporting per-message replay detection could (when replay_det_state is TRUE) implement the feature as follows: The underlying mechanism would insert timestamps in data elements output by GSS_GetMIC() and GSS_Wrap(), and would maintain (within a time-limited window) a cache (qualified by originator-recipient pair) identifying received data elements processed by GSS_VerifyMIC() and GSS_Unwrap(). When this feature is active, exception status returns (GSS_S_DUPLICATE_TOKEN, GSS_S_OLD_TOKEN) will be provided when GSS_VerifyMIC() or GSS_Unwrap() is presented with a message which is either a detected duplicate of a prior message or which is too old to validate against a cache of recently received messages.

1.2.4: Quality of Protection

Some mech_types provide their users with fine granularity control over the means used to provide per-message protection, allowing callers to trade off security processing overhead dynamically against the protection requirements of particular messages. A per-message quality-of-protection parameter (analogous to quality-of-service, or QOS) selects among different QOP options supported by that mechanism. On context establishment for a multi-QOP mech_type, context-level data provides the prerequisite data for a range of protection qualities.

It is expected that the majority of callers will not wish to exert explicit mechanism-specific QOP control and will therefore request selection of a default QOP. Definitions of, and choices among, non-default QOP values are mechanism-specific, and no ordered sequences of QOP values can be assumed equivalent across different mechanisms. Meaningful use of non-default QOP values demands that callers be familiar with the QOP definitions of an underlying mechanism or mechanisms, and is therefore a non-portable construct. The

GSS_S_BAD_QOP major_status value is defined in order to indicate that a provided QOP value is unsupported for a security context, most likely because that value is unrecognized by the underlying mechanism.

1.2.5: Anonymity Support

In certain situations or environments, an application may wish to authenticate a peer and/or protect communications using GSS-API per-message services without revealing its own identity. For example, consider an application which provides read access to a research database, and which permits queries by arbitrary requestors. A client of such a service might wish to authenticate the service, to establish trust in the information received from it, but might not wish to disclose its identity to the service for privacy reasons.

In ordinary GSS-API usage, a context initiator's identity is made available to the context acceptor as part of the context establishment process. To provide for anonymity support, a facility (input anon_req_flag to GSS_Init_sec_context()) is provided through which context initiators may request that their identity not be provided to the context acceptor. Mechanisms are not required to honor this request, but a caller will be informed (via returned anon_state indicator from GSS_Init_sec_context()) whether or not the request is honored. Note that authentication as the anonymous principal does not necessarily imply that credentials are not required in order to establish a context.

The following Object Identifier value is provided as a means to identify anonymous names, and can be compared against in order to determine, in a mechanism-independent fashion, whether a name refers to an anonymous principal:

```
{1(iso), 3(org), 6(dod), 1(internet), 5(security), 6(nametypes),
3(gss-anonymous-name)}
```

The recommended symbolic name corresponding to this definition is GSS_C_NT_ANONYMOUS.

Four possible combinations of anon_state and mutual_state are possible, with the following results:

anon_state == FALSE, mutual_state == FALSE: initiator authenticated to target.

anon_state == FALSE, mutual_state == TRUE: initiator authenticated to target, target authenticated to initiator.

anon_state == TRUE, mutual_state == FALSE: initiator authenticated as anonymous principal to target.

anon_state == TRUE, mutual_state == TRUE: initiator authenticated as anonymous principal to target, target authenticated to initiator.

1.2.6: Initialization

No initialization calls (i.e., calls which must be invoked prior to invocation of other facilities in the interface) are defined in GSS-API. As an implication of this fact, GSS-API implementations must themselves be self-initializing.

1.2.7: Per-Message Protection During Context Establishment

A facility is defined in GSS-V2 to enable protection and buffering of data messages for later transfer while a security context's establishment is in GSS_S_CONTINUE_NEEDED status, to be used in cases where the caller side already possesses the necessary session key to enable this processing. Specifically, a new state Boolean, called prot_ready_state, is added to the set of information returned by GSS_Init_sec_context(), GSS_Accept_sec_context(), and GSS_Inquire_context().

For context establishment calls, this state Boolean is valid and interpretable when the associated major_status is either GSS_S_CONTINUE_NEEDED, or GSS_S_COMPLETE. Callers of GSS-API (both initiators and acceptors) can assume that per-message protection (via GSS_Wrap(), GSS_Unwrap(), GSS_GetMIC() and GSS_VerifyMIC()) is available and ready for use if either: prot_ready_state == TRUE, or major_status == GSS_S_COMPLETE, though mutual authentication (if requested) cannot be guaranteed until GSS_S_COMPLETE is returned.

This achieves full, transparent backward compatibility for GSS-API V1 callers, who need not even know of the existence of prot_ready_state, and who will get the expected behavior from GSS_S_COMPLETE, but who will not be able to use per-message protection before GSS_S_COMPLETE is returned.

It is not a requirement that GSS-V2 mechanisms ever return TRUE prot_ready_state before completion of context establishment (indeed, some mechanisms will not evolve usable message protection keys, especially at the context acceptor, before context establishment is complete). It is expected but not required that GSS-V2 mechanisms will return TRUE prot_ready_state upon completion of context establishment if they support per-message protection at all (however GSS-V2 applications should not assume that TRUE prot_ready_state will always be returned together with the GSS_S_COMPLETE major_status, since GSS-V2 implementations may continue to support GSS-V1 mechanism code, which will never return TRUE prot_ready_state).

When prot_ready_state is returned TRUE, mechanisms shall also set those context service indicator flags (deleg_state, mutual_state, replay_det_state, sequence_state, anon_state, trans_state, conf_avail, integ_avail) which represent facilities confirmed, at that time, to be available on the context being established. In situations where prot_ready_state is returned before GSS_S_COMPLETE, it is possible that additional facilities may be confirmed and subsequently indicated when GSS_S_COMPLETE is returned.

1.2.8: Implementation Robustness

This section recommends aspects of GSS-API implementation behavior in the interests of overall robustness.

If a token is presented for processing on a GSS-API security context and that token is determined to be invalid for that context, the context's state should not be disrupted for purposes of processing subsequent valid tokens.

Certain local conditions at a GSS-API implementation (e.g., unavailability of memory) may preclude, temporarily or permanently, the successful processing of tokens on a GSS-API security context, typically generating GSS_S_FAILURE major_status returns along with locally-significant minor_status. For robust operation under such conditions, the following recommendations are made:

Failing calls should free any memory they allocate, so that callers may retry without causing further loss of resources.

Failure of an individual call on an established context should not preclude subsequent calls from succeeding on the same context.

Whenever possible, it should be possible for GSS_Delete_sec_context() calls to be successfully processed even if other calls cannot succeed, thereby enabling context-related resources to be released.

2: Interface Descriptions

This section describes the GSS-API's service interface, dividing the set of calls offered into four groups. Credential management calls are related to the acquisition and release of credentials by principals. Context-level calls are related to the management of security contexts between principals. Per-message calls are related to the protection of individual messages on established security contexts. Support calls provide ancillary functions useful to GSS-API callers. Table 2 groups and summarizes the calls in tabular fashion.

Table 2: GSS-API Calls

CREDENTIAL MANAGEMENT	
GSS_Acquire_cred	acquire credentials for use
GSS_Release_cred	release credentials after use
GSS_Inquire_cred	display information about credentials
GSS_Add_cred	construct credentials incrementally
GSS_Inquire_cred_by_mech	display per-mechanism credential information
CONTEXT-LEVEL CALLS	
GSS_Init_sec_context	initiate outbound security context
GSS_Accept_sec_context	accept inbound security context
GSS_Delete_sec_context	flush context when no longer needed
GSS_Process_context_token	process received control token on context
GSS_Context_time	indicate validity time remaining on context
GSS_Inquire_context	display information about context
GSS_Wrap_size_limit	determine GSS_Wrap token size limit
GSS_Export_sec_context	transfer context to other process
GSS_Import_sec_context	import transferred context
PER-MESSAGE CALLS	
GSS_GetMIC	apply integrity check, receive as token separate from message
GSS_VerifyMIC	validate integrity check token along with message
GSS_Wrap	sign, optionally encrypt, encapsulate
GSS_Unwrap	decapsulate, decrypt if needed, validate integrity check

SUPPORT CALLS

GSS_Display_status	translate status codes to printable form
GSS_Indicate_mechs	indicate mech_types supported on local system
GSS_Compare_name	compare two names for equality
GSS_Display_name	translate name to printable form
GSS_Import_name	convert printable name to normalized form
GSS_Release_name	free storage of normalized-form name
GSS_Release_buffer	free storage of printable name
GSS_Release_OID	free storage of OID object
GSS_Release_OID_set	free storage of OID set object
GSS_Create_empty_OID_set	create empty OID set
GSS_Add_OID_set_member	add member to OID set
GSS_Test_OID_set_member	test if OID is member of OID set
GSS_OID_to_str	display OID as string
GSS_Str_to_OID	construct OID from string
GSS_Inquire_names_for_mech	indicate name types supported by mechanism
GSS_Inquire_mechs_for_name	indicates mechanisms supporting name type
GSS_Canonicalize_name	translate name to per-mechanism form
GSS_Export_name	externalize per-mechanism name
GSS_Duplicate_name	duplicate name object

2.1: Credential management calls

These GSS-API calls provide functions related to the management of credentials. Their characterization with regard to whether or not they may block pending exchanges with other network entities (e.g., directories or authentication servers) depends in part on OS-specific (extra-GSS-API) issues, so is not specified in this document.

The GSS_Acquire_cred() call is defined within the GSS-API in support of application portability, with a particular orientation towards support of portable server applications. It is recognized that (for certain systems and mechanisms) credentials for interactive users may be managed differently from credentials for server processes; in such environments, it is the GSS-API implementation's responsibility to distinguish these cases and the procedures for making this distinction are a local matter. The GSS_Release_cred() call provides a means for callers to indicate to the GSS-API that use of a credentials structure is no longer required. The GSS_Inquire_cred() call allows callers to determine information about a credentials structure. The GSS_Add_cred() call enables callers to append elements to an existing credential structure, allowing iterative construction of a multi-mechanism credential. The GSS_Inquire_cred_by_mech() call enables callers to extract per-mechanism information describing a credentials structure.

2.1.1: GSS_Acquire_cred call

Inputs:

- o desired_name INTERNAL NAME, -NULL requests locally-determined default
- o lifetime_req INTEGER,-in seconds; 0 requests default
- o desired_mechs SET OF OBJECT IDENTIFIER,-empty set requests system-selected default
- o cred_usage INTEGER -0=INITIATE-AND-ACCEPT, 1=INITIATE-ONLY, 2=ACCEPT-ONLY

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o output_cred_handle CREDENTIAL HANDLE,
- o actual_mechs SET OF OBJECT IDENTIFIER,
- o lifetime_rec INTEGER -in seconds, or reserved value for INDEFINITE

Return major_status codes:

- o GSS_S_COMPLETE indicates that requested credentials were successfully established, for the duration indicated in lifetime_rec, suitable for the usage requested in cred_usage, for the set of mech_types indicated in actual_mechs, and that those credentials can be referenced for subsequent use with the handle returned in output_cred_handle.
- o GSS_S_BAD_MECH indicates that a mech_type unsupported by the GSS-API implementation type was requested, causing the credential establishment operation to fail.

- o GSS_S_BAD_NAMETYPE indicates that the provided desired_name is uninterpretable or of a type unsupported by the applicable underlying GSS-API mechanism(s), so no credentials could be established for the accompanying desired_name.

