NETWORK WORKING GROUP
INTERNET-DRAFT
Expires: April 8, 2006

J. C. Luciani
Novell, Inc.
November 8, 2005

GSS_API V2: C# Bindings
draft-ietf-kitten-gssapi-csharp-bindings-00.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

The Generic Security Services Application Program Interface (GSS-API) offers application programmers uniform access to security services atop a variety of underlying cryptographic mechanisms. This document specifies the C# language bindings for GSS-API which is described at a language independent conceptual level in RFC 2743 [RFC2743].

The GSS-API C# bindings were designed to emulate the Java bindings as defined in RFC 2853 [RFC2853].

Table of Contents

1.Introduction

This document specifies the C# language bindings for the Generic Security Services Application Programming Interface Version 2 (GSS-API v2). GSS-API allows a caller application to authenticate a principal identity, to delegate rights to a peer, and to apply security services such as confidentiality and integrity on a per-message basis.

One of the design goals utilized when defining the C# bindings for GSS-API was to emulate the Java bindings specified in RFC 2853 as much as possible while still taking advantage of C# features such as Properties. By emulating the Java bindings, we hoped to leverage work already done and to make life easier for developers utilizing GSS-API under C# and Java. As a result of this design goal, the C# bindings match the Java bindings very closely.

Because of the similarity between the Java and C# bindings and in the spirit of leveraging work already done, this document borrows heavily from RFC 2853.

2.GSS-API Operational Paradigm

The Generic Security Service Application Programming Interface Version 2 defines a generic security API to calling applications. It allows a communicating application to authenticate a user associated with another application, to delegate rights to another application, and to apply security services such as confidentiality and integrity on a per-message basis.

There are four stages to using GSS-API:

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

2) A pair of communicating applications establish a joint security context using their credentials. The security context encapsulates shared state information, which is required in order that per-message security services may be provided. Examples of state information that might be shared between applications as part of a security context are cryptographic keys, and message sequence numbers. As part of the establishment of a security context, the context initiator is authenticated to the responder, and may require that the responder is authenticated back to the initiator. The initiator may optionally give the responder the right to initiate further security contexts, acting as an agent or

delegate of the initiator. This transfer of rights is termed "delegation", and is achieved by creating a set of credentials, similar to those used by the initiating application, but which may be used by the responder.

A GSSContext object is used to establish and maintain the shared information that makes up the security context. Certain GSSContext methods will generate a token, which applications treat as cryptographically protected, opaque data. The caller of such GSSContext method is responsible for transferring the token to the peer application, encapsulated if necessary in an application-to-application protocol. On receipt of such a token, the peer application should pass it to a corresponding GSSContext method which will decode the token and extract the information, updating the security context state information accordingly.

3) Per-message services are invoked on a GSSContext object to apply either:

integrity and data origin authentication, or

confidentiality, integrity and data origin authentication

to application data, which are treated by GSS-API as arbitrary octet-strings. An application transmitting a message that it wishes to protect will call the appropriate GSSContext method (getMIC or wrap) to apply protection, and send the resulting token to the receiving application. The receiver will pass the received token (and, in the case of data protected by getMIC, the accompanying message-data) to the corresponding decoding method of the GSSContext interface (verifyMIC or unwrap) to remove the protection and validate the data.

4) At the completion of a communications session (which may extend across several transport connections), each application uses a GSSContext method to invalidate the security context and release any system or cryptographic resources held. Multiple contexts may also be used (either successively or simultaneously) within a single communications association, at the discretion of the applications.

3. Additional Controls

This section discusses the optional services that a context initiator may request of the GSS-API before the context establishment. Each of these services is requested by manipulating the appropriate property of the GSSContext interface before the first call to init is performed.

Only the context initiator can request context flags.

The optional services defined are:

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

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

Replay Detection
: In addition to providing message integrity services, GSSContext per-message operations of getMIC and wrap should include message numbering information to enable verifyMIC and unwrap to detect if a message has been duplicated.

Out-of-Sequence Detection
: In addition to providing message integrity services, GSSContext per-message operations (getMIC and wrap) should include message sequencing information to enable verifyMIC and unwrap to detect if a message has been received out of sequence.

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

Some mechanisms may not support all optional services, and some mechanisms may only support some services in conjunction with others. The GSSContext interface offers query methods to allow the verification by the calling application of which services will be available from the context when the establishment phase is complete. In general, if the security mechanism is capable of providing a requested service, it should do so even if additional services must be enabled in order to provide the requested service. If the mechanism is incapable of providing a requested service, it should proceed without the service leaving the application to abort the context establishment process if it considers the requested service to be mandatory.

Some mechanisms may specify that support for some services is optional, and that implementers of the mechanism need not provide it. This is most commonly true of the confidentiality service, often because of legal restrictions on the use of data-encryption, but may apply to any of the services. Such mechanisms are required to send at least one token from acceptor to initiator during context

establishment when the initiator indicates a desire to use such a service, so that the initiating GSS-API can correctly indicate whether the service is supported by the acceptor's GSS-API.

3.1. Delegation

The GSS-API allows delegation to be controlled by the initiating application via manipulation of the credential delegation property before the first call to init has been issued. Some mechanisms do not support delegation, and for such mechanisms attempts by an application to enable delegation are ignored.

The acceptor of a security context, for which the initiator enabled delegation, can check if delegation was enabled by reading the credential delegation property of the GSSContext object. In cases when it is, the delegated credential object can be obtained by reading the delegated credential property. The obtained GSSCredential object may then be used to initiate subsequent GSS-API security contexts as an agent or delegate of the initiator. If the original initiator's identity is "A" and the delegate's identity is "B", then, depending on the underlying mechanism, the identity embodied by the delegated credential may be either "A" or "B acting for A".

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

3.2. Mutual Authentication

Usually, a context acceptor will require that a context initiator authenticate itself so that the acceptor may make an access-control decision prior to performing a service for the initiator. In some cases, the initiator may also request that the acceptor authenticate

itself. GSS-API allows the initiating application to request this

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

3.3. Replay and Out-of-Sequence Detection

The GSS-API may provide detection of mis-ordered messages once a security context has been established. Protection may be applied to messages by either application, by calling either getMIC or wrap methods of the GSSContext object, and verified by the peer application by calling verifyMIC or unwrap for the peer's GSSContext object.

The getMIC method calculates a cryptographic checksum of an application message, and returns that checksum in a token. The application should pass both the token and the message to the peer application, which presents them to the verifyMIC method of the peer's GSSContext object.

The wrap method calculates a cryptographic checksum of an application message, and places both the checksum and the message inside a single token. The application should pass the token to the peer application, which presents it to the unwrap method of the peer's GSSContext object to extract the message and verify the checksum.

Either pair of routines may be capable of detecting out-of-sequence message delivery, or duplication of messages. Details of such mis-ordered messages are indicated through supplementary query methods of the MessageProp object that is filled in by each of these routines.

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

3.4. Anonymous Authentication

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

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

In addition to informing the application that a context is established anonymously (via the isAnonymous property of the GSSContext interface), the srcName property of the acceptor's GSSContext object will, for such contexts, be set to a reserved internal-form name, defined by the implementation.

The ToString method for a GSSName object representing an anonymous entity will return a printable name. The returned value will be syntactically distinguishable from any valid principal name supported by the implementation. The associated name-type object identifier will be an oid representing the value of GSSNameTypes.NT_ANONYMOUS. This name-type oid will be defined as a public, static Oid object of the GSSName interface. The printable form of an anonymous name should be chosen such that it implies anonymity, since this name may appear in, for example, audit logs. For example, the string "<anonymous>" might be a good choice, if no valid printable names supported by the implementation can begin with "<" and end with ">".

When using the equal method of the GSSName interface, and one of the operands is a GSSName instance representing an anonymous entity, the method must return "false".

3.5. Confidentiality

If a GSSContext supports the confidentiality service, wrap method may be used to encrypt application messages. Messages are selectively encrypted, under the control of the setPrivacy property of the MessageProp object used in the wrap method.

3.6. Inter-process Context Transfer

GSS-API V2 provides functionality which allows a security context to be transferred between processes on a single machine. These are implemented using the export method of GSSContext and a byte array constructor of the same interface. The most common use for such a feature is a client-server design where the server is implemented as a single process that accepts incoming security contexts, which then launches child processes to deal with the data on these contexts. In such a design, the child processes must have access to the security context object created within the parent so that they can use per-message protection services and delete the security context when the communication session ends.

Since the security context data structure is expected to contain sequencing information, it is impractical in general to share a context between processes. Thus GSSContext interface provides an export method that the process, which currently owns the context, can call to declare that it has no intention to use the context subsequently, and to create an inter-process token containing information needed by the adopting process to successfully re-create the context. After successful completion of export, the original security context is made inaccessible to the calling process by GSS-API and any further usage of this object will result in failures. The originating process transfers the inter-process token to the adopting process, which creates a new GSSContext object using the byte array constructor. The properties of the context are equivalent to that of the original context.

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

Implementations are not required to support the inter-process transfer of security contexts. Reading the isTransferable property of the GSSContext interface will indicate if the context object is transferable.

3.7. The Use of Incomplete Contexts

Some mechanisms may allow the per-message services to be used before the context establishment process is complete. For example, a mechanism may include sufficient information in its initial context-level tokens for the context acceptor to immediately decode messages protected with wrap or getMIC. For such a mechanism, the initiating application need not wait until subsequent context-level tokens have been sent and received before invoking the per-message protection services.

An application can read the isProtReady property of the GSSContext object to determine if the per-message services are available in advance of complete context establishment. Applications wishing to use per-message protection services on partially-established contexts should query this method before attempting to invoke wrap or getMIC.

4. C# GSS-API Overview

The C# GSS-API leverages many of the native C# types as well as features of the operating environment such as automatic garbage collection, exception handling, and encapsulation. This allows for an API that is simpler and with fewer functions than the API described in RFC 1509 [RFC1509].

The assemblies implementing GSS-API are not part of the .NET and Mono frameworks. Because of this, it is necessary for the assemblies to be installed before they can be used by applications.

C# GSS-API, unlike its Java counterpart does not allow applications to choose between different security providers. The provider utilized by the application is determine at application build time based on the C# GSS-API assemblies linked in. The mechanisms supported by the C# GSS-API assemblies are not controlled by the application. C# GSS-API assemblies are vendor specific.

The GSS-API types are present in the org.ietf.gss namespace.

Applications need to include the following line at the top of their listings to make use of the GSS-API types:

```
using org.ietf.gss;
```

4.1. Object Identifiers

An Oid object will be used to represent Universal Object Identifiers (Oids). Oids are ISO-defined, hierarchically globally-interpretable identifiers used within the GSS-API framework to identify security mechanisms and name formats. The Oid object can be created from a string representation of its dot notation (e.g. "1.3.6.1.5.6.2") as well as from its ASN.1 DER encoding. Methods are also provided to test equality and provide the DER representation for the object.

An important feature of the Oid class is that its instances are immutable - i.e. there are no methods defined that allow one to change the contents of an Oid. This property allows one to treat these objects as "statics" without the need to perform copies.

Certain routines allow the usage of a default oid. A "null" value can be used in those cases.

4.2. Object Identifier Sets

Object identifiers sets are represented as arrays of Oid objects. C# arrays contain a length field which allows for easy manipulation and reference.

In order to support the full functionality of RFC 2743, the Oid class includes a method which checks for existence of an Oid object within a specified array. This is equivalent in functionality to gss_test_oid_set_member. The use of C# arrays and C#'s automatic garbage collection has eliminated the need for the following routines: gss_create_empty_oid_set, gss_release_oid_set, and gss_add_oid_set_member. C# GSS-API implementations will not contain them. C#'s automatic garbage collection and the immutable property of the Oid object eliminates the complicated memory management issues of the C counterpart.

When ever a default value for an Object Identifier Set is required, a "null" value can be used. Please consult the detailed method description for details.

4.3. Credentials

GSS-API credentials are represented by the GSSCredential interface. The interface contains several constructs to allow for the creation of most common credential objects for the initiator and the acceptor. Comparisons are performed using the interface's "Equals" method. The following general description of GSS-API credentials is included from the C-bindings specification:

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

Credentials describe a set of mechanism-specific principals, and give their holder the ability to act as any of those principals. All principal identities asserted by a single GSS-API credential should belong to the same entity, although enforcement of this property is an implementation-specific matter. A single GSSCredential object represents all the credential elements that have been acquired.

The creation's of an GSSContext object allows the value of "null" to be specified as the GSSCredential input parameter. This will indicate 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:

For the initiator side of the context:

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

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

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

otherwise

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

and for the acceptor side of the context

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

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

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

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

The purpose of the above rules is to allow security contexts to be established by both initiator and acceptor using the default behavior whenever possible. Applications requesting default behavior are likely to be more portable across mechanisms and implementations than ones that instantiate an GSSCredential object representing a specific identity.

4.4. Contexts

The GSSContext interface is used to represent one end of a GSS-API security context, storing state information appropriate to that end of the peer communication, including cryptographic state information. The instantiation of the context object is done differently by the initiator and the acceptor. After the context has been instantiated, the initiator may choose to set various context options which will determine the characteristics of the desired security context. When all the application desired characteristics have been set, the initiator will call the initSecContext method which will produce a token for consumption by the peer's acceptSecContext method. It is the responsibility of the application to deliver the authentication token(s) between the peer applications for processing. Upon completion of the context establishment phase, context attributes can be retrieved, by both the initiator and acceptor, reading the properties of the GSSContext object. These will reflect the actual attributes of the established context. At this point the context can be used by the application to apply cryptographic services to its

data.

4.5. Authentication Tokens

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

C# GSS-API uses byte arrays to represent authentication tokens. Overloaded methods exist which allow the caller to supply streams which will be used for the reading and writing of the token data.

4.6. Interprocess Tokens

Certain GSS-API routines are intended to transfer data between processes in multi-process programs. These routines use a caller-opaque octet-string, generated by the GSS-API in one process for use by the GSS-API in another process. The calling application is responsible for transferring such tokens between processes. Note that, while GSS-API implementors are encouraged to avoid placing sensitive information within interprocess tokens, or to cryptographically protect them, many implementations will be unable to avoid placing key material or other sensitive data within them. It is the application's responsibility to ensure that interprocess tokens are protected in transit, and transferred only to processes that are trustworthy. An interprocess token is represented using a byte array emitted from the export method of the GSSContext interface. The receiver of the interprocess token would initialize a GSSContext object with this token to create a new context. Once a context has been exported, the GSSContext object is invalidated.

4.7. Error Reporting

RFC 2743 defined the usage of major and minor status values for signaling of GSS-API errors. The major code, also called GSS status code, is used to signal errors at the GSS-API level independent of the underlying mechanism(s). The minor status value or Mechanism status code, is a mechanism defined error value indicating a mechanism specific error code.

C# GSS-API uses exceptions implemented by the GSSException class to signal both minor and major error values. Both mechanism specific errors and GSS-API level errors are signaled through instances of this class. The usage of exceptions replaces the need for major and minor codes to be used within the API calls. GSSException class also

contains methods to obtain textual representations for both the major and minor values, which is equivalent to the functionality of gss_display_status.

4.7.1. GSS Status Codes

GSS status codes indicate errors that are independent of the underlying mechanism(s) used to provide the security service. The errors that can be indicated via a GSS status code are generic API routine errors (errors that are defined in the GSS-API specification). These bindings take advantage of the C# exceptions mechanism, thus eliminating the need for calling errors.

A GSS status code indicates a single fatal generic API error from the routine that has thrown the GSSException. Using exceptions announces that a fatal error has occurred during the execution of the method. The GSS-API operational model also allows for the signaling of supplementary status information from the per-message calls. These need to be handled as return values since using exceptions is not appropriate for informatory or warning-like information. The methods that are capable of producing supplementary information are the two per-message methods GSSContext.verifyMIC() and GSSContext.unwrap(). These methods fill the supplementary status codes in the MessageProp object that was passed in.

GSSException object, along with providing the functionality for setting of the various error codes and translating them into textual representation, also contains the definitions of all the numeric error values. The following table lists the definitions of error codes:

Table: GSS Status Codes

Name	Value	Meaning
BAD_MECH	1	An unsupported mechanism was requested.
BAD_NAME	2	An invalid name was supplied.
BAD_NAMETYPE	3	A supplied name was of an unsupported type.
BAD_BINDINGS	4	Incorrect channel bindings were supplied.
BAD_STATUS	5	An invalid status code was supplied.

BAD_MIC	6	A token had an invalid MIC.
NO_CRED	7	No credentials were supplied, or the credentials were unavailable or inaccessible.
NO_CONTEXT	8	Invalid context has been supplied.
DEFECTIVE_TOKEN	9	A supplied token was invalid.
DEFECTIVE_CREDENTIAL	10	A supplied credential was invalid.
CREDENTIALS_EXPIRED	11	The referenced credentials have expired.
CONTEXT_EXPIRED	12	The context has expired.
FAILURE	13	Miscellaneous failure, unspecified at the GSS-API level.
BAD_QOP	14	The quality-of-protection requested could not be provided.
UNAUTHORIZED	15	The operation is forbidden by local security policy.
UNAVAILABLE	16	The operation or option is unavailable.
DUPLICATE_ELEMENT	17	The requested credential element already exists.
NAME_NOT_MN	18	The provided name was not a mechanism name.
OLD_TOKEN	19	The token's validity period has expired.
DUPLICATE_TOKEN	20	The token was a duplicate of an earlier version.