- o GSS_S_BAD_NAME indicates that the provided desired_name is inconsistent in terms of internally-incorporated type specifier information, so no credentials could be established for the accompanying desired_name.

- o GSS_S_FAILURE indicates that credential establishment failed for reasons unspecified at the GSS-API level, including lack of authorization to establish and use credentials associated with the identity named in the input desired_name argument.

GSS_Acquire_cred() is used to acquire credentials so that a principal can (as a function of the input cred_usage parameter) initiate and/or accept security contexts under the identity represented by the desired_name input argument. On successful completion, the returned output_cred_handle result provides a handle for subsequent references to the acquired credentials. Typically, single-user client processes requesting that default credential behavior be applied for context establishment purposes will have no need to invoke this call.

A caller may provide the value NULL for desired_name, signifying a request for credentials corresponding to a principal identity selected by default for the caller. The procedures used by GSS-API implementations to select the appropriate principal identity in response to such a request are local matters. It is possible that multiple pre-established credentials may exist for the same principal identity (for example, as a result of multiple user login sessions) when GSS_Acquire_cred() is called; the means used in such cases to select a specific credential are local matters. The input lifetime_req argument to GSS_Acquire_cred() may provide useful information for local GSS-API implementations to employ in making this disambiguation in a manner which will best satisfy a caller’s intent.

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

The caller of GSS_Acquire_cred() can explicitly specify a set of mech_types which are to be accommodated in the returned credentials (desired_mechs argument), or can request credentials for a system-defined default set of mech_types. Selection of the system-specified default set is recommended in the interests of application portability. The actual_mechs return value may be interrogated by the caller to determine the set of mechanisms with which the returned credentials may be used.

2.1.2: GSS_Release_cred call

Input:

- o cred_handle CREDENTIAL HANDLE - NULL specifies that the credential elements used when default credential behavior is requested be released.

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER

Return major_status codes:

- o GSS_S_COMPLETE indicates that the credentials referenced by the input cred_handle were released for purposes of subsequent access by the caller. The effect on other processes which may be authorized shared access to such credentials is a local matter.
- o GSS_S_NO_CRED indicates that no release operation was performed, either because the input cred_handle was invalid or because the caller lacks authorization to access the referenced credentials.
- o GSS_S_FAILURE indicates that the release operation failed for reasons unspecified at the GSS-API level.

Provides a means for a caller to explicitly request that credentials be released when their use is no longer required. Note that system-specific credential management functions are also likely to exist, for example to assure that credentials shared among processes are properly deleted when all affected processes terminate, even if no explicit release requests are issued by those processes. Given the fact that multiple callers are not precluded from gaining authorized access to the same credentials, invocation of GSS_Release_cred() cannot be assumed to delete a particular set of credentials on a system-wide basis.

2.1.3: GSS_Inquire_cred call

Input:

- o cred_handle CREDENTIAL HANDLE -NULL specifies that the credential elements used when default credential behavior is requested are to be queried

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o cred_name INTERNAL NAME,
- o lifetime_rec INTEGER -in seconds, or reserved value for INDEFINITE
- o cred_usage INTEGER, -0=INITIATE-AND-ACCEPT, 1=INITIATE-ONLY, 2=ACCEPT-ONLY
- o mech_set SET OF OBJECT IDENTIFIER

Return major_status codes:

- o GSS_S_COMPLETE indicates that the credentials referenced by the input cred_handle argument were valid, and that the output cred_name, lifetime_rec, and cred_usage values represent, respectively, the credentials' associated principal name, remaining lifetime, suitable usage modes, and supported mechanism types.
- o GSS_S_NO_CRED indicates that no information could be returned about the referenced credentials, either because the input cred_handle was invalid or because the caller lacks authorization to access the referenced credentials.

- o GSS_S_DEFECTIVE_CREDENTIAL indicates that the referenced credentials are invalid.

- o GSS_S_CREDENTIALS_EXPIRED indicates that the referenced credentials have expired.

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

The GSS_Inquire_cred() call is defined primarily for the use of those callers which request use of default credential behavior rather than acquiring credentials explicitly with GSS_Acquire_cred(). It enables callers to determine a credential structure's associated principal name, remaining validity period, usability for security context initiation and/or acceptance, and supported mechanisms.

For a multi-mechanism credential, the returned "lifetime" specifier indicates the shortest lifetime of any of the mechanisms' elements in the credential (for either context initiation or acceptance purposes).

GSS_Inquire_cred() should indicate INITIATE-AND-ACCEPT for "cred_usage" if both of the following conditions hold:

(1) there exists in the credential an element which allows context initiation using some mechanism

(2) there exists in the credential an element which allows context acceptance using some mechanism (allowably, but not necessarily, one of the same mechanism(s) qualifying for (1)).

If condition (1) holds but not condition (2), GSS_Inquire_cred() should indicate INITIATE-ONLY for "cred_usage". If condition (2) holds but not condition (1), GSS_Inquire_cred() should indicate ACCEPT-ONLY for "cred_usage".

Callers requiring finer disambiguation among available combinations of lifetimes, usage modes, and mechanisms should call the GSS_Inquire_cred_by_mech() routine, passing that routine one of the mech OIDs returned by GSS_Inquire_cred().

2.1.4: GSS_Add_cred call

Inputs:

- o input_cred_handle CREDENTIAL HANDLE - handle to credential structure created with prior GSS_Acquire_cred() or GSS_Add_cred() call, or NULL to append elements to the set which are applied for the caller when default credential behavior is specified.

- o desired_name INTERNAL NAME - NULL requests locally-determined default

- o initiator_time_req INTEGER - in seconds; 0 requests default

- o acceptor_time_req INTEGER - in seconds; 0 requests default

- o desired_mech OBJECT IDENTIFIER

- o cred_usage INTEGER - 0=INITIATE-AND-ACCEPT, 1=INITIATE-ONLY, 2=ACCEPT-ONLY

Outputs:

- o major_status INTEGER,

- o minor_status INTEGER,

- o output_cred_handle CREDENTIAL HANDLE, - NULL to request that credential elements be added "in place" to the credential structure identified by input_cred_handle, non-NULL pointer to request that a new credential structure and handle be created.

- o actual_mechs SET OF OBJECT IDENTIFIER,

- o initiator_time_rec INTEGER - in seconds, or reserved value for INDEFINITE

- o acceptor_time_rec INTEGER - in seconds, or reserved value for INDEFINITE

- o cred_usage INTEGER, -0=INITIATE-AND-ACCEPT, 1=INITIATE-ONLY, 2=ACCEPT-ONLY

- o mech_set SET OF OBJECT IDENTIFIER -- full set of mechanisms supported by resulting credential.

Return major_status codes:

- o GSS_S_COMPLETE indicates that the credentials referenced by the input_cred_handle argument were valid, and that the resulting credential from GSS_Add_cred() is valid for the durations indicated in initiator_time_rec and acceptor_time_rec, suitable for the usage requested in cred_usage, and for the mechanisms indicated in actual_mechs.

- o GSS_S_DUPLICATE_ELEMENT indicates that the input desired_mech specified a mechanism for which the referenced credential already contained a credential element with overlapping cred_usage and validity time specifiers.

- o GSS_S_BAD_MECH indicates that the input desired_mech specified a mechanism unsupported by the GSS-API implementation, causing the GSS_Add_cred() operation to fail.

- o GSS_S_BAD_NAMETYPE indicates that the provided desired_name is uninterpretable or of a type unsupported by the applicable underlying GSS-API mechanism(s), so the GSS_Add_cred() operation could not be performed for that name.

- o GSS_S_BAD_NAME indicates that the provided desired_name is inconsistent in terms of internally-incorporated type specifier information, so the GSS_Add_cred() operation could not be performed for that name.

- o GSS_S_NO_CRED indicates that the input_cred_handle referenced invalid or inaccessible credentials.

- o GSS_S_FAILURE indicates that the operation failed for reasons unspecified at the GSS-API level, including lack of authorization to establish or use credentials representing the requested identity.

GSS_Add_cred() enables callers to construct credentials iteratively by adding credential elements in successive operations, corresponding to different mechanisms. This offers particular value in multi-mechanism environments, as the major_status and minor_status values returned on each iteration are individually visible and can therefore be interpreted unambiguously on a per-mechanism basis.

The same input desired_name, or default reference, should be used on all GSS_Acquire_cred() and GSS_Add_cred() calls corresponding to a particular credential.

2.1.5: GSS_Inquire_cred_by_mech call

Inputs:

- o cred_handle CREDENTIAL HANDLE -- NULL specifies that the credential elements used when default credential behavior is requested are to be queried

- o mech_type OBJECT IDENTIFIER -- specific mechanism for which credentials are being queried

Outputs:

- o major_status INTEGER,

- o minor_status INTEGER,

- o cred_name INTERNAL NAME, -- guaranteed to be MN

- o lifetime_rec_initiate INTEGER -- in seconds, or reserved value for INDEFINITE

- o lifetime_rec_accept INTEGER -- in seconds, or reserved value for INDEFINITE

- o cred_usage INTEGER, -0=INITIATE-AND-ACCEPT, 1=INITIATE-ONLY, 2=ACCEPT-ONLY

Return major_status codes:

- o GSS_S_COMPLETE indicates that the credentials referenced by the input cred_handle argument were valid, that the mechanism indicated by the input mech_type was represented with elements within those credentials, and that the output cred_name, lifetime_rec_initiate, lifetime_rec_accept, and cred_usage values represent, respectively, the credentials' associated principal name, remaining lifetimes, and suitable usage modes.

- o GSS_S_NO_CRED indicates that no information could be returned about the referenced credentials, either because the input cred_handle was invalid or because the caller lacks authorization to access the referenced credentials.

- o GSS_S_DEFECTIVE_CREDENTIAL indicates that the referenced credentials are invalid.

- o GSS_S_CREDENTIALS_EXPIRED indicates that the referenced credentials have expired.

- o GSS_S_BAD_MECH indicates that the referenced credentials do not contain elements for the requested mechanism.
- o GSS_S_FAILURE indicates that the operation failed for reasons unspecified at the GSS-API level.

The GSS_Inquire_cred_by_mech() call enables callers in multi-mechanism environments to acquire specific data about available combinations of lifetimes, usage modes, and mechanisms within a credential structure. The lifetime_rec_initiate result indicates the available lifetime for context initiation purposes; the lifetime_rec_accept result indicates the available lifetime for context acceptance purposes.

2.2: Context-level calls

This group of calls is devoted to the establishment and management of security contexts between peers. A context's initiator calls GSS_Init_sec_context(), resulting in generation of a token which the caller passes to the target. At the target, that token is passed to GSS_Accept_sec_context(). Depending on the underlying mech_type and specified options, additional token exchanges may be performed in the course of context establishment; such exchanges are accommodated by GSS_S_CONTINUE_NEEDED status returns from GSS_Init_sec_context() and GSS_Accept_sec_context().

Either party to an established context may invoke GSS_Delete_sec_context() to flush context information when a context is no longer required. GSS_Process_context_token() is used to process received tokens carrying context-level control information. GSS_Context_time() allows a caller to determine the length of time for which an established context will remain valid. GSS_Inquire_context() returns status information describing context characteristics. GSS_Wrap_size_limit() allows a caller to determine the size of a token which will be generated by a GSS_Wrap() operation. GSS_Export_sec_context() and GSS_Import_sec_context() enable transfer of active contexts between processes on an end system.

2.2.1: GSS_Init_sec_context call

Inputs:

- o claimant_cred_handle CREDENTIAL HANDLE, -NULL specifies "use default"
- o input_context_handle CONTEXT HANDLE, -0 specifies "none assigned yet"

- targ_name INTERNAL NAME,
- mech_type OBJECT IDENTIFIER, -NULL parameter specifies "use default"
- deleg_req_flag BOOLEAN,
- mutual_req_flag BOOLEAN,
- replay_det_req_flag BOOLEAN,
- sequence_req_flag BOOLEAN,
- anon_req_flag BOOLEAN,
- lifetime_req INTEGER,-0 specifies default lifetime
- chan_bindings OCTET STRING,
- input_token OCTET STRING-NULL or token received from target

Outputs:

- major_status INTEGER,
- minor_status INTEGER,
- output_context_handle CONTEXT HANDLE,
- mech_type OBJECT IDENTIFIER, -actual mechanism always indicated, never NULL
- output_token OCTET STRING, -NULL or token to pass to context target
- deleg_state BOOLEAN,
- mutual_state BOOLEAN,
- replay_det_state BOOLEAN,
- sequence_state BOOLEAN,
- anon_state BOOLEAN,
- trans_state BOOLEAN,
- prot_ready_state BOOLEAN, -- see Section 1.2.7

- conf_avail BOOLEAN,
- integ_avail BOOLEAN,
- lifetime_rec INTEGER - in seconds, or reserved value for INDEFINITE

This call may block pending network interactions for those mech_types in which an authentication server or other network entity must be consulted on behalf of a context initiator in order to generate an output_token suitable for presentation to a specified target.

Return major_status codes:

- GSS_S_COMPLETE indicates that context-level information was successfully initialized, and that the returned output_token will provide sufficient information for the target to perform per-message processing on the newly-established context.
- GSS_S_CONTINUE_NEEDED indicates that control information in the returned output_token must be sent to the target, and that a reply must be received and passed as the input_token argument to a continuation call to GSS_Init_sec_context(), before per-message processing can be performed in conjunction with this context.
- GSS_S_DEFECTIVE_TOKEN indicates that consistency checks performed on the input_token failed, preventing further processing from being performed based on that token.
- GSS_S_DEFECTIVE_CREDENTIAL indicates that consistency checks performed on the credential structure referenced by claimant_cred_handle failed, preventing further processing from being performed using that credential structure.
- GSS_S_BAD_SIG indicates that the received input_token contains an incorrect integrity check, so context setup cannot be accomplished.
- GSS_S_NO_CRED indicates that no context was established, either because the input cred_handle was invalid, because the referenced credentials are valid for context acceptor use only, or because the caller lacks authorization to access the referenced credentials.
- GSS_S_CREDENTIALS_EXPIRED indicates that the credentials provided through the input claimant_cred_handle argument are no longer valid, so context establishment cannot be completed.

- o GSS_S_BAD_BINDINGS indicates that a mismatch between the caller-provided chan_bindings and those extracted from the input_token was detected, signifying a security-relevant event and preventing context establishment. (This result will be returned by GSS_Init_sec_context only for contexts where mutual_state is TRUE.)

- o GSS_S_OLD_TOKEN indicates that the input_token is too old to be checked for integrity. This is a fatal error during context establishment.

- o GSS_S_DUPLICATE_TOKEN indicates that the input token has a correct integrity check, but is a duplicate of a token already processed. This is a fatal error during context establishment.

- o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided; this major status will be returned only for successor calls following GSS_S_CONTINUE_ NEEDED status returns.

- o GSS_S_BAD_NAMETYPE indicates that the provided targ_name is of a type uninterpretable or unsupported by the applicable underlying GSS-API mechanism(s), so context establishment cannot be completed.

- o GSS_S_BAD_NAME indicates that the provided targ_name is inconsistent in terms of internally-incorporated type specifier information, so context establishment cannot be accomplished.

- o GSS_S_BAD_MECH indicates receipt of a context establishment token or of a caller request specifying a mechanism unsupported by the local system or with the caller’s active credentials

- o GSS_S_FAILURE indicates that context setup could not be accomplished for reasons unspecified at the GSS-API level, and that no interface-defined recovery action is available.

This routine is used by a context initiator, and ordinarily emits one (or, for the case of a multi-step exchange, more than one) output_token suitable for use by the target within the selected mech_type’s protocol. Using information in the credentials structure referenced by claimant_cred_handle, GSS_Init_sec_context() initializes the data structures required to establish a security context with target targ_name. The targ_name may be any valid INTERNAL NAME; it need not be an MN. The claimant_cred_handle must correspond to the same valid credentials structure on the initial call to GSS_Init_sec_context() and on any successor calls resulting from GSS_S_CONTINUE_NEEDED status returns; different protocol sequences modeled by the GSS_S_CONTINUE_NEEDED facility will require access to credentials at different points in the context establishment sequence.

The input_context_handle argument is 0, specifying "not yet assigned", on the first GSS_Init_sec_context() call relating to a given context. If successful (i.e., if accompanied by major_status GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED), and only if successful, the initial GSS_Init_sec_context() call returns a non-zero output_context_handle for use in future references to this context. Once a non-zero output_context_handle has been returned, GSS-API callers should call GSS_Delete_sec_context() to release context-related resources if errors occur in later phases of context establishment, or when an established context is no longer required.

When continuation attempts to GSS_Init_sec_context() are needed to perform context establishment, the previously-returned non-zero handle value is entered into the input_context_handle argument and will be echoed in the returned output_context_handle argument. On such continuation attempts (and only on continuation attempts) the input_token value is used, to provide the token returned from the context's target.

The chan_bindings argument is used by the caller to provide information binding the security context to security-related characteristics (e.g., addresses, cryptographic keys) of the underlying communications channel. See Section 1.1.6 of this document for more discussion of this argument's usage.

The input_token argument contains a message received from the target, and is significant only on a call to GSS_Init_sec_context() which follows a previous return indicating GSS_S_CONTINUE_NEEDED major_status.

It is the caller's responsibility to establish a communications path to the target, and to transmit any returned output_token (independent of the accompanying returned major_status value) to the target over that path. The output_token can, however, be transmitted along with the first application-provided input message to be processed by GSS_GetMIC() or GSS_Wrap() in conjunction with a successfully-established context.

The initiator may request various context-level functions through input flags: the deleg_req_flag requests delegation of access rights, the mutual_req_flag requests mutual authentication, the replay_det_req_flag requests that replay detection features be applied to messages transferred on the established context, and the sequence_req_flag requests that sequencing be enforced. (See Section

1.2.3 for more information on replay detection and sequencing features.) The anon_req_flag requests that the initiator's identity not be transferred within tokens to be sent to the acceptor.

Not all of the optionally-requestable features will be available in all underlying mech_types. The corresponding return state values deleg_state, mutual_state, replay_det_state, and sequence_state indicate, as a function of mech_type processing capabilities and initiator-provided input flags, the set of features which will be active on the context. The returned trans_state value indicates whether the context is transferable to other processes through use of GSS_Export_sec_context(). These state indicators' values are undefined unless either the routine's major_status indicates GSS_S_COMPLETE, or TRUE prot_ready_state is returned along with GSS_S_CONTINUE_NEEDED major_status; for the latter case, it is possible that additional features, not confirmed or indicated along with TRUE prot_ready_state, will be confirmed and indicated when GSS_S_COMPLETE is subsequently returned.

The returned anon_state and prot_ready_state values are significant for both GSS_S_COMPLETE and GSS_S_CONTINUE_NEEDED major_status returns from GSS_Init_sec_context(). When anon_state is returned TRUE, this indicates that neither the current token nor its predecessors delivers or has delivered the initiator's identity. Callers wishing to perform context establishment only if anonymity support is provided should transfer a returned token from GSS_Init_sec_context() to the peer only if it is accompanied by a TRUE anon_state indicator. When prot_ready_state is returned TRUE in conjunction with GSS_S_CONTINUE_NEEDED major_status, this indicates that per-message protection operations may be applied on the context: see Section 1.2.7 for further discussion of this facility.

Failure to provide the precise set of features requested by the caller does not cause context establishment to fail; it is the caller's prerogative to delete the context if the feature set provided is unsuitable for the caller's use.

The returned mech_type value indicates the specific mechanism employed on the context, is valid only along with major_status GSS_S_COMPLETE, and will never indicate the value for "default". Note that, for the case of certain mechanisms which themselves perform negotiation, the returned mech_type result may indicate selection of a mechanism identified by an OID different than that passed in the input mech_type argument.

The conf_avail return value indicates whether the context supports per-message confidentiality services, and so informs the caller whether or not a request for encryption through the conf_req_flag input to GSS_Wrap() can be honored. In similar fashion, the integ_avail return value indicates whether per-message integrity services are available (through either GSS_GetMIC() or GSS_Wrap()) on the established context. These state indicators' values are undefined unless either the routine's major_status indicates GSS_S_COMPLETE, or TRUE prot_ready_state is returned along with GSS_S_CONTINUE_NEEDED major_status.

The lifetime_req input specifies a desired upper bound for the lifetime of the context to be established, with a value of 0 used to request a default lifetime. The lifetime_rec return value indicates the length of time for which the context will be valid, expressed as an offset from the present; depending on mechanism capabilities, credential lifetimes, and local policy, it may not correspond to the value requested in lifetime_req. If no constraints on context lifetime are imposed, this may be indicated by returning a reserved value representing INDEFINITE lifetime_req. The value of lifetime_rec is undefined unless the routine's major_status indicates GSS_S_COMPLETE.

If the mutual_state is TRUE, this fact will be reflected within the output_token. A call to GSS_Accept_sec_context() at the target in conjunction with such a context will return a token, to be processed by a continuation call to GSS_Init_sec_context(), in order to achieve mutual authentication.

2.2.2: GSS_Accept_sec_context call

Inputs:

- o acceptor_cred_handle CREDENTIAL HANDLE, -- NULL specifies "use default"
- o input_context_handle CONTEXT HANDLE, -- 0 specifies "not yet assigned"
- o chan_bindings OCTET STRING,
- o input_token OCTET STRING

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o src_name INTERNAL NAME, -- guaranteed to be MN

- o mech_type OBJECT IDENTIFIER,
- o output_context_handle CONTEXT HANDLE,
- o deleg_state BOOLEAN,
- o mutual_state BOOLEAN,
- o replay_det_state BOOLEAN,
- o sequence_state BOOLEAN,
- o anon_state BOOLEAN,
- o trans_state BOOLEAN,
- o prot_ready_state BOOLEAN, -- see Section 1.2.7 for discussion
- o conf_avail BOOLEAN,
- o integ_avail BOOLEAN,
- o lifetime_rec INTEGER, - in seconds, or reserved value for INDEFINITE
- o delegated_cred_handle CREDENTIAL HANDLE,
- o output_token OCTET STRING -NULL or token to pass to context initiator

This call may block pending network interactions for those mech_types in which a directory service or other network entity must be consulted on behalf of a context acceptor in order to validate a received input_token.

Return major_status codes:

- o GSS_S_COMPLETE indicates that context-level data structures were successfully initialized, and that per-message processing can now be performed in conjunction with this context.
- o GSS_S_CONTINUE_NEEDED indicates that control information in the returned output_token must be sent to the initiator, and that a response must be received and passed as the input_token argument to a continuation call to GSS_Accept_sec_context(), before per-message processing can be performed in conjunction with this context.

- GSS_S_DEFECTIVE_TOKEN indicates that consistency checks performed on the input_token failed, preventing further processing from being performed based on that token.