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

The different major status codes that can be contained in the

GSSException object thrown by the methods in this specification are the same as the major status codes returned by the corresponding calls in RFC 2743.

4.7.2. Mechanism-specific Status Codes

Mechanism-specific status codes are communicated in two ways, they are part of any GSSException thrown from the mechanism specific layer to signal a fatal error, or they are part of the MessageProp object that the per-message calls use to signal non-fatal errors.

A default value of 0 in either the GSSException object or the MessageProp object will be used to represent the absence of any mechanism specific status code.

4.7.3. Supplementary Status Codes

Supplementary status codes are confined to the per-message methods of the GSSContext interface. Because of the informative nature of these errors it is not appropriate to use exceptions to signal them. Instead, the per-message operations of the GSSContext interface return these values in a MessageProp object.

The MessageProp class defines boolean properties indicating the following supplementary states:

Table: Supplementary Status Methods

Method Name	Meaning when set to "true"
isDuplicateToken	The token was a duplicate of an earlier token.
isOldToken	The token's validity period has expired.
isUnseqToken	A later token has already been processed.
isGapToken	An expected per-message token was not received.

A "true" value for any of the above properties indicates that the token exhibited the specified property. The application must determine the appropriate course of action for these supplementary values. They are not treated as errors by the GSS-API.

4.8. Names

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

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

Two distinct conceptual representations are defined for names:

1) A GSS-API form represented by implementations of the GSSName interface: A single GSSName object may contain multiple names from different namespaces, but all names should refer to the same entity. An example of such an internal name would be the name returned from a call to the getName method of the GSSCredential interface, when applied to a credential containing credential elements for multiple authentication mechanisms employing different namespaces. This GSSName object will contain a distinct name for the entity for each authentication mechanism.

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

2) Mechanism-specific contiguous byte array and string forms: Different GSSName initialization methods are provided to handle both byte array and string formats and to accommodate various calling applications and name types. These formats are capable of containing only a single name (from a single namespace). Contiguous string names are always accompanied by an object identifier specifying the namespace to which the name belongs, and their format is dependent on the authentication mechanism that employs that name. The string name forms are assumed to be printable, and may therefore be used by GSS-API applications for communication with their users. The byte array name formats are assumed to be in non-printable formats (e.g. the byte array returned from the export method of the GSSName interface).

A GSSName object can be converted to a contiguous representation by using the ToString method. This will guarantee that the name will be converted to a printable format. Different initialization methods in the GSSName interface are defined allowing support for multiple syntaxes for each supported namespace, and allowing users the freedom to choose a preferred name representation. The ToString method should use an implementation-chosen printable syntax for each

supported name-type. To obtain the printable name type, the

getStringNameType method can be used.

There is no guarantee that calling the ToString method on the GSSName interface will produce the same string form as the original imported string name. Furthermore, it is possible that the name was not even constructed from a string representation. The same applies to name-space identifiers which may not necessarily survive unchanged after a journey through the internal name-form. An example of this might be a mechanism that authenticates X.500 names, but provides an algorithmic mapping of Internet DNS names into X.500. That mechanism's implementation of GSSName might, when presented with a DNS name, generate an internal name that contained both the original DNS name and the equivalent X.500 name. Alternatively, it might only store the X.500 name. In the latter case, the ToString method of GSSName would most likely generate a printable X.500 name, rather than the original DNS name.

The context acceptor can obtain a GSSName object representing the entity performing the context initiation (by reading the srcName property). Since this name has been authenticated by a single mechanism, it contains only a single name (even if the internal name presented by the context initiator to the GSSContext object had multiple components). Such names are termed internal mechanism names, or "MN"s and the property values srcName and targName of the GSSContext interface are always of this type. Since some applications may require MNs without wanting to incur the overhead of an authentication operation, creation methods are provided that take not only the name buffer and name type, but also the mechanism oid for which this name should be created. When dealing with an existing GSSName object, the canonicalize method may be invoked to convert a general internal name into an MN.

GSSName objects can be compared using their Equal method, which returns "true" if the two names being compared refer to the same entity. This is the preferred way to perform name comparisons instead of using the printable names that a given GSS-API implementation may support. Since GSS-API assumes that all primitive names contained within a given internal name refer to the same entity, equal can return "true" if the two names have at least one primitive name in common. If the implementation embodies knowledge of equivalence relationships between names taken from different namespaces, this knowledge may also allow successful comparisons of internal names containing no overlapping primitive elements.

When used in large access control lists, the overhead of creating an GSSName object on each name and invoking the equal method on each name from the ACL may be prohibitive. As an alternative way of supporting this case, GSS-API defines a special form of the

contiguous byte array name which may be compared directly (byte by

byte). Contiguous names suitable for comparison are generated by the export method. Exported names may be re-imported by using the byte array constructor and specifying the GSSNameTypes.NT_EXPORT_NAME as the name type object identifier. The resulting GSSName name will also be a MN. The GSSName interface defines public static Oid objects representing the standard name types. Structurally, an exported name object consists of a header containing an OID identifying the mechanism that authenticated the name, and a trailer containing the name itself, where the syntax of the trailer is defined by the individual mechanism specification. Detailed description of the format is specified in the language-independent GSS-API specification [RFC2743].

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

It is important to note that the above are guidelines as how GSSName implementations should behave, and are not intended to be specific requirements of how names objects must be implemented. The mechanism designers are free to decide on the details of their implementations of the GSSName interface as long as the behavior satisfies the above guidelines.

4.9. Channel Bindings

GSS-API supports the use of user-specified tags to identify a given context to the peer application. These tags are intended to be used to identify the particular communications channel that carries the context. Channel bindings are communicated to the GSS-API using the ChannelBinding object. The application may use byte arrays to specify the application data to be used in the channel binding as well as using instances of the EndPoint class. The EndPoint for the initiator and/or acceptor can be used within an instance of a ChannelBinding. ChannelBinding can be set for the GSSContext object by setting the channelBinding property before the first call to init or accept has been performed. The channelBinding property of a GSSContext object defaults to "null". Currently the ChannelBinding class only supports addresses of type IP. Applications that use other types of addresses can include them as part of the application specific data.

Conceptually, the GSS-API concatenates the initiator and acceptor address information, and the application supplied byte array to form

an octet string. The mechanism calculates a MIC over this octet

string and binds the MIC to the context establishment token emitted by init method of the GSSContext interface. The same bindings are set by the context acceptor for its GSSContext object and during processing of the accept method a MIC is calculated in the same way. The calculated MIC is compared with that found in the token, and if the MICs differ, accept will throw a GSSException with the major code set to BAD_BINDINGS, and the context will not be established. Some mechanisms may include the actual channel binding data in the token (rather than just a MIC); applications should therefore not use confidential data as channel-binding components.

Individual mechanisms may impose additional constraints on addresses that may appear in channel bindings. For example, a mechanism may verify that the initiator address field of the channel binding contains the correct network address of the host system. Portable applications should therefore ensure that they either provide correct information for the address fields, or omit setting of the addressing information.

5. Introduction to GSS-API Classes and Interfaces

This section presents a brief description of the classes and interfaces that constitute the GSS-API.

This section also shows the corresponding RFC 2743 functionality implemented by each of the classes. Detailed description of these classes and their methods is presented in section 6.

5.1. GSSManager Class

This abstract class serves as a factory to instantiate implementations of the GSS-API interfaces and also provides methods to make queries about underlying security mechanisms.

A default implementation can be obtained using the static method getInstance(). Applications that desire to provide their own implementation of the GSSManager class can simply extend the abstract class themselves.

This class contains equivalents of the following RFC 2743 routines:

gss_import_name	Create an internal name from the supplied information.
gss_acquire_cred	Acquire credential for use.
gss_import_sec_context	Create a previously exported context.

gss_indicate_mechs	List the mechanisms supported by this GSS-API implementation.
gss_inquire_mechs_for_name	List the mechanisms supporting the specified name type.
gss_inquire_names_for_mech	List the name types supported by the specified mechanism.

5.2. GSSName Interface

GSS-API names are represented in the C# bindings through the GSSName interface. Different name formats and their definitions are identified with universal Object Identifiers (oids). The format of the names can be derived based on the unique oid of each name type. The following GSS-API routines are provided by the GSSName interface:

RFC 2743 Routine	Function
gss_display_name	Covert internal name representation to text format.
gss_compare_name	Compare two internal names.
gss_canonicalize_name	Convert an internal name to a mechanism name.
gss_export_name	Convert a mechanism name to export format.