- GSS_S_DEFECTIVE_CREDENTIAL indicates that consistency checks performed on the credential structure referenced by acceptor_cred_handle failed, preventing further processing from being performed using that credential structure.

- GSS_S_BAD_SIG indicates that the received input_token contains an incorrect integrity check, so context setup cannot be accomplished.

- GSS_S_DUPLICATE_TOKEN indicates that the integrity check on the received input_token was correct, but that the input_token was recognized as a duplicate of an input_token already processed. No new context is established.

- GSS_S_OLD_TOKEN indicates that the integrity check on the received input_token was correct, but that the input_token is too old to be checked for duplication against previously-processed input_tokens. No new context is established.

- GSS_S_NO_CRED indicates that no context was established, either because the input cred_handle was invalid, because the referenced credentials are valid for context initiator use only, or because the caller lacks authorization to access the referenced credentials.

- GSS_S_CREDENTIALS_EXPIRED indicates that the credentials provided through the input acceptor_cred_handle argument are no longer valid, so context establishment cannot be completed.

- GSS_S_BAD_BINDINGS indicates that a mismatch between the caller-provided chan_bindings and those extracted from the input_token was detected, signifying a security-relevant event and preventing context establishment.

- GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided; this major status will be returned only for successor calls following GSS_S_CONTINUE_NEEDED status returns.

- GSS_S_BAD_MECH indicates receipt of a context establishment token specifying a mechanism unsupported by the local system or with the caller’s active credentials.

o GSS_S_FAILURE indicates that context setup could not be accomplished for reasons unspecified at the GSS-API level, and that no interface-defined recovery action is available.

The GSS_Accept_sec_context() routine is used by a context target. Using information in the credentials structure referenced by the input acceptor_cred_handle, it verifies the incoming input_token and (following the successful completion of a context establishment sequence) returns the authenticated src_name and the mech_type used. The returned src_name is guaranteed to be an MN, processed by the mechanism under which the context was established. The acceptor_cred_handle must correspond to the same valid credentials structure on the initial call to GSS_Accept_sec_context() and on any successor calls resulting from GSS_S_CONTINUE_NEEDED status returns; different protocol sequences modeled by the GSS_S_CONTINUE_NEEDED mechanism will require access to credentials at different points in the context establishment sequence.

The input_context_handle argument is 0, specifying "not yet assigned", on the first GSS_Accept_sec_context() call relating to a given context. If successful (i.e., if accompanied by major_status GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED), and only if successful, the initial GSS_Accept_sec_context() call returns a non-zero output_context_handle for use in future references to this context. Once a non-zero output_context_handle has been returned, GSS-API callers should call GSS_Delete_sec_context() to release context-related resources if errors occur in later phases of context establishment, or when an established context is no longer required.

The chan_bindings argument is used by the caller to provide information binding the security context to security-related characteristics (e.g., addresses, cryptographic keys) of the underlying communications channel. See Section 1.1.6 of this document for more discussion of this argument’s usage.

The returned state results (deleg_state, mutual_state, replay_det_state, sequence_state, anon_state, trans_state, and prot_ready_state) reflect the same information as described for GSS_Init_sec_context(), and their values are significant under the same return state conditions.

The conf_avail return value indicates whether the context supports per-message confidentiality services, and so informs the caller whether or not a request for encryption through the conf_req_flag input to GSS_Wrap() can be honored. In similar fashion, the integ_avail return value indicates whether per-message integrity services are available (through either GSS_GetMIC() or GSS_Wrap()) on the established context. These values are significant under the same return state conditions as described under GSS_Init_sec_context().

The lifetime_rec return value is significant only in conjunction with GSS_S_COMPLETE major_status, and indicates the length of time for which the context will be valid, expressed as an offset from the present.

The mech_type return value indicates the specific mechanism employed on the context, is valid only along with major_status GSS_S_COMPLETE, and will never indicate the value for "default".

The delegated_cred_handle result is significant only when deleg_state is TRUE, and provides a means for the target to reference the delegated credentials. The output_token result, when non-NULL, provides a context-level token to be returned to the context initiator to continue a multi-step context establishment sequence. As noted with GSS_Init_sec_context(), any returned token should be transferred to the context's peer (in this case, the context initiator), independent of the value of the accompanying returned major_status.

Note: A target must be able to distinguish a context-level input_token, which is passed to GSS_Accept_sec_context(), from the per-message data elements passed to GSS_VerifyMIC() or GSS_Unwrap(). These data elements may arrive in a single application message, and GSS_Accept_sec_context() must be performed before per-message processing can be performed successfully.

2.2.3: GSS_Delete_sec_context call

Input:

o context_handle CONTEXT HANDLE

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o output_context_token OCTET STRING

Return major_status codes:

o GSS_S_COMPLETE indicates that the context was recognized, and that relevant context-specific information was flushed. If the caller provides a non-null buffer to receive an output_context_token, and the mechanism returns a non-NULL token into that buffer, the returned output_context_token is ready for transfer to the context's peer.

o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided, so no deletion was performed.

o GSS_S_FAILURE indicates that the context is recognized, but that the GSS_Delete_sec_context() operation could not be performed for reasons unspecified at the GSS-API level.

This call may block pending network interactions for mech_types in which active notification must be made to a central server when a security context is to be deleted.

This call can be made by either peer in a security context, to flush context-specific information. If a non-null output_context_token parameter is provided by the caller, an output_context_token may be returned to the caller. If an output_context_token is provided to the caller, it can be passed to the context's peer to inform the peer's GSS-API implementation that the peer's corresponding context information can also be flushed. (Once a context is established, the peers involved are expected to retain cached credential and context-related information until the information's expiration time is reached or until a GSS_Delete_sec_context() call is made.)

The facility for context_token usage to signal context deletion is retained for compatibility with GSS-API Version 1. For current usage, it is recommended that both peers to a context invoke GSS_Delete_sec_context() independently, passing a null output_context_token buffer to indicate that no context_token is required. Implementations of GSS_Delete_sec_context() should delete relevant locally-stored context information.

Attempts to perform per-message processing on a deleted context will result in error returns.

2.2.4: GSS_Process_context_token call

Inputs:

- o context_handle CONTEXT HANDLE,
- o input_context_token OCTET STRING

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,

Return major_status codes:

- o GSS_S_COMPLETE indicates that the input_context_token was successfully processed in conjunction with the context referenced by context_handle.
- o GSS_S_DEFECTIVE_TOKEN indicates that consistency checks performed on the received context_token failed, preventing further processing from being performed with that token.
- o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided.
- o GSS_S_FAILURE indicates that the context is recognized, but that the GSS_Process_context_token() operation could not be performed for reasons unspecified at the GSS-API level.

This call is used to process context_tokens received from a peer once a context has been established, with corresponding impact on context-level state information. One use for this facility is processing of the context_tokens generated by GSS_Delete_sec_context(); GSS_Process_context_token() will not block pending network interactions for that purpose. Another use is to process tokens indicating remote-peer context establishment failures after the point where the local GSS-API implementation has already indicated GSS_S_COMPLETE status.

2.2.5: GSS_Context_time call

Input:

- o context_handle CONTEXT HANDLE,

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o lifetime_rec INTEGER - in seconds, or reserved value for INDEFINITE

Return major_status codes:

- o GSS_S_COMPLETE indicates that the referenced context is valid, and will remain valid for the amount of time indicated in lifetime_rec.
- o GSS_S_CONTEXT_EXPIRED indicates that data items related to the referenced context have expired.
- o GSS_S_CREDENTIALS_EXPIRED indicates that the context is recognized, but that its associated credentials have expired.
- o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided.
- o GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level.

This call is used to determine the amount of time for which a currently established context will remain valid.

2.2.6: GSS_Inquire_context call

Input:

- o context_handle CONTEXT HANDLE,

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,

```
   o  src_name INTERNAL NAME,  -- name of context initiator,
                               -- guaranteed to be MN

   o  targ_name INTERNAL NAME,  -- name of context target,
                                -- guaranteed to be MN


   o  lifetime_rec INTEGER -- in seconds, or reserved value for
      INDEFINITE,

   o  mech_type OBJECT IDENTIFIER, -- the mechanism supporting this
      security context

   o  deleg_state BOOLEAN,

   o  mutual_state BOOLEAN,

   o  replay_det_state BOOLEAN,

   o  sequence_state BOOLEAN,

   o  anon_state BOOLEAN,

   o  trans_state BOOLEAN,

   o  prot_ready_state BOOLEAN,

   o  conf_avail BOOLEAN,

   o  integ_avail BOOLEAN,

   o  locally_initiated BOOLEAN, -- TRUE if initiator, FALSE if acceptor
```

Return major_status codes:

- o GSS_S_COMPLETE indicates that the referenced context is valid and that src_name, targ_name, lifetime_rec, mech_type, deleg_state, mutual_state, replay_det_state, sequence_state, anon_state, trans_state, prot_ready_state, conf_avail, integ_avail, and locally_initiated return values describe the corresponding characteristics of the context.

- o GSS_S_CONTEXT_EXPIRED indicates that the provided input context_handle is recognized, but that the referenced context has expired. Return values other than major_status and minor_status are undefined.

- o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided. Return values other than major_status and minor_status are undefined.

- o GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level. Return values other than major_status and minor_status are undefined.

This call is used to extract information describing characteristics of a security context.

2.2.7: GSS_Wrap_size_limit call

Inputs:

- o context_handle CONTEXT HANDLE,

- o qop INTEGER,

- o output_size INTEGER

Outputs:

- o major_status INTEGER,

- o minor_status INTEGER,

- o max_input_size INTEGER

Return major_status codes:

- o GSS_S_COMPLETE indicates a successful token size determination: an input message with a length in octets equal to the returned max_input_size value will, when passed to GSS_Wrap() for processing on the context identified by the context_handle parameter and with the quality of protection specifier provided in the qop parameter, yield an output token no larger than the value of the provided output_size parameter.

- o GSS_S_CONTEXT_EXPIRED indicates that the provided input context_handle is recognized, but that the referenced context has expired. Return values other than major_status and minor_status are undefined.

- o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided. Return values other than major_status and minor_status are undefined.

o GSS_S_BAD_QOP indicates that the provided QOP value is not recognized or supported for the context.

o GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level. Return values other than major_status and minor_status are undefined.

This call is used to determine the largest input datum which may be passed to GSS_Wrap() without yielding an output token larger than a caller-specified value.

2.2.8:   GSS_Export_sec_context call

Inputs:

o context_handle CONTEXT HANDLE

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o interprocess_token OCTET STRING

Return major_status codes:

o GSS_S_COMPLETE indicates that the referenced context has been successfully exported to a representation in the interprocess_token, and is no longer available for use by the caller.

o GSS_S_UNAVAILABLE indicates that the context export facility is not available for use on the referenced context. (This status should occur only for contexts for which the trans_state value is FALSE.) Return values other than major_status and minor_status are undefined.

o GSS_S_CONTEXT_EXPIRED indicates that the provided input context_handle is recognized, but that the referenced context has expired. Return values other than major_status and minor_status are undefined.

o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided. Return values other than major_status and minor_status are undefined.

o GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level. Return values other than major_status and minor_status are undefined.

This call generates an interprocess token for transfer to another process within an end system, in order to transfer control of a security context to that process. The recipient of the interprocess token will call GSS_Import_sec_context() to accept the transfer. The GSS_Export_sec_context() operation is defined for use only with security contexts which are fully and successfully established (i.e., those for which GSS_Init_sec_context() and GSS_Accept_sec_context() have returned GSS_S_COMPLETE major_status).

To ensure portability, a caller of GSS_Export_sec_context() must not assume that a context may continue to be used once it has been exported; following export, the context referenced by the context_handle cannot be assumed to remain valid. Further, portable callers must not assume that a given interprocess token can be imported by GSS_Import_sec_context() more than once, thereby creating multiple instantiations of a single context. GSS-API implementations may detect and reject attempted multiple imports, but are not required to do so.

The internal representation contained within the interprocess token is an implementation-defined local matter. Interprocess tokens cannot be assumed to be transferable across different GSS-API implementations.

It is recommended that GSS-API implementations adopt policies suited to their operational environments in order to define the set of processes eligible to import a context, but specific constraints in this area are local matters. Candidate examples include transfers between processes operating on behalf of the same user identity, or processes comprising a common job. However, it may be impossible to enforce such policies in some implementations.

In support of the above goals, implementations may protect the transferred context data by using cryptography to protect data within the interprocess token, or by using interprocess tokens as a means to reference local interprocess communication facilities (protected by other means) rather than storing the context data directly within the tokens.

Transfer of an open context may, for certain mechanisms and implementations, reveal data about the credential which was used to establish the context. Callers should, therefore, be cautious about the trustworthiness of processes to which they transfer contexts. Although the GSS-API implementation may provide its own set of protections over the exported context, the caller is responsible for protecting the interprocess token from disclosure, and for taking care that the context is transferred to an appropriate destination process.

2.2.9: GSS_Import_sec_context call

Inputs:

o interprocess_token OCTET STRING

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o context_handle CONTEXT HANDLE

Return major_status codes:

o GSS_S_COMPLETE indicates that the context represented by the input interprocess_token has been successfully transferred to the caller, and is available for future use via the output context_handle.

o GSS_S_CONTEXT_EXPIRED indicates that the context represented by the input interprocess_token has expired. Return values other than major_status and minor_status are undefined.

o GSS_S_NO_CONTEXT indicates that the context represented by the input interprocess_token was invalid. Return values other than major_status and minor_status are undefined.

o GSS_S_DEFECTIVE_TOKEN indicates that the input interprocess_token was defective. Return values other than major_status and minor_status are undefined.

o GSS_S_UNAVAILABLE indicates that the context import facility is not available for use on the referenced context. Return values other than major_status and minor_status are undefined.

o GSS_S_UNAUTHORIZED indicates that the context represented by the input interprocess_token is unauthorized for transfer to the caller. Return values other than major_status and minor_status are undefined.

- GSS_S_FAILURE indicates that the requested operation failed for reasons unspecified at the GSS-API level. Return values other than major_status and minor_status are undefined.

This call processes an interprocess token generated by GSS_Export_sec_context(), making the transferred context available for use by the caller. After a successful GSS_Import_sec_context() operation, the imported context is available for use by the importing process.

For further discussion of the security and authorization issues regarding this call, please see the discussion in Section 2.2.8.

2.3: Per-message calls

This group of calls is used to perform per-message protection processing on an established security context. None of these calls block pending network interactions. These calls may be invoked by a context's initiator or by the context's target. The four members of this group should be considered as two pairs; the output from GSS_GetMIC() is properly input to GSS_VerifyMIC(), and the output from GSS_Wrap() is properly input to GSS_Unwrap().

GSS_GetMIC() and GSS_VerifyMIC() support data origin authentication and data integrity services. When GSS_GetMIC() is invoked on an input message, it yields a per-message token containing data items which allow underlying mechanisms to provide the specified security services. The original message, along with the generated per-message token, is passed to the remote peer; these two data elements are processed by GSS_VerifyMIC(), which validates the message in conjunction with the separate token.

GSS_Wrap() and GSS_Unwrap() support caller-requested confidentiality in addition to the data origin authentication and data integrity services offered by GSS_GetMIC() and GSS_VerifyMIC(). GSS_Wrap() outputs a single data element, encapsulating optionally enciphered user data as well as associated token data items. The data element output from GSS_Wrap() is passed to the remote peer and processed by GSS_Unwrap() at that system. GSS_Unwrap() combines decipherment (as required) with validation of data items related to authentication and integrity.

2.3.1: GSS_GetMIC call

Note: This call is functionally equivalent to the GSS_Sign call as defined in previous versions of this specification. In the interests of backward compatibility, it is recommended that implementations support this function under both names for the present; future references to this function as GSS_Sign are deprecated.

Inputs:

o context_handle CONTEXT HANDLE,

o qop_req INTEGER,-0 specifies default QOP

o message OCTET STRING

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o per_msg_token OCTET STRING

Return major_status codes:

o GSS_S_COMPLETE indicates that an integrity check, suitable for an established security context, was successfully applied and that the message and corresponding per_msg_token are ready for transmission.

o GSS_S_CONTEXT_EXPIRED indicates that context-related data items have expired, so that the requested operation cannot be performed.

o GSS_S_CREDENTIALS_EXPIRED indicates that the context is recognized, but that its associated credentials have expired, so that the requested operation cannot be performed.

o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided.

o GSS_S_BAD_QOP indicates that the provided QOP value is not recognized or supported for the context.

o GSS_S_FAILURE indicates that the context is recognized, but that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Using the security context referenced by context_handle, apply an integrity check to the input message (along with timestamps and/or other data included in support of mech_type-specific mechanisms) and return the result in per_msg_token. The qop_req parameter, interpretation of which is discussed in Section 1.2.4, allows quality-of-protection control. The caller passes the message and the per_msg_token to the target.

The GSS_GetMIC() function completes before the message and per_msg_token is sent to the peer; successful application of GSS_GetMIC() does not guarantee that a corresponding GSS_VerifyMIC() has been (or can necessarily be) performed successfully when the message arrives at the destination.

Mechanisms which do not support per-message protection services should return GSS_S_FAILURE if this routine is called.

2.3.2: GSS_VerifyMIC call

Note: This call is functionally equivalent to the GSS_Verify call as defined in previous versions of this specification. In the interests of backward compatibility, it is recommended that implementations support this function under both names for the present; future references to this function as GSS_Verify are deprecated.

Inputs:

o context_handle CONTEXT HANDLE,

o message OCTET STRING,

o per_msg_token OCTET STRING

Outputs:

o qop_state INTEGER,

o major_status INTEGER,

o minor_status INTEGER,

Return major_status codes:

o GSS_S_COMPLETE indicates that the message was successfully verified.

- o GSS_S_DEFECTIVE_TOKEN indicates that consistency checks performed on the received per_msg_token failed, preventing further processing from being performed with that token.

- o GSS_S_BAD_SIG indicates that the received per_msg_token contains an incorrect integrity check for the message.

- o GSS_S_DUPLICATE_TOKEN, GSS_S_OLD_TOKEN, GSS_S_UNSEQ_TOKEN, and GSS_S_GAP_TOKEN values appear in conjunction with the optional per-message replay detection features described in Section 1.2.3; their semantics are described in that section.

- o GSS_S_CONTEXT_EXPIRED indicates that context-related data items have expired, so that the requested operation cannot be performed.

- o GSS_S_CREDENTIALS_EXPIRED indicates that the context is recognized, but that its associated credentials have expired, so that the requested operation cannot be performed.

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

- o GSS_S_FAILURE indicates that the context is recognized, but that the GSS_VerifyMIC() operation could not be performed for reasons unspecified at the GSS-API level.

Using the security context referenced by context_handle, verify that the input per_msg_token contains an appropriate integrity check for the input message, and apply any active replay detection or sequencing features. Return an indication of the quality-of-protection applied to the processed message in the qop_state result. Since the GSS_VerifyMIC() routine never provides a confidentiality service, its implementations should not return non-zero values in the confidentiality fields of the output qop_state.

Mechanisms which do not support per-message protection services should return GSS_S_FAILURE if this routine is called.

2.3.3: GSS_Wrap call

Note: This call is functionally equivalent to the GSS_Seal call as defined in previous versions of this specification. In the interests of backward compatibility, it is recommended that implementations support this function under both names for the present; future references to this function as GSS_Seal are deprecated.

Inputs:

o context_handle CONTEXT HANDLE,

o conf_req_flag BOOLEAN,

o qop_req INTEGER,-0 specifies default QOP

o input_message OCTET STRING

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o conf_state BOOLEAN,

o output_message OCTET STRING

Return major_status codes:

o GSS_S_COMPLETE indicates that the input_message was successfully processed and that the output_message is ready for transmission.

o GSS_S_CONTEXT_EXPIRED indicates that context-related data items have expired, so that the requested operation cannot be performed.

o GSS_S_CREDENTIALS_EXPIRED indicates that the context is recognized,
but that its associated credentials have expired, so that the requested operation cannot be performed.

o GSS_S_NO_CONTEXT indicates that no valid context was recognized for the input context_handle provided.

o GSS_S_BAD_QOP indicates that the provided QOP value is not recognized or supported for the context.

o GSS_S_FAILURE indicates that the context is recognized, but that the GSS_Wrap() operation could not be performed for reasons unspecified at the GSS-API level.

Performs the data origin authentication and data integrity functions of GSS_GetMIC(). If the input conf_req_flag is TRUE, requests that confidentiality be applied to the input_message. Confidentiality may not be supported in all mech_types or by all implementations; the returned conf_state flag indicates whether confidentiality was provided for the input_message. The qop_req parameter, interpretation of which is discussed in Section 1.2.4, allows quality-of-protection control.

In all cases, the GSS_Wrap() call yields a single output_message data element containing (optionally enciphered) user data as well as control information.

Mechanisms which do not support per-message protection services should return GSS_S_FAILURE if this routine is called.

2.3.4: GSS_Unwrap call

Note: This call is functionally equivalent to the GSS_Unseal call as defined in previous versions of this specification. In the interests of backward compatibility, it is recommended that implementations support this function under both names for the present; future references to this function as GSS_Unseal are deprecated.

Inputs:

o context_handle CONTEXT HANDLE,

o input_message OCTET STRING

Outputs:

o conf_state BOOLEAN,

o qop_state INTEGER,

o major_status INTEGER,

o minor_status INTEGER,

o output_message OCTET STRING

Return major_status codes:

o GSS_S_COMPLETE indicates that the input_message was successfully processed and that the resulting output_message is available.

o GSS_S_DEFECTIVE_TOKEN indicates that consistency checks performed on the per_msg_token extracted from the input_message failed, preventing further processing from being performed.

- GSS_S_BAD_SIG indicates that an incorrect integrity check was detected for the message.

- GSS_S_DUPLICATE_TOKEN, GSS_S_OLD_TOKEN, GSS_S_UNSEQ_TOKEN, and GSS_S_GAP_TOKEN values appear in conjunction with the optional per-message replay detection features described in Section 1.2.3; their semantics are described in that section.

- GSS_S_CONTEXT_EXPIRED indicates that context-related data items have expired, so that the requested operation cannot be performed.

- GSS_S_CREDENTIALS_EXPIRED indicates that the context is recognized, but that its associated credentials have expired, so that the requested operation cannot be performed.

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

- GSS_S_FAILURE indicates that the context is recognized, but that the GSS_Unwrap() operation could not be performed for reasons unspecified at the GSS-API level.

Processes a data element generated (and optionally enciphered) by GSS_Wrap(), provided as input_message. The returned conf_state value indicates whether confidentiality was applied to the input_message. If conf_state is TRUE, GSS_Unwrap() deciphers the input_message. Returns an indication of the quality-of-protection applied to the processed message in the qop_state result. GSS_Wrap() performs the data integrity and data origin authentication checking functions of GSS_VerifyMIC() on the plaintext data. Plaintext data is returned in output_message.