The gss_release_name call is not provided as C# does its own garbage collection. The gss_duplicate_name call is also redundant; the GSSName interface has no methods that can change the state of the object so it is safe for sharing.

5.3. GSSCredential Interface

The GSSCredential interface is responsible for the encapsulation of GSS-API credentials. Credentials identify a single entity and provide the necessary cryptographic information to enable the creation of a context on behalf of that entity. A single credential may contain multiple mechanism specific credentials, each referred to as a credential element. The GSSCredential interface provides the functionality of the following GSS-API routines:

RFC 2743 Routine	Function
gss_add_cred	Constructs credentials incrementally.
gss_inquire_cred	Obtain information about credential.

gss_inquire_cred_by_mech	Obtain per-mechanism information about a credential.
gss_release_cred	Disposes of credentials after use.

5.4. GSSContext Interface

This interface encapsulates the functionality of context-level calls required for security context establishment and management between peers as well as the per-message services offered to applications. A context is established between a pair of peers and allows the usage of security services on a per-message basis on application data. It is created over a single security mechanism. The GSSContext interface provides the functionality of the following GSS-API routines:

RFC 2743 Routine	Function
gss_init_sec_context	Initiate the creation of a security context with a peer.
gss_accept_sec_context	Accept a security context initiated by a peer.
gss_delete_sec_context	Destroy a security context.
gss_context_time	Obtain remaining context time.
gss_inquire_context	Obtain context characteristics.
gss_wrap_size_limit	Determine token-size limit for gss_wrap.
gss_export_sec_context	Transfer security context to another process.
gss_get_mic	Calculate a cryptographic Message Integrity Code (MIC) for a message.
gss_verify_mic	Verify integrity on a received message.
gss_wrap	Attach a MIC to a message and optionally encrypt the message content.
gss_unwrap	Obtain a previously wrapped application message verifying its integrity and optionally decrypting it.

The functionality offered by the gss_process_context_token routine has not been included in the C# bindings specification. The

corresponding functionality of gss_delete_sec_context has also been modified to not return any peer tokens. This has been proposed in accordance to the recommendations stated in RFC 2743. GSSContext does offer the functionality of destroying the locally-stored context information.

5.5. MessageProp Class

This helper class is used in the per-message operations on the context. An instance of this class is created by the application and then passed into the per-message calls. In some cases, the application conveys information to the GSS-API implementation through this object and in other cases the GSS-API returns information to the application by setting it in this object. See the description of the per-message operations wrap, unwrap, getMIC, and verifyMIC in the GSSContext interface for details.

5.6. GSSException Class

Exceptions are used in the C# bindings to signal fatal errors to the calling applications. This replaces the major and minor codes used in the C-bindings specification as a method of signaling failures. The GSSException class handles both minor and major codes, as well as their translation into textual representation. All GSS-API methods can throw this exception.

RFC 2743 Routine	Function
gss_display_status	Retrieve textual representation of error codes.

5.7. Oid Class

This utility class is used to represent Universal Object Identifiers and their associated operations. GSS-API uses object identifiers to distinguish between security mechanisms and name types. This class, aside from being used whenever an object identifier is needed, implements the following GSS-API functionality:

RFC 2743 Routine	Function
gss_test_oid_set_member	Determine if the specified oid is part of a set of oids.

5.8. ChannelBinding Class

An instance of this class is used to specify channel binding information to the GSSContext object before the start of a security context establishment. The application may use a byte array to

specify application data to be used in the channel binding as well as use instances of the EndPoint. Currently the ChannelBinding class only supports addresses of type IP. Applications that use other types of addresses can include them as part of the application specific data.

5.9. GSSConstants Class

This utility class defines various constants utilized throughout the API.

5.10. GSSNameTypes Class

This class defines OIDs which specify different types of GSSNames.

5.11. GSSCredentialUsage Enumeration

This enumeration defines the usage categories for GSSCredentials. GSSCredentials can be used for context initiation and context acceptance or only one of those functions.

6. Detailed GSS-API Description

This section lists a detailed description of all the public methods that each of the GSS-API classes and interfaces must provide.

6.1. public abstract class GSSManager

The GSSManager class is an abstract class that serves as a factory for three GSS interfaces: GSSName, GSSCredential, and GSSContext. It also provides methods for applications to determine what mechanisms are available from the GSS implementation and what nametypes these mechanisms support. An instance of the default GSSManager subclass may be obtained through the static method getInstance(), but applications are free to instantiate other subclasses of GSSManager.

All but one method in this class are declared abstract. This means that subclasses have to provide the complete implementation for those methods. The only exception to this is the static method getInstance() which will have platform specific code to return an instance of the default subclass.

Platform providers of GSS are required not to add any constructors to this class, private, public, or protected. This will ensure that all subclasses invoke only the default constructor provided to the base class by the compiler.

6.1.1. Example Code

```
GSSManager mgr = GSSManager.getInstance();

// What mechs are available to us?
Oid[] supportedMechs = mgr.getMechs();

// What name types does this spkm implementation support?
Oid[] nameTypes = mgr.getNamesForMech(spkm1);
```

6.1.2. getInstance

```
public static GSSManager getInstance();
```

Returns the default GSSManager implementation.

Throws GSSException if an error is detected.

6.1.3. getMechs

```
public abstract Oid[] getMechs();
```

Returns an array of Oid objects indicating mechanisms available to GSS-API callers. A "null" value is returned when no mechanism are available (an example of this would be when mechanism are dynamically configured, and currently no mechanisms are installed).

Throws GSSException if an error is detected.

6.1.4. getNamesForMech

```
public abstract Oid[] getNamesForMech(Oid mech);
```

Returns name type Oid's supported by the specified mechanism.

Throws GSSException if an error is detected.

Parameters:

mech	The Oid object for the mechanism to query.

6.1.5. getMechsForName

```
public abstract  Oid[] getMechsForName(Oid nameType);
```

Returns an array of Oid objects corresponding to the mechanisms that support the specific name type. "null" is returned when no mechanisms are found to support the specified name type.

Throws GSSException if an error is detected.

Parameters:

nameType The Oid object for the name type.

6.1.6. createName

```
public abstract GSSName createName(string nameStr,
                                   Oid nameType);
```

Factory method to convert a contiguous string name from the specified namespace to a GSSName object. In general, the GSSName object created will not be an MN; two examples that are exceptions to this are when the namespace type parameter indicates GSSNameTypes.NT_EXPORT_NAME or when the GSS-API implementation is not multi-mechanism.

Throws GSSException if an error is detected.

Parameters:

nameStr The string representing a printable form of the name to create.

nameType The Oid specifying the namespace of the printable name supplied. Note that nameType serves to describe and qualify the interpretation of the input nameStr, it does not necessarily imply a type for the output GSSName implementation. "null" value specifies that a mechanism specific default printable syntax should be assumed by each mechanism that examines nameStr.

6.1.7. createName

```
public abstract GSSName createName(byte[] name,
                                   Oid nameType);
```

Factory method to convert a contiguous byte array containing a name from the specified namespace to a GSSName object. In general, the GSSName object created will not be an MN; two examples that are exceptions to this are when the namespace type parameter indicates GSSNameTypes.NT_EXPORT_NAME or when the GSS-API implementation is not multi-mechanism.

Throws GSSException if an error is detected.

Parameters:

name The byte array containing the name to create.

nameType The Oid specifying the namespace of the name supplied in the byte array. Note that nameType serves to describe and qualify the interpretation of the input name byte array, it does not necessarily imply a type for the output GSSName implementation. "null" value can be used to specify that a mechanism specific default syntax should be assumed by each mechanism that examines the byte array.

6.1.8. createName

```
public abstract GSSName createName(string nameStr,
                                   Oid nameType,
                                   Oid mech);
```

Factory method to convert a contiguous string name from the specified namespace to an GSSName object that is a mechanism name (MN). In other words, this method is a utility that does the equivalent of two steps: the createName described in 6.1.6 and then also the GSSName.canonicalize() described in 6.2.5.

Throws GSSException if an error is detected.

Parameters:

nameStr The string representing a printable form of the name to create.

nameType The Oid specifying the namespace of the printable name supplied. Note that nameType serves to describe and qualify the interpretation of the input nameStr, it does not necessarily imply a type for the output GSSName implementation. "null" value can be used to specify that a mechanism specific default printable syntax should be assumed when the mechanism examines nameStr.

mech Oid specifying the mechanism for which this name should be created.

6.1.9. createName

```
public abstract GSSName createName(byte[] name,
                                   Oid nameType,
                                   Oid mech);
```

Throws GSSException if an error is detected.

Factory method to convert a contiguous byte array containing a name from the specified namespace to a GSSName object that is an MN. In other words, this method is a utility that does the equivalent of two steps: the createName described in 6.1.7 and then also the GSSName.canonicalize() described in 6.2.5.

Parameters:

name	The byte array representing the name to create.
nameType	The Oid specifying the namespace of the name supplied in the byte array. Note that nameType serves to describe and qualify the interpretation of the input name byte array, it does not necessarily imply a type for the output GSSName implementation. "null" value can be used to specify that a mechanism specific default syntax should be assumed by each mechanism that examines the byte array.
mech	Oid specifying the mechanism for which this name should be created.

6.1.10. createCredential

```
public abstract GSSCredential createCredential(int usage);
```

Factory method for acquiring default credentials. This will cause the GSS-API to use system specific defaults for the set of mechanisms, name, and a DEFAULT lifetime.

Throws GSSException if an error is detected.

Parameters:

usage	The intended usage for this credential object. The value of this parameter must be one of: GSSCredential.ACCEPT_AND_INITIATE GSSCredential.ACCEPT_ONLY GSSCredential.INITIATE_ONLY

6.1.11. createCredential

```
public abstract GSSCredential createCredential(GSSName aName,
                                               int lifetime,
                                               Oid mech,
                                               int usage);
```

Throws GSSException if an error is detected.

Factory method for acquiring a single mechanism credential.

Parameters:

aName	Name of the principal for whom this credential is to be acquired. Use "null" to specify the default principal.
lifetime	The number of seconds that credentials should remain valid. Use GSSCredential.INDEFINITE_LIFETIME to request that the credentials have the maximum permitted lifetime. Use GSSCredential.DEFAULT_LIFETIME to request default credential lifetime.
mech	The oid of the desired mechanism. Use "(Oid) null" to request the default mechanism(s).
usage	The intended usage for this credential object. The value of this parameter must be one of: GSSCredential.ACCEPT_AND_INITIATE GSSCredential.ACCEPT_ONLY GSSCredential.INITIATE_ONLY

6.1.12. createCredential

```
public abstract GSSCredential createCredential(GSSName aName,
                                               int lifetime,
                                               Oid mechs[],
                                               int usage);
```

Factory method for acquiring credentials over a set of mechanisms. Acquires credentials for each of the mechanisms specified in the array called mechs. To determine the list of mechanisms for which the acquisition of credentials succeeded, the caller should use the GSSCredential.getMechs() method.

Throws GSSException if an error is detected.

Parameters:

aName — Name of the principal for whom this credential is to be acquired. Use "null" to specify the default principal.

lifetime — The number of seconds that credentials should remain valid. Use GSSCredential.INDEFINITE_LIFETIME to request that the credentials have the maximum permitted lifetime. Use GSSCredential.DEFAULT_LIFETIME to request default credential lifetime.

mechs — The array of mechanisms over which the credential is to be acquired. Use "(Oid[]) null" for requesting a system specific default set of mechanisms.

usage — The intended usage for this credential object. The value of this parameter must be one of:

GSSCredential.ACCEPT_AND_INITIATE
GSSCredential.ACCEPT_ONLY
GSSCredential.INITIATE_ONLY

6.1.13. createContext

```
public abstract GSSContext createContext(GSSName peer,
                                         Oid mech,
                                         GSSCredential myCred,
                                         int lifetime);
```

Factory method for creating a context on the initiator's side. Context flags may be modified through the mutator methods prior to calling GSSContext.initSecContext().

Throws GSSException if an error is detected.

Parameters:

peer — Name of the target peer.

mech — Oid of the desired mechanism. Use "(Oid) null" to request default mechanism.

myCred — Credentials of the initiator. Use "null" to act as a default initiator principal.

lifetime	The request lifetime, in seconds, for the context. Use GSSContext.INDEFINITE_LIFETIME and GSSContext.DEFAULT_LIFETIME to request indefinite or default context lifetime.

6.1.14. createContext

```
public abstract GSSContext createContext(GSSCredential myCred);
```

Factory method for creating a context on the acceptor' side. The context's properties will be determined from the input token supplied to the accept method.

Throws GSSException if an error is detected.

Parameters:

myCred	Credentials for the acceptor. Use "null" to act as a default acceptor principal.

6.1.15. createContext

```
public abstract GSSContext createContext(byte[] interProcessToken);
```

Factory method for creating a previously exported context. The context properties will be determined from the input token and can't be modified through the set methods.

Throws GSSException if an error is detected.

Parameters:

interProcessToken	The token previously emitted from the export method.

6.2. public class GSSConstants

This class defines constants that are common among various interfaces and/or classes of the API.

6.2.1. DEFAULT_LIFETIME

```
const int DEFAULT_LIFETIME = 0;
```

A constant representing the default lifetime for a context or credential.

6.2.2. INDEFINITE_LIFETIME

```
const int INDEFINITE_LIFETIME = Int32.MaxValue;
```

A constant representing indefinite lifetime for a context or credential.

6.3. public class GSSNameTypes

This class defines the various types of GSSNames.

6.3.1. NT_HOSTBASED_SERVICE

```
public static Oid NT_HOSTBASED_SERVICE;
```

Property which indicates Oid of name type of a host-based service name form. It is used to represent services associated with host computers. This name form is constructed using two elements, "service" and "hostname", as follows:

```
service@hostname
```

Values for the "service" element are registered with the IANA. It represents the following value: { 1(iso), 3(org), 6(dod), 1(internet), 5(security), 6(nametypes), 2(gss-host-based-services) }

6.3.2. NT_USER_NAME

```
public static Oid NT_USER_NAME;
```

Property which indicates Oid of name type of a named user on a local system. It represents the following value: { 1(iso), 2(member-body), 840(United States), 113554(mit), 1(infosys), 2(gssapi), 1(generic), 1(user_name) }

6.3.3. NT_MACHINE_UID_NAME

```
public static Oid NT_MACHINE_UID_NAME;
```

Property which indicates Oid of name type of a numeric user identifier corresponding to a user on a local system. (e.g. Uid). It represents the following value: { 1(iso), 2(member-body), 840(United States), 113554(mit), 1(infosys), 2(gssapi), 1(generic), 2(machine_uid_name) }

6.3.4. NT_STRING_UID_NAME

```
public static Oid NT_STRING_UID_NAME;
```

Property which indicates Oid of name type of a string of digits representing the numeric user identifier of a user on a local system. It represents the following value: { 1(iso), 2(member-body), 840(United States), 113554(mit), 1(infosys), 2(gssapi), 1(generic), 3(string_uid_name) }

6.3.5. NT_ANONYMOUS

```
public static Oid NT_ANONYMOUS;
```

Property which indicates Oid of name type of an anonymous entity. It represents the following value: { 1(iso), 3(org), 6(dod), 1(internet), 5(security), 6(nametypes), 3(gss-anonymous-name) }

6.3.6. NT_EXPORT_NAME

```
public static Oid NT_EXPORT_NAME;
```

Property which indicates Oid of name type of an exported name produced by the export method. It represents the following value: { 1(iso), 3(org), 6(dod), 1(internet), 5(security), 6(nametypes), 4(gss-api-exported-name) }

6.4. public interface GSSName

This interface encapsulates a single GSS-API principal entity. Different name formats and their definitions are identified with universal Object Identifiers (Oids). The format of the names can be derived based on the unique oid of its namespace type.

6.4.1. Example Code

Included below are code examples utilizing the GSSName interface. The code below creates a GSSName, converts it to a mechanism name (MN), performs a comparison, obtains a printable representation of the name, exports it and then re-imports to obtain a new GSSName.

```
GSSManager mgr = GSSManager.getInstance();

// create a host based service name
GSSName name = mgr.createName("service@host",
                              GSSNameTypes.NT_HOSTBASED_SERVICE);

Oid krb5 = new Oid("1.2.840.113554.1.2.2");

GSSName mechName = name.canonicalize(krb5);

// the above two steps are equivalent to the following
GSSName mechName = mgr.createName("service@host",
                                  GSSNameTypes.NT_HOSTBASED_SERVICE,
                                  krb5);

// perform name comparison
if (name.Equals(mechName))
   Console.WriteLine("Names are equals.");

// obtain textual representation of name and its printable
// name type
Console.WriteLine(mechName.ToString() +
                  mechName.getStringNameType().ToString());

// export and re-import the name
byte[] exportName = mechName.export();

// create a new name object from the exported buffer
GSSName newName = mgr.createName(exportName,
                                 GSSNameTypes.NT_EXPORT_NAME);
```

6.4.2. Equals

```
bool Equals(GSSName another);
```

Compares two GSSName objects to determine whether they refer to the same entity. If either of the names represents an anonymous entity, the method will return "false".

Throws GSSException if an error is detected.

Parameters:

another	GSSName object to compare with.