Mechanisms which do not support per-message protection services should return GSS_S_FAILURE if this routine is called.

2.4: Support calls

This group of calls provides support functions useful to GSS-API callers, independent of the state of established contexts. Their characterization with regard to blocking or non-blocking status in terms of network interactions is unspecified.

2.4.1: GSS_Display_status call

Inputs:

- o status_value INTEGER,-GSS-API major_status or minor_status return value
- o status_type INTEGER,-1 if major_status, 2 if minor_status
- o mech_type OBJECT IDENTIFIER-mech_type to be used for minor_ status translation

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o status_string_set SET OF OCTET STRING

Return major_status codes:

- o GSS_S_COMPLETE indicates that a valid printable status representation (possibly representing more than one status event encoded within the status_value) is available in the returned status_string_set.
- o GSS_S_BAD_MECH indicates that translation in accordance with an unsupported mech_type was requested, so translation could not be performed.
- o GSS_S_BAD_STATUS indicates that the input status_value was invalid, or that the input status_type carried a value other than 1 or 2, so translation could not be performed.
- o GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Provides a means for callers to translate GSS-API-returned major and minor status codes into printable string representations.

2.4.2: GSS_Indicate_mechs call

Input:

- o (none)

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o mech_set SET OF OBJECT IDENTIFIER

Return major_status codes:

o GSS_S_COMPLETE indicates that a set of available mechanisms has been returned in mech_set.

o GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Allows callers to determine the set of mechanism types available on the local system. This call is intended for support of specialized callers who need to request non-default mech_type sets from GSS_Acquire_cred(), and should not be needed by other callers.

2.4.3: GSS_Compare_name call

Inputs:

o name1 INTERNAL NAME,

o name2 INTERNAL NAME

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o name_equal BOOLEAN

Return major_status codes:

o GSS_S_COMPLETE indicates that name1 and name2 were comparable, and that the name_equal result indicates whether name1 and name2 represent the same entity.

o GSS_S_BAD_NAMETYPE indicates that one or both of name1 and name2 contained internal type specifiers uninterpretable by the applicable underlying GSS-API mechanism(s), or that the two names' types are different and incomparable, so that the comparison operation could not be completed.

- o GSS_S_BAD_NAME indicates that one or both of the input names was ill-formed in terms of its internal type specifier, so the comparison operation could not be completed.

- o GSS_S_FAILURE indicates that the call's operation could not be performed for reasons unspecified at the GSS-API level.

Allows callers to compare two internal name representations to determine whether they refer to the same entity. If either name presented to GSS_Compare_name() denotes an anonymous principal, GSS_Compare_name() shall indicate FALSE. It is not required that either or both inputs name1 and name2 be MNs; for some implementations and cases, GSS_S_BAD_NAMETYPE may be returned, indicating name incomparability, for the case where neither input name is an MN.

2.4.4: GSS_Display_name call

Inputs:

- o name INTERNAL NAME

Outputs:

- o major_status INTEGER,

- o minor_status INTEGER,

- o name_string OCTET STRING,

- o name_type OBJECT IDENTIFIER

Return major_status codes:

- o GSS_S_COMPLETE indicates that a valid printable name representation is available in the returned name_string.

- o GSS_S_BAD_NAMETYPE indicates that the provided name was of a type uninterpretable by the applicable underlying GSS-API mechanism(s), so no printable representation could be generated.

- o GSS_S_BAD_NAME indicates that the contents of the provided name were inconsistent with the internally-indicated name type, so no printable representation could be generated.

- o GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Allows callers to translate an internal name representation into a printable form with associated namespace type descriptor. The syntax of the printable form is a local matter.

If the input name represents an anonymous identity, a reserved value (GSS_C_NT_ANONYMOUS) shall be returned for name_type.

2.4.5: GSS_Import_name call

Inputs:

- o input_name_string OCTET STRING,
- o input_name_type OBJECT IDENTIFIER

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o output_name INTERNAL NAME

Return major_status codes:

- o GSS_S_COMPLETE indicates that a valid name representation is output in output_name and described by the type value in output_name_type.
- o GSS_S_BAD_NAMETYPE indicates that the input_name_type is unsupported by the applicable underlying GSS-API mechanism(s), so the import operation could not be completed.
- o GSS_S_BAD_NAME indicates that the provided input_name_string is ill-formed in terms of the input_name_type, so the import operation could not be completed.
- o GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Allows callers to provide a name representation as a contiguous octet string, designate the type of namespace in conjunction with which it should be parsed, and convert that representation to an internal form suitable for input to other GSS-API routines. The syntax of the input_name_string is defined in conjunction with its associated name type; depending on the input_name_type, the associated input_name_string may or may not be a printable string. Note: The input_name_type argument serves to describe and qualify the interpretation of the associated input_name_string; it does not specify the data type of the returned output_name.

If a mechanism claims support for a particular name type, its GSS_Import_name() operation shall be able to accept all possible values conformant to the external name syntax as defined for that name type. These imported values may correspond to:

(1) locally registered entities (for which credentials may be acquired),

(2) non-local entities (for which local credentials cannot be acquired, but which may be referenced as targets of initiated security contexts or initiators of accepted security contexts), or to

(3) neither of the above.

Determination of whether a particular name belongs to class (1), (2), or (3) as described above is not guaranteed to be performed by the GSS_Import_name() function.

The internal name generated by a GSS_Import_name() operation may be a single-mechanism MN, and is likely to be an MN within a single-mechanism implementation, but portable callers must not depend on this property (and must not, therefore, assume that the output from GSS_Import_name() can be passed directly to GSS_Export_name() without first being processed through GSS_Canonicalize_name()).

2.4.6: GSS_Release_name call

Inputs:

o name INTERNAL NAME

Outputs:

o major_status INTEGER,

o minor_status INTEGER

Return major_status codes:

o GSS_S_COMPLETE indicates that the storage associated with the input name was successfully released.

o GSS_S_BAD_NAME indicates that the input name argument did not contain a valid name.

- o GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Allows callers to release the storage associated with an internal name representation. This call's specific behavior depends on the language and programming environment within which a GSS-API implementation operates, and is therefore detailed within applicable bindings specifications; in particular, this call may be superfluous within bindings where memory management is automatic.

2.4.7: GSS_Release_buffer call

Inputs:

- o buffer OCTET STRING

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER

Return major_status codes:

- o GSS_S_COMPLETE indicates that the storage associated with the input buffer was successfully released.
- o GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

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

2.4.8: GSS_Release_OID_set call

Inputs:

- o buffer SET OF OBJECT IDENTIFIER

Outputs:

- o major_status INTEGER,

o minor_status INTEGER

Return major_status codes:

o GSS_S_COMPLETE indicates that the storage associated with the input object identifier set was successfully released.

o GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Allows callers to release the storage associated with an object identifier set object allocated by another GSS-API call. This call's specific behavior depends on the language and programming environment within which a GSS-API implementation operates, and is therefore detailed within applicable bindings specifications; in particular, this call may be superfluous within bindings where memory management is automatic.

2.4.9: GSS_Create_empty_OID_set call

Inputs:

o (none)

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o oid_set SET OF OBJECT IDENTIFIER

Return major_status codes:

o GSS_S_COMPLETE indicates successful completion

o GSS_S_FAILURE indicates that the operation failed

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

2.4.10: GSS_Add_OID_set_member call

Inputs:

- member_oid OBJECT IDENTIFIER,
- oid_set SET OF OBJECT IDENTIFIER

Outputs:

- major_status INTEGER,
- minor_status INTEGER,

Return major_status codes:

- GSS_S_COMPLETE indicates successful completion
- GSS_S_FAILURE indicates that the operation failed

Adds 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().

2.4.11: GSS_Test_OID_set_member call

Inputs:

- member OBJECT IDENTIFIER,
- set SET OF OBJECT IDENTIFIER

Outputs:

- major_status INTEGER,
- minor_status INTEGER,
- present BOOLEAN

Return major_status codes:

- GSS_S_COMPLETE indicates successful completion
- GSS_S_FAILURE indicates that the operation failed

Interrogates 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().

2.4.12: GSS_Release_OID call

Inputs:

- oid OBJECT IDENTIFIER

Outputs:

- major_status INTEGER,
- minor_status INTEGER

Return major_status codes:

- GSS_S_COMPLETE indicates successful completion
- GSS_S_FAILURE indicates that the operation failed

Allows the caller to release the storage associated with an OBJECT IDENTIFIER buffer allocated by another GSS-API call. This call's specific behavior depends on the language and programming environment within which a GSS-API implementation operates, and is therefore detailed within applicable bindings specifications; in particular, this call may be superfluous within bindings where memory management is automatic.

2.4.13: GSS_OID_to_str call

Inputs:

- oid OBJECT IDENTIFIER

Outputs:

- major_status INTEGER,
- minor_status INTEGER,
- oid_str OCTET STRING

Return major_status codes:

- GSS_S_COMPLETE indicates successful completion o GSS_S_FAILURE indicates that the operation failed

The function GSS_OID_to_str() returns a string representing the input OID in numeric ASN.1 syntax format (curly-brace enclosed, space-delimited, e.g., "{2 16 840 1 113687 1 2 1}"). The string is releasable using GSS_Release_buffer(). If the input "oid" does not represent a syntactically valid object identifier, GSS_S_FAILURE status is returned and the returned oid_str result is NULL.

2.4.14: GSS_Str_to_OID call

Inputs:

o oid_str OCTET STRING

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o oid OBJECT IDENTIFIER

Return major_status codes:

o GSS_S_COMPLETE indicates successful completion

o GSS_S_FAILURE indicates that the operation failed

The function GSS_Str_to_OID() constructs and returns an OID from its printable form; implementations should be able to accept the numeric ASN.1 syntax form as described for GSS_OID_to_str(), and this form should be used for portability, but implementations of this routine may also accept other formats (e.g., "1.2.3.3"). The OID is suitable for release using the function GSS_Release_OID(). If the input oid_str cannot be translated into an OID, GSS_S_FAILURE status is returned and the "oid" result is NULL.

2.4.15: GSS_Inquire_names_for_mech call

Input:

o input_mech_type OBJECT IDENTIFIER, -- mechanism type

Outputs:

o major_status INTEGER,

- minor_status INTEGER,
- name_type_set SET OF OBJECT IDENTIFIER

Return major_status codes:

- GSS_S_COMPLETE indicates that the output name_type_set contains a list of name types which are supported by the locally available mechanism identified by input_mech_type.
- GSS_S_BAD_MECH indicates that the mechanism identified by input_mech_type was unsupported within the local implementation, causing the query to fail.
- GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

Allows callers to determine the set of name types which are supportable by a specific locally-available mechanism.

2.4.16: GSS_Inquire_mechs_for_name call

Inputs:

- input_name INTERNAL NAME,

Outputs:

- major_status INTEGER,
- minor_status INTEGER,
- mech_types SET OF OBJECT IDENTIFIER

Return major_status codes:

- GSS_S_COMPLETE indicates that a set of object identifiers, corresponding to the set of mechanisms suitable for processing the input_name, is available in mech_types.
- GSS_S_BAD_NAME indicates that the input_name could not be processed.
- GSS_S_BAD_NAMETYPE indicates that the type of the input_name is unsupported by the GSS-API implementation.
- GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

This routine returns the mechanism set with which the input_name may be processed. After use, the mech_types object should be freed by the caller via the GSS_Release_OID_set() call. Note: it is anticipated that implementations of GSS_Inquire_mechs_for_name() will commonly operate based on type information describing the capabilities of available mechanisms; it is not guaranteed that all identified mechanisms will necessarily be able to canonicalize (via GSS_Canonicalize_name()) a particular name.