6.4.3. Equals

```
bool Equals(Object another);
```

A variation of the equals method described in 6.2.3 that is provided to override the Object.Equals() method that the implementing class will inherit. The behavior is exactly the same as that in 6.2.3 except that no GSSException is thrown; instead, false will be returned in the situation where an error occurs. (Note that the C# language specification requires that two objects that are equal according to the Equals(Object) method must return the same integer result when the GetHashCode() method is called on them.)

Parameters:

another	GSSName object to compare with.

6.4.4. canonicalize

```
GSSName canonicalize(Oid mech);
```

Creates a mechanism name (MN) from an arbitrary internal name. This is equivalent to using the factory methods described in 6.1.8 or 6.1.9 that take the mechanism name as one of their parameters.

Throws GSSException if an error is detected.

Parameters:

mech	The oid for the mechanism for which the canonical form of the name is requested.

6.4.5. export

```
byte[] export();
```

Returns a canonical contiguous byte representation of a mechanism name (MN), suitable for direct, byte by byte comparison by authorization functions. If the name is not an MN, implementations may throw a GSSException with the NAME_NOT_MN status code. If an implementation chooses not to throw an exception, it should use some system specific default mechanism to canonicalize the name and then export it. The format of the header of the output buffer is specified in RFC 2743.

6.4.6. ToString

```
string ToString();
```

Returns a textual representation of the GSSName object. To retrieve the printed name format, which determines the syntax of the returned string, the getStringNameType method can be used.

Throws GSSException if an error is detected.

6.4.7. stringNameType

```
Oid stringNameType;
```

Property of the object containing the oid representing the type of name returned through the ToString method. Using this oid, the syntax of the printable name can be determined.

Throws GSSException if an error is detected.

6.4.8. isAnonymous

```
bool isAnonymous;
```

Property which indicates whether the name object represents an anonymous entity or not. If "true" then it is an anonymous name.

6.4.9. isMN

```
bool isMN;
```

Property which indicates whether the name object contains only one mechanism element and is a mechanism name as defined by RFC 2743.

6.5. public enum GSSCredentialUsage

This enumeration defines the usage categories for credentials.

6.5.1. INITIATE_AND_ACCEPT

Credentials of this type can be used for both context initiation and acceptance.

6.5.2. INITIATE_ONLY

Credentials of this type can be used for context initiation only.

6.5.3. ACCEPT_ONLY

Credentials of this type can be used for context acceptance only.

6.6. public interface GSSCredential: ICloneable

This interface encapsulates the GSS-API credentials for an entity. A credential contains all the necessary cryptographic information to enable the creation of a context on behalf of the entity that it represents. It may contain multiple, distinct, mechanism specific credential elements, each containing information for a specific security mechanism, but all referring to the same entity.

A credential may be used to perform context initiation, acceptance, or both.

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

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

Applications will create a credential object passing the desired parameters. The application can then use the query methods to obtain specific information about the instantiated credential object (equivalent to the gss_inquire routines). When the credential is no longer needed, the application should call the dispose (equivalent to gss_release_cred) method to release any resources held by the credential object and to destroy any cryptographically sensitive information.

Classes implementing this interface also implement the ICloneable interface. This indicates that the class will support the Clone() method that will allow the creation of duplicate credentials. This is useful when called just before the add() call to retain a copy of the original credential.

6.6.1. Example Code

This example code demonstrates the creation of a GSSCredential implementation for a specific entity, querying of its fields, and its release when it is no longer needed.

```
GSSManager mgr = GSSManager.getInstance();

// start by creating a name object for the entity
GSSName name = mgr.createName("userName", GSSNameTypes.NT_USER_NAME);

// now acquire credentials for the entity
GSSCredential cred = mgr.createCredential(name,
                                    GSSCredentialUsage.ACCEPT_ONLY);

// display credential information - name, remaining lifetime,
// and the mechanisms it has been acquired over
Console.WriteLine(cred.getName().ToString());
Console.WriteLine(cred.getRemainingLifetime());

Oid[] mechs = cred.getMechs();
if (mechs != null) {
   for (int i = 0; i < mechs.length; i++)
      Console.WriteLine(mechs[i].ToString());
}

// release system resources held by the credential
cred.dispose();
```

6.6.2. dispose

```
void dispose();
```

Releases any sensitive information that the GSSCredential object may be containing. Applications should call this method as soon as the credential is no longer needed to minimize the time any sensitive information is maintained.

Throws GSSException if an error is detected.

6.6.3. getName

```
GSSName getName();
```

Retrieves the name of the entity that the credential asserts.

Throws GSSException if an error is detected.

6.6.4. getName

```
GSSName getName(Oid mechOID);
```

Retrieves a mechanism name of the entity that the credential asserts.

Throws GSSException if an error is detected.

Parameters:

mechOID	The mechanism for which information should be returned.

6.6.5. getRemainingLifetime

```
int getRemainingLifetime();
```

Returns the remaining lifetime in seconds for a credential. The remaining lifetime is the minimum lifetime for any of the underlying credential mechanisms. A return value of GSSConstants.INDEFINITE_LIFETIME indicates that the credential does not expire. A return value of 0 indicates that the credential is already expired.

Throws GSSException if an error is detected.

6.6.6. getRemainingInitLifetime

```
int getRemainingInitLifetime(Oid mech);
```

Returns the remaining lifetime is seconds for the credential to remain capable of initiating security contexts under the specified mechanism. A return value of GSSConstants.INDEFINITE_LIFETIME indicates that the credential does not expire for context initiation. A return value of 0 indicates that the credential is already expired.

Throws GSSException if an error is detected.

Parameters:

mechOID	The mechanism for which information should be returned.

6.6.7. getRemainingAcceptLifetime

```
int getRemainingAcceptLifetime(Oid mech);
```

Returns the remaining lifetime is seconds for the credential to remain capable of accepting security contexts under the specified mechanism. A return value of GSSConstants.INDEFINITE_LIFETIME indicates that the credential does not expire for context acceptance. A return value of 0 indicates that the credential is already expired.

Throws GSSException if an error is detected.

Parameters:

mechOID	The mechanism for which information should be returned.

6.6.8. getUsage

```
GSSCredentialUsage getUsage();
```

Returns the usage category for the credential.

Throws GSSException if an error is detected.

6.6.9. getUsage

```
GSSCredentialUsage getUsage(Oid mechOID);
```

Returns the usage category for the specified credential mechanism.

Throws GSSException if an error is detected.

Parameters:

mechOID	The mechanism for which information should be returned.

6.6.10. getMechs

```
Oid[] getMechs();
```

Returns an array of mechanisms supported by this credential.

Throws GSSException if an error is detected.

6.6.11. add

```
void add(GSSName aName,
         int initLifetime,
         int acceptLifetime,
         Oid mech,
         int usage);
```

Adds a mechanism specific credential-element to an existing credential. This method allows the construction of credentials one mechanism at a time.

This routine is envisioned to be used mainly by context acceptors during the creation of acceptance credentials which are to be used with a variety of clients using different security mechanisms.

This routine adds the new credential element "in-place". To add the element in a new credential, first call Clone() to obtain a copy of this credential, then call its add() method.

Throws GSSException if an error is detected.

Parameters:

aName	Name of the principal for whom this credential is to be acquired. Use "null" to specify the default principal.

initLifetime
The number of seconds that credentials should remain valid for initiating of security contexts. Use GSSCredential.INDEFINITE_LIFETIME to request that the credentials have the maximum permitted lifetime. Use GSSCredential.DEFAULT_LIFETIME to request default credential lifetime.

acceptLifetime
The number of seconds that credentials should remain valid for accepting of security contexts. Use GSSCredential.INDEFINITE_LIFETIME to request that the credentials have the maximum permitted lifetime. Use GSSCredential.DEFAULT_LIFETIME to request default credential lifetime.

mech The mechanisms over which the credential is to be acquired.

usage The intended usage for this credential object. The value of this parameter must be one of: GSSCredential.ACCEPT_AND_INITIATE, GSSCredential.ACCEPT_ONLY, GSSCredential.INITIATE_ONLY

6.6.12. Equals

```
bool Equals(Object another);
```

Tests if this GSSCredential refers to the same entity as the supplied object. The two credentials must be acquired over the same mechanisms and must refer to the same principal. Returns "true" if the two GSSCredentials refer to the same entity; "false" otherwise.

Parameters:

another Another GSSCredential object for comparison.

6.7. public interface GSSContext

This interface encapsulates the GSS-API security context and provides the security services (wrap, unwrap, getMIC, verifyMIC) that are available over the context. Security contexts are established between peers using locally acquired credentials. Multiple contexts may exist simultaneously between a pair of peers, using the same or different set of credentials. GSS-API functions in a manner independent of the underlying transport protocol and depends on its calling application to transport its tokens between peers.

Before the context establishment phase is initiated, the context initiator may request specific characteristics desired of the established context. These can be set by manipulating the GSSContext properties. After the context is established, the caller can check the actual characteristic and services offered by the context by examining the GSSContext properties.

The context establishment phase begins with the first call to the init method by the context initiator. During this phase the initSecContext and acceptSecContext methods will produce GSS-API authentication tokens which the calling application needs to send to its peer. If an error occurs at any point, an exception will get thrown and the code will start executing in a catch block. If not, the normal flow of code continues and the application can read the isEstablished property. If this property is false it indicates that a token is needed from its peer in order to continue the context establishment phase. A setting of true signals that the local end of the context is established. This may still require that a token be sent to the peer, if one is produced by GSS-API. During the context establishment phase, the isProtReady property indicates whether or not the context can be used for the per-message operations. This allows applications to use per-message operations on contexts which aren't fully established.

After the context has been established or the isProtReady property is "true", the query routines can be invoked to determine the actual characteristics and services of the established context. The application can also start using the per-message methods of wrap and getMIC to obtain cryptographic operations on application supplied data.

When the context is no longer needed, the application should call dispose to release any system resources the context may be using.

6.7.1. Example Code

The example code presented below demonstrates the usage of the GSSContext interface for the initiating peer. Different operations on the GSSContext object are presented, including: object instantiation, setting of desired flags, context establishment, query of actual context flags, per-message operations on application data, and finally context deletion.

```
GSSManager mgr = GSSManager.getInstance();

// start by creating the name for a service entity
GSSName targetName = mgr.createName("service@host",
                                    GSSNameTypes.NT_HOSTBASED_SERVICE);

// create a context using default credentials for the above entity
// and the implementation specific default mechanism
GSSContext context = mgr.createContext(targetName,
                                    null, /* default mechanism */
                                    null, /* default credentials */
                                    GSSConstants.INDEFINITE_LIFETIME);

// set desired context options - all others are false by default
context.confidentiality = true;
context.mutualAuthentication = true;
context.replayDetection = true;
context.sequenceDetection = true;

// establish a context between peers - using byte arrays
byte[] inTok = new byte[0];

try {
   do {
      byte[] outTok = context.initSecContext(inTok,
                                             0,
                                             inTok.length);

      // send the token if present
      if (outTok != null)
         sendToken(outTok);

      // check if we should expect more tokens
      if (context.isEstablished)
         break;

      // another token expected from peer
      inTok = readToken();

   } while (true);
```

```
   } catch (GSSException e) {
      Console.Writeline("GSSAPI error: " + e.Message);
   }

   // display context information
   Console.Writeline("Remaining lifetime in seconds = " +
                     context.lifetime);
   Console.Writeline("Context mechanism = " + context.Mech.ToString());
   Console.Writeline("Initiator = " + context.srcName.ToString());
   Console.Writeline("Acceptor = " + context.targName.ToString());

   if (context.confidentiality)
      Console.Writeline("Confidentiality security service available");

   if (context.integrity)
      Console.Writeline("Integrity security service available");

   // perform wrap on an application supplied message, appMsg,
   // using QOP = 0, and requesting privacy service
   byte[] appMsg ...

   MessageProp mProp = new MessageProp(0, true);

   byte[] tok = context.wrap(appMsg, 0, appMsg.length, mProp);

   if (mProp.privacy)
      Console.Writeline("Message protected with privacy.");

   sendToken(tok);

   // release the local-end of the context
   context.dispose();
```

6.7.2. initSecContext

```
byte[] initSecContext(byte[] inputBuf,
                      int offset,
                      int len);
```

Called by the context initiator to start the context creation process. This is equivalent to the stream based method except that the token buffers are handled as byte arrays instead of using stream objects. This method may return an output token which the application will need to send to the peer for processing by the accept call. The application can check the isEstablished property to determine if the context establishment phase is complete for this peer. A value of "false" for isEstablished indicates that more tokens are expected to be supplied to the initSecContext method. Note that it is possible for the initSecContext() method to return a token for the peer when isEstablished is set to "true". This indicates that the token needs to be sent to the peer, but the local end of the context is now fully established.

Throws GSSException if an error is detected.

Parameters:

inputBuf	Token generated by the peer. This parameter is ignored on the first call.
offset	The offset within the inputBuf where the token begins.
len	The length of the token within the inputBuf (starting at the offset).

6.7.2.1. Example Code

```
// Create a new GSSContext implementation object.
// GSSContext wrapper implements interface GSSContext.
GSSContext context = mgr.createContext(...);

byte[] inTok = new byte[0];

try {
   do {
      byte[] outTok = context.initSecContext(inTok,
                                             0,
                                             inTok.length);

      // send the token if present
      if (outTok != null)
         sendToken(outTok);

      // check if we should expect more tokens
      if (context.isEstablished)
         break;

      // another token expected from peer
      inTok = readToken();

   } while (true);

} catch (GSSException e) {
   Console.Writeline("GSSAPI error: " + e.Message);
}
```

6.7.3. initSecContext

```
void initSecContext(Stream inStream,
                    Stream outStream);
```

Called by the context initiator to start the context creation process. This is equivalent to the byte array based method. This method may write an output token to the outStream, which the application will need to send to the peer for processing by the accept call. The application can check the isEstablished property to determine if the context establishment phase is complete for this peer. A value of "false" for isEstablished indicates that more tokens are expected to be supplied to the initSecContext method. Note that it is possible that the initSecContext() method to return a token for the peer when isEstablished is set to "true". This indicates that the token needs to be sent to the peer, but the local end of the context is now fully established.

The GSS-API authentication tokens contain a definitive start and end. This method will attempt to read one of these tokens per invocation, and may block on the stream if only part of the token is available.

Throws GSSException if an error is detected.

Parameters:

inStream Stream to read the token generated by the peer. This parameter is ignored on the first call.

outStream Stream where the output token will be written. During the final stage of context establishment, there may be no bytes written.

6.7.3.1. Example Code

```
// Create a new GSSContext implementation object.

// GSSContext wrapper implements interface GSSContext.
GSSContext context = mgr.createContext(...);

MemoryStream os = new MemoryStream();
Stream is = null;

try {
   do {
      context.initSecContext(is, os);

      // send token if present
      if (os.Length > 0)
         sendToken(os);

      // check if we should expect more tokens
      if (context.isEstablished)
         break;

      // another token expected from peer
      is = recvToken();

   } while (true);

} catch (GSSException e) {
   Console.Writeline("GSSAPI error: " + e.Message);
}
```

6.7.4. acceptSecContext

```
byte[] acceptSecContext(byte[] inTok,
                        int offset,
                        int len);
```

Called by the context acceptor upon receiving a token from the peer. This call is equivalent to the stream based method except that the token buffers are handled as byte arrays instead of using stream objects.

This method may return an output token which the application will need to send to the peer for further processing by the init call.

A "null" return value indicates that no token needs to be sent to the peer. The application can check the isEstablished property to determine if the context establishment phase is complete for this peer. A value of "false" for isEstablished indicates that more tokens are expected to be supplied to this method. Note that it is possible that the acceptSecContext() method to return a token for the peer when isEstablished is set to "true". This indicates that the token needs to be sent to the peer, but the local end of the context is now fully established.

Upon completion of the context establishment, the available context options may be queried through the get methods.

Throws GSSException if an error is detected.

Parameters:

inTok	Token generated by the peer.
offset	The offset within the inTok where the token begins.
len	The length of the token within the inTok (starting at the offset).

6.7.4.1. Example Code

```
// acquire server credentials
GSSCredential server = mgr.createCredential(...);

// create acceptor GSS-API context from the default provider
GSSContext context = mgr.createContext(server, null);

try {
   do {
      byte[] inTok = readToken();

      byte[] outTok = context.acceptSecContext(inTok,
                                               0,
                                               inTok.length);

      // possibly send token to peer
      if (outTok != null)
         sendToken(outTok);

      // check if local context establishment is complete
      if (context.isEstablished)
         break;
   } while (true);

} catch (GSSException e) {
   Console.Writeline("GSS-API error: " + e.Message);
}
```

6.7.5. acceptSecContext

```
void acceptSecContext(Stream inStream,
                      Stream outStream);
```

Called by the context acceptor upon receiving a token from the peer. This call is equivalent to the byte array method. It may write an output token to the outStream, which the application will need to send to the peer for processing by its initSecContext method. The application can check the isEstablished property to determine if the context establishment phase is complete for this peer. A value of "false" for isEstablished indicates that more tokens are expected to be supplied to this method. Note that it is possible for the acceptSecContext() method to return a token for the peer when isEstablished is set to "true". This indicates that the token needs to be sent to the peer, but the local end of the context is now fully established.