2.4.17: GSS_Canonicalize_name call

Inputs:

```
o  input_name INTERNAL NAME,

o  mech_type OBJECT IDENTIFIER  -- must be explicit mechanism,
                                   not "default" specifier
```

Outputs:

o major_status INTEGER,

o minor_status INTEGER,

o output_name INTERNAL NAME

Return major_status codes:

o GSS_S_COMPLETE indicates that a mechanism-specific reduction of the input_name, as processed by the mechanism identified by mech_type, is available in output_name.

o GSS_S_BAD_MECH indicates that the identified mechanism is unsupported.

o GSS_S_BAD_NAMETYPE indicates that the input name does not contain an element with suitable type for processing by the identified mechanism.

o GSS_S_BAD_NAME indicates that the input name contains an element with suitable type for processing by the identified mechanism, but that this element could not be processed successfully.

o GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

This routine reduces a GSS-API internal name, which may in general contain elements corresponding to multiple mechanisms, to a mechanism-specific Mechanism Name (MN) by applying the translations corresponding to the mechanism identified by mech_type.

2.4.18: GSS_Export_name call

Inputs:

- input_name INTERNAL NAME, -- required to be MN

Outputs:

- major_status INTEGER,
- minor_status INTEGER,
- output_name OCTET STRING

Return major_status codes:

- GSS_S_COMPLETE indicates that a flat representation of the input name is available in output_name.
- GSS_S_NAME_NOT_MN indicates that the input name contained elements corresponding to multiple mechanisms, so cannot be exported into a single-mechanism flat form.
- GSS_S_BAD_NAME indicates that the input name was an MN, but could not be processed.
- GSS_S_BAD_NAMETYPE indicates that the input name was an MN, but that its type is unsupported by the GSS-API implementation.
- GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

This routine creates a flat name representation, suitable for bytewise comparison or for input to GSS_Import_name() in conjunction with the reserved GSS-API Exported Name Object OID, from a internal-form Mechanism Name (MN) as emitted, e.g., by GSS_Canonicalize_name() or GSS_Accept_sec_context().

The emitted GSS-API Exported Name Object is self-describing; no associated parameter-level OID need be emitted by this call. This flat representation consists of a mechanism-independent wrapper layer, defined in Section 3.2 of this document, enclosing a mechanism-defined name representation.

In all cases, the flat name output by GSS_Export_name() to correspond to a particular input MN must be invariant over time within a particular installation.

The GSS_S_NAME_NOT_MN status code is provided to enable implementations to reject input names which are not MNs. It is not, however, required for purposes of conformance to this specification that all non-MN input names must necessarily be rejected.

2.4.19: GSS_Duplicate_name call

Inputs:

- o src_name INTERNAL NAME

Outputs:

- o major_status INTEGER,
- o minor_status INTEGER,
- o dest_name INTERNAL NAME

Return major_status codes:

- o GSS_S_COMPLETE indicates that dest_name references an internal name object containing the same name as passed to src_name.
- o GSS_S_BAD_NAME indicates that the input name was invalid.
- o GSS_S_BAD_NAMETYPE indicates that the input name's type is unsupported by the GSS-API implementation.
- o GSS_S_FAILURE indicates that the requested operation could not be performed for reasons unspecified at the GSS-API level.

This routine takes input internal name src_name, and returns another reference (dest_name) to that name which can be used even if src_name is later freed. (Note: This may be implemented by copying or through use of reference counts.)

3: Data Structure Definitions for GSS-V2 Usage

Subsections of this section define, for interoperability and portability purposes, certain data structures for use with GSS-V2.

3.1: Mechanism-Independent Token Format

This section specifies a mechanism-independent level of encapsulating representation for the initial token of a GSS-API context establishment sequence, incorporating an identifier of the mechanism type to be used on that context and enabling tokens to be interpreted unambiguously at GSS-API peers. Use of this format is required for initial context establishment tokens of Internet standards-track GSS-API mechanisms; use in non-initial tokens is optional.

The encoding format for the token tag is derived from ASN.1 and DER (per illustrative ASN.1 syntax included later within this subsection), but its concrete representation is defined directly in terms of octets rather than at the ASN.1 level in order to facilitate interoperable implementation without use of general ASN.1 processing code. The token tag consists of the following elements, in order:

1. 0x60 -- Tag for [APPLICATION 0] SEQUENCE; indicates that constructed form, definite length encoding follows.

2. Token length octets, specifying length of subsequent data (i.e., the summed lengths of elements 3-5 in this list, and of the mechanism-defined token object following the tag). This element comprises a variable number of octets:

2a. If the indicated value is less than 128, it shall be represented in a single octet with bit 8 (high order) set to "0" and the remaining bits representing the value.

2b. If the indicated value is 128 or more, it shall be represented in two or more octets, with bit 8 of the first octet set to "1" and the remaining bits of the first octet specifying the number of additional octets. The subsequent octets carry the value, 8 bits per octet, most significant digit first. The minimum number of octets shall be used to encode the length (i.e., no octets representing leading zeros shall be included within the length encoding).

3. 0x06 -- Tag for OBJECT IDENTIFIER

4. Object identifier length -- length (number of octets) of the encoded object identifier contained in element 5, encoded per rules as described in 2a. and 2b. above.

5. Object identifier octets -- variable number of octets, encoded per ASN.1 BER rules:

5a. The first octet contains the sum of two values: (1) the top-level object identifier component, multiplied by 40 (decimal), and (2) the second-level object identifier component. This special case is the only point within an object identifier encoding where a single octet represents contents of more than one component.

5b. Subsequent octets, if required, encode successively-lower components in the represented object identifier. A component's encoding may span multiple octets, encoding 7 bits per octet (most significant bits first) and with bit 8 set to "1" on all but the final octet in the component's encoding. The minimum number of octets shall be used to encode each component (i.e., no octets representing leading zeros shall be included within a component's encoding).

(Note: In many implementations, elements 3-5 may be stored and referenced as a contiguous string constant.)

The token tag is immediately followed by a mechanism-defined token object. Note that no independent size specifier intervenes following the object identifier value to indicate the size of the mechanism-defined token object. While ASN.1 usage within mechanism-defined tokens is permitted, there is no requirement that the mechanism-specific innerContextToken, innerMsgToken, and sealedUserData data elements must employ ASN.1 BER/DER encoding conventions.

The following ASN.1 syntax is included for descriptive purposes only, to illustrate structural relationships among token and tag objects. For interoperability purposes, token and tag encoding shall be performed using the concrete encoding procedures described earlier in this subsection.

```
GSS-API DEFINITIONS ::=

BEGIN

MechType ::= OBJECT IDENTIFIER
-- data structure definitions

-- callers must be able to distinguish among
-- InitialContextToken, SubsequentContextToken,
-- PerMsgToken, and SealedMessage data elements
-- based on the usage in which they occur

InitialContextToken ::=
-- option indication (delegation, etc.) indicated within
-- mechanism-specific token
[APPLICATION 0] IMPLICIT SEQUENCE {
        thisMech MechType,
        innerContextToken ANY DEFINED BY thisMech
           -- contents mechanism-specific
           -- ASN.1 structure not required
        }

SubsequentContextToken ::= innerContextToken ANY
-- interpretation based on predecessor InitialContextToken
-- ASN.1 structure not required

PerMsgToken ::=
-- as emitted by GSS_GetMIC and processed by GSS_VerifyMIC
-- ASN.1 structure not required
        innerMsgToken ANY

SealedMessage ::=
-- as emitted by GSS_Wrap and processed by GSS_Unwrap
-- includes internal, mechanism-defined indicator
-- of whether or not encrypted
-- ASN.1 structure not required
        sealedUserData ANY

END
```

3.2: Mechanism-Independent Exported Name Object Format

This section specifies a mechanism-independent level of encapsulating representation for names exported via the GSS_Export_name() call, including an object identifier representing the exporting mechanism. The format of names encapsulated via this representation shall be defined within individual mechanism drafts. Name objects of this type will be identified with the following Object Identifier:

{1(iso), 3(org), 6(dod), 1(internet), 5(security), 6(nametypes), 4(gss-api-exported-name)}

No name type OID is included in this mechanism-independent level of format definition, since (depending on individual mechanism specifications) the enclosed name may be implicitly typed or may be explicitly typed using a means other than OID encoding.

Length	Name	Description
2	TOK_ID	Token Identifier For exported name objects, this must be hex 04 01.
2	MECH_OID_LEN	Length of the Mechanism OID
MECH_OID_LEN	MECH_OID	Mechanism OID, in DER
4	NAME_LEN	Length of name
NAME_LEN	NAME	Exported name; format defined in applicable mechanism draft.

4: Name Type Definitions

This section includes definitions for name types and associated syntaxes which are defined in a mechanism-independent fashion at the GSS-API level rather than being defined in individual mechanism specifications.

4.1: Host-Based Service Name Form

The following Object Identifier value is provided as a means to identify this name form:

{1(iso), 3(org), 6(dod), 1(internet), 5(security), 6(nametypes), 2(gss-host-based-services)}

The recommended symbolic name for this type is "GSS_C_NT_HOSTBASED_SERVICE".

This name type is used to represent services associated with host computers. This name form is constructed using two elements, "service" and "hostname", as follows:

service@hostname

When a reference to a name of this type is resolved, the "hostname" is canonicalized by attempting a DNS lookup and using the fully-qualified domain name which is returned, or by using the "hostname" as provided if the DNS lookup fails. The canonicalization operation also maps the host's name into lower-case characters.

The "hostname" element may be omitted. If no "@" separator is included, the entire name is interpreted as the service specifier, with the "hostname" defaulted to the canonicalized name of the local host.

Values for the "service" element are registered with the IANA.

4.2: User Name Form

This name form shall be represented by the Object Identifier {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) user_name(1)}. The recommended mechanism-independent symbolic name for this type is "GSS_C_NT_USER_NAME". (Note: the same name form and OID is defined within the Kerberos V5 GSS-API mechanism, but the symbolic name recommended there begins with a "GSS_KRB5_NT_" prefix.)

This name type is used to indicate a named user on a local system. Its interpretation is OS-specific. This name form is constructed as:

username

4.3: Machine UID Form

This name form shall be represented by the Object Identifier {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) machine_uid_name(2)}. The recommended mechanism-independent symbolic name for this type is "GSS_C_NT_MACHINE_UID_NAME". (Note: the same name form and OID is defined within the Kerberos V5 GSS-API mechanism, but the symbolic name recommended there begins with a "GSS_KRB5_NT_" prefix.)

This name type is used to indicate a numeric user identifier corresponding to a user on a local system. Its interpretation is OS-specific. The gss_buffer_desc representing a name of this type should contain a locally-significant uid_t, represented in host byte order. The GSS_Import_name() operation resolves this uid into a username, which is then treated as the User Name Form.

4.4: String UID Form

This name form shall be represented by the Object Identifier {iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2) generic(1) string_uid_name(3)}. The recommended symbolic name for this type is "GSS_C_NT_STRING_UID_NAME". (Note: the same name form and OID is defined within the Kerberos V5 GSS-API mechanism, but the symbolic name recommended there begins with a "GSS_KRB5_NT_" prefix.)

This name type is used to indicate a string of digits representing the numeric user identifier of a user on a local system. Its interpretation is OS-specific. This name type is similar to the Machine UID Form, except that the buffer contains a string representing the uid_t.

5: Mechanism-Specific Example Scenarios

This section provides illustrative overviews of the use of various candidate mechanism types to support the GSS-API. These discussions are intended primarily for readers familiar with specific security technologies, demonstrating how GSS-API functions can be used and implemented by candidate underlying mechanisms. They should not be regarded as constrictive to implementations or as defining the only means through which GSS-API functions can be realized with a particular underlying technology, and do not demonstrate all GSS-API features with each technology.

5.1: Kerberos V5, single-TGT

OS-specific login functions yield a TGT to the local realm Kerberos server; TGT is placed in a credentials structure for the client. Client calls GSS_Acquire_cred() to acquire a cred_handle in order to reference the credentials for use in establishing security contexts.

Client calls GSS_Init_sec_context(). If the requested service is located in a different realm, GSS_Init_sec_context() gets the necessary TGT/key pairs needed to traverse the path from local to target realm; these data are placed in the owner’s TGT cache. After any needed remote realm resolution, GSS_Init_sec_context() yields a service ticket to the requested service with a corresponding session key; these data are stored in conjunction with the context. GSS-API code sends KRB_TGS_REQ request(s) and receives KRB_TGS_REP response(s) (in the successful case) or KRB_ERROR.

Assuming success, GSS_Init_sec_context() builds a Kerberos-formatted KRB_AP_REQ message, and returns it in output_token. The client sends the output_token to the service.

The service passes the received token as the input_token argument to GSS_Accept_sec_context(), which verifies the authenticator, provides the service with the client's authenticated name, and returns an output_context_handle.

Both parties now hold the session key associated with the service ticket, and can use this key in subsequent GSS_GetMIC(), GSS_VerifyMIC(), GSS_Wrap(), and GSS_Unwrap() operations.

5.2: Kerberos V5, double-TGT

TGT acquisition as above.

Note: To avoid unnecessary frequent invocations of error paths when implementing the GSS-API atop Kerberos V5, it seems appropriate to represent "single-TGT K-V5" and "double-TGT K-V5" with separate mech_types, and this discussion makes that assumption.

Based on the (specified or defaulted) mech_type, GSS_Init_sec_context() determines that the double-TGT protocol should be employed for the specified target. GSS_Init_sec_context() returns GSS_S_CONTINUE_NEEDED major_status, and its returned output_token contains a request to the service for the service's TGT. (If a service TGT with suitably long remaining lifetime already exists in a cache, it may be usable, obviating the need for this step.) The client passes the output_token to the service. Note: this scenario illustrates a different use for the GSS_S_CONTINUE_NEEDED status return facility than for support of mutual authentication; note that both uses can coexist as successive operations within a single context establishment operation.

The service passes the received token as the input_token argument to GSS_Accept_sec_context(), which recognizes it as a request for TGT. (Note that current Kerberos V5 defines no intra-protocol mechanism to represent such a request.) GSS_Accept_sec_context() returns GSS_S_CONTINUE_NEEDED major_status and provides the service's TGT in its output_token. The service sends the output_token to the client.

The client passes the received token as the input_token argument to a continuation of GSS_Init_sec_context(). GSS_Init_sec_context() caches the received service TGT and uses it as part of a service ticket request to the Kerberos authentication server, storing the returned service ticket and session key in conjunction with the context. GSS_Init_sec_context() builds a Kerberos-formatted authenticator, and returns it in output_token along with GSS_S_COMPLETE return major_status. The client sends the output_token to the service.

Service passes the received token as the input_token argument to a continuation call to GSS_Accept_sec_context(). GSS_Accept_sec_context() verifies the authenticator, provides the service with the client's authenticated name, and returns major_status GSS_S_COMPLETE.

GSS_GetMIC(), GSS_VerifyMIC(), GSS_Wrap(), and GSS_Unwrap() as above.

5.3: X.509 Authentication Framework

This example illustrates use of the GSS-API in conjunction with public-key mechanisms, consistent with the X.509 Directory Authentication Framework.

The GSS_Acquire_cred() call establishes a credentials structure, making the client's private key accessible for use on behalf of the client.

The client calls GSS_Init_sec_context(), which interrogates the Directory to acquire (and validate) a chain of public-key certificates, thereby collecting the public key of the service. The certificate validation operation determines that suitable integrity checks were applied by trusted authorities and that those certificates have not expired. GSS_Init_sec_context() generates a secret key for use in per-message protection operations on the context, and enciphers that secret key under the service's public key.

The enciphered secret key, along with an authenticator quantity signed with the client's private key, is included in the output_token from GSS_Init_sec_context(). The output_token also carries a certification path, consisting of a certificate chain leading from the service to the client; a variant approach would defer this path resolution to be performed by the service instead of being asserted by the client. The client application sends the output_token to the service.

The service passes the received token as the input_token argument to GSS_Accept_sec_context(). GSS_Accept_sec_context() validates the certification path, and as a result determines a certified binding between the client's distinguished name and the client's public key. Given that public key, GSS_Accept_sec_context() can process the input_token's authenticator quantity and verify that the client's private key was used to sign the input_token. At this point, the client is authenticated to the service. The service uses its private key to decipher the enciphered secret key provided to it for per-message protection operations on the context.

The client calls GSS_GetMIC() or GSS_Wrap() on a data message, which causes per-message authentication, integrity, and (optional) confidentiality facilities to be applied to that message. The service uses the context's shared secret key to perform corresponding GSS_VerifyMIC() and GSS_Unwrap() calls.

6: Security Considerations

Security issues are discussed throughout this memo.

7: Related Activities

In order to implement the GSS-API atop existing, emerging, and future security mechanisms:

> object identifiers must be assigned to candidate GSS-API mechanisms and the name types which they support
>
> concrete data element formats and processing procedures must be defined for candidate mechanisms

Calling applications must implement formatting conventions which will enable them to distinguish GSS-API tokens from other data carried in their application protocols.

Concrete language bindings are required for the programming environments in which the GSS-API is to be employed, as RFC-1509 defines for the C programming language and GSS-V1.

APPENDIX A

MECHANISM DESIGN CONSTRAINTS

The following constraints on GSS-API mechanism designs are adopted in response to observed caller protocol requirements, and adherence thereto is anticipated in subsequent descriptions of GSS-API mechanisms to be documented in standards-track Internet specifications.

It is strongly recommended that mechanisms offering per-message protection services also offer at least one of the replay detection and sequencing services, as mechanisms offering neither of the latter will fail to satisfy recognized requirements of certain candidate caller protocols.

APPENDIX B

COMPATIBILITY WITH GSS-V1

It is the intent of this document to define an interface and procedures which preserve compatibility between GSS-V1 (RFC-1508) callers and GSS- V2 providers. All calls defined in GSS-V1 are preserved, and it has been a goal that GSS-V1 callers should be able to operate atop GSS-V2 provider implementations. Certain detailed changes, summarized in this section, have been made in order to resolve omissions identified in GSS-V1.

The following GSS-V1 constructs, while supported within GSS-V2, are deprecated:

> Names for per-message processing routines: GSS_Seal() deprecated in favor of GSS_Wrap(); GSS_Sign() deprecated in favor of GSS_GetMIC(); GSS_Unseal() deprecated in favor of GSS_Unwrap(); GSS_Verify() deprecated in favor of GSS_VerifyMIC().
>
> GSS_Delete_sec_context() facility for context_token usage, allowing mechanisms to signal context deletion, is retained for compatibility with GSS-V1. For current usage, it is recommended that both peers to a context invoke GSS_Delete_sec_context() independently, passing a null output_context_token buffer to indicate that no context_token is required. Implementations of GSS_Delete_sec_context() should delete relevant locally-stored context information.

This GSS-V2 specification adds the following calls which are not present in GSS-V1:

Credential management calls: GSS_Add_cred(), GSS_Inquire_cred_by_mech().

Context-level calls: GSS_Inquire_context(), GSS_Wrap_size_limit(), GSS_Export_sec_context(), GSS_Import_sec_context().

Per-message calls: No new calls. Existing calls have been renamed.

Support calls: GSS_Create_empty_OID_set(), GSS_Add_OID_set_member(), GSS_Test_OID_set_member(), GSS_Release_OID(), GSS_OID_to_str(), GSS_Str_to_OID(), GSS_Inquire_names_for_mech(), GSS_Inquire_mechs_for_name(), GSS_Canonicalize_name(), GSS_Export_name(), GSS_Duplicate_name().

This GSS-V2 specification introduces three new facilities applicable to security contexts, indicated using the following context state values which are not present in GSS-V1:

anon_state, set TRUE to indicate that a context's initiator is anonymous from the viewpoint of the target; Section 1.2.5 of this specification provides a summary description of the GSS-V2 anonymity support facility, support and use of which is optional.

prot_ready_state, set TRUE to indicate that a context may be used for per-message protection before final completion of context establishment; Section 1.2.7 of this specification provides a summary description of the GSS-V2 facility enabling mechanisms to selectively permit per-message protection during context establishment, support and use of which is optional.

trans_state, set TRUE to indicate that a context is transferable to another process using the GSS-V2 GSS_Export_sec_context() facility.

These state values are represented (at the C bindings level) in positions within a bit vector which are unused in GSS-V1, and may be safely ignored by GSS-V1 callers.

Relative to GSS-V1, GSS-V2 provides additional guidance to GSS-API implementors in the following areas: implementation robustness, credential management, behavior in multi-mechanism configurations, naming support, and inclusion of optional sequencing services. The token tagging facility as defined in GSS-V2, Section 3.1, is now described directly in terms of octets to facilitate interoperable implementation without general ASN.1 processing code; the corresponding ASN.1 syntax, included for descriptive purposes, is unchanged from that in GSS-V1. For use in conjunction with added naming support facilities, a new Exported Name Object construct is added. Additional name types are introduced in Section 4.

This GSS-V2 specification adds the following major_status values which are not defined in GSS-V1:

GSS_S_BAD_QOP	unsupported QOP value
GSS_S_UNAUTHORIZED	operation unauthorized
GSS_S_UNAVAILABLE	operation unavailable
GSS_S_DUPLICATE_ELEMENT	duplicate credential element requested
GSS_S_NAME_NOT_MN	name contains multi-mechanism elements
GSS_S_GAP_TOKEN	skipped predecessor token(s) detected

Of these added status codes, only two values are defined to be returnable by calls existing in GSS-V1: GSS_S_BAD_QOP (returnable by GSS_GetMIC() and GSS_Wrap()), and GSS_S_GAP_TOKEN (returnable by GSS_VerifyMIC() and GSS_Unwrap()).

Additionally, GSS-V2 descriptions of certain calls present in GSS-V1 have been updated to allow return of additional major_status values from the set as defined in GSS-V1: GSS_Inquire_cred() has GSS_S_DEFECTIVE_CREDENTIAL and GSS_S_CREDENTIALS_EXPIRED defined as returnable, GSS_Init_sec_context() has GSS_S_OLD_TOKEN, GSS_S_DUPLICATE_TOKEN, and GSS_S_BAD_MECH defined as returnable, and GSS_Accept_sec_context() has GSS_S_BAD_MECH defined as returnable.

Author’s Address

John Linn
OpenVision Technologies
One Main St.
Cambridge, MA 02142 USA

Phone: +1 617.374.2245
EMail: John.Linn@ov.com