The GSS-API authentication tokens contain a definitive start and end. This method will attempt to read one of these tokens per invocation, and may block on the stream if only part of the token is available.

Throws GSSException if an error is detected.

Parameters:

inStream Contains the token generated by the peer.

outStream Stream where the output token will be written. During the final stage of context establishment, there may be no bytes written.

6.7.5.1. Example Code

```
// acquire server credentials
GSSCredential server = mgr.createCredential(...);

// create acceptor GSS-API context from the default provider
GSSContext context = mgr.createContext(server, null);

MemoryStream os = new MemoryStream();
Stream is = null;

try {
   do {
      is = recvToken();

      context.acceptSecContext(is, os);

      // possibly send token to peer
      if (os.Length > 0)
         sendToken(os);

      // check if local context establishment is complete
      if (context.isEstablished)
         break;
   } while (true);

} catch (GSSException e) {
   Console.Writeline("GSS-API error: " + e.Message);
}
```

6.7.6. isEstablished

```
bool isEstablished;
```

Property which indicates the state of the context. A setting of "true" for the property indicates that the context has been fully established on the caller's side and no more tokens are needed from the peer. The property should be examined after calls to initSecContext() or acceptSecContext() when no GSSException is thrown.

6.7.7. dispose

```
void dispose();
```

Releases any system resources and cryptographic information stored in the context object. This will invalidate the context.

Throws GSSException if an error is detected.

6.7.8. getWrapSizeLimit

```
int getWrapSizeLimit(int qop,
                     bool confReq,
                     int maxTokenSize);
```

Returns the maximum message size that, if presented to the wrap method with the same confReq and qop parameters, will result in an output token containing no more than the maxTokenSize bytes.

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

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

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

Throws GSSException if an error is detected.

Parameters:

qop Indicates the level of protection wrap will be asked to provide.

confReq Indicates if wrap will be asked to provide privacy service.

maxTokenSize
The desired maximum size of the token emitted by wrap.

6.7.9. wrap

```
byte[] wrap(byte[] inBuf,
            int offset,
            int len,
            MessageProp msgProp);
```

Applies per-message security services over the established security context. The method will return a token with a cryptographic MIC and may optionally encrypt the specified inBuf. This method is equivalent in functionality to its stream counterpart. The returned byte array will contain both the MIC and the message.

The MessageProp object is instantiated by the application and used to specify a QOP value which selects cryptographic algorithms, and a privacy service to optionally encrypt the message. The underlying mechanism that is used in the call may not be able to provide the privacy service. It sets the actual privacy service that it does provide in this MessageProp object which the caller should then query upon return. If the mechanism is not able to provide the requested QOP, it throws a GSSException with the BAD_QOP code.

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

The application will be responsible for sending the token to the peer.

Throws GSSException if an error is detected.

Parameters:

inBuf	Application data to be protected.
offset	The offset within the inBuf where the data begins.
len	The length of the data within the inBuf (starting at the offset).
msgProp	Instance of MessageProp that is used by the application to set the desired QOP and privacy state. Set the desired QOP to 0 to request the default QOP. Upon return from this method, this object will contain the actual privacy state that was applied to the message by the underlying mechanism.

6.7.10. wrap

```
void wrap(Stream inStream,
          Stream outStream,
          MessageProp msgProp);
```

Applies per-message security services over the established security context. The method will produce a token with a cryptographic MIC and may optionally encrypt the message in inStream. The outStream will contain both the MIC and the message.

The MessageProp object is instantiated by the application and used to specify a QOP value which selects cryptographic algorithms, and a privacy service to optionally encrypt the message. The underlying mechanism that is used in the call may not be able to provide the privacy service. It sets the actual privacy service that it does provide in this MessageProp object which the caller should then query upon return. If the mechanism is not able to provide the requested QOP, it throws a GSSException with the BAD_QOP code.

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

The application will be responsible for sending the token to the peer.

Throws GSSException if an error is detected.

Parameters:

inStream	Stream containing the application data to be protected.
outStream	The stream to write the protected message to. The application is responsible for sending this to the other peer for processing in its unwrap method.
msgProp	Instance of MessageProp that is used by the application to set the desired QOP and privacy state. Set the desired QOP to 0 to request the default QOP. Upon return from this method, this object will contain the the actual privacy state that was applied to the message by the underlying mechanism.

6.7.11. unwrap

```
byte[] unwrap(byte[] inBuf,
              int offset,
              int len,
              MessageProp msgProp);
```

Used by the peer application to process tokens generated with the wrap call. This call is equal in functionality to its stream counterpart. The method will return the message supplied in the peer application to the wrap call, verifying the embedded MIC.

The MessageProp object is instantiated by the application and is used by the underlying mechanism to return information to the caller such as the QOP, whether confidentiality was applied to the message, and other supplementary message state information.

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

Throws GSSException if an error is detected.

Parameters:

inBuf	GSS-API wrap token received from peer.
offset	The offset within the inBuf where the token begins.
len	The length of the token within the inBuf (starting at the offset).
msgProp	Upon return from the method, this object will contain the applied QOP, the privacy state of the message, and supplementary information stating whether the token was a duplicate, old, out of sequence or arriving after a gap.

6.7.12. unwrap

```
void unwrap(Stream inStream,
            Stream outStream,
            MessageProp msgProp);
```

Used by the peer application to process tokens generated with the wrap call. This call is equal in functionality to its byte array counterpart. It will produce the message supplied in the peer application to the wrap call, verifying the embedded MIC.

The MessageProp object is instantiated by the application and is used by the underlying mechanism to return information to the caller such as the QOP, whether confidentiality was applied to the message, and other supplementary message state information.

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

Throws GSSException if an error is detected.

Parameters:

inStream	Stream containing the GSS-API wrap token received from the peer.
outStream	The stream to write the application message to.
msgProp	Upon return from the method, this object will contain the applied QOP, the privacy state of the message, and supplementary information stating whether the token was a duplicate, old, out of sequence or arriving after a gap.

6.7.13. getMIC

```
byte[] getMIC(byte[] inMsg,
              int offset,
              int len,
              MessageProp msgProp);
```

Produces a token containing a cryptographic MIC for the supplied message, for transfer to the peer application. Unlike wrap, which encapsulates the user message in the returned token, only the message MIC is returned in the output token. This method is identical in functionality to its stream counterpart.

Note that privacy can only be applied through the wrap call.

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

Throws GSSException if an error is detected.

Parameters:

inMsg	Message to generate MIC over.
offset	The offset within the inMsg where the token begins.
len	The length of the token within the inMsg (starting at the offset).
msgProp	Instance of MessageProp that is used by the application to set the desired QOP. Set the desired QOP to 0 in msgProp to request the default QOP. Alternatively pass in "null" for msgProp to request default QOP.

6.7.14. getMIC

```
void getMIC(Stream inStream,
            Stream outStream,
            MessageProp msgProp);
```

Produces a token containing a cryptographic MIC for the supplied message, for transfer to the peer application. Unlike wrap, which encapsulates the user message in the returned token, only the message MIC is produced in the output token. This method is identical in functionality to its byte array counterpart.

Note that privacy can only be applied through the wrap call.

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

Throws GSSException if an error is detected.

Parameters:

inStream	Stream containing the message to generate MIC over.
outStream	Stream to write the GSS-API output token to.
msgProp	Instance of MessageProp that is used by the application to set the desired QOP. Set the desired QOP to 0 in msgProp to request the default QOP. Alternatively pass in "null" for msgProp to request default QOP.

6.7.15. verifyMIC

```
void verifyMIC(byte[] inTok,
               int tokOffset,
               int tokLen,
               byte[] inMsg,
               int msgOffset,
               int msgLen,
               MessageProp msgProp);
```

Verifies the cryptographic MIC, contained in the token parameter, over the supplied message. This method is equivalent in functionality to its stream counterpart.

The MessageProp object is instantiated by the application and is used by the underlying mechanism to return information to the caller such as the QOP indicating the strength of protection that was applied to the message and other supplementary message state information.

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

Throws GSSException if an error is detected.

Parameters:

inTok	Token generated by peer's getMIC method.
tokOffset	The offset within the inTok where the token begins.
tokLen	The length of the token within the inTok (starting at the offset).
inMsg	Application message to verify the cryptographic MIC over.
msgOffset	The offset within the inMsg where the message begins.
msgLen	The length of the message within the inMsg (starting at the offset).
msgProp	Upon return from the method, this object will contain the applied QOP and supplementary information stating whether the token was a duplicate, old, out of sequence or arriving after a gap. The confidentiality state will be set to "false".

6.7.16. verifyMIC

```
void verifyMIC(Stream tokStream,
               Stream msgStream,
               MessageProp msgProp);
```

Verifies the cryptographic MIC, contained in the token parameter, over the supplied message. This method is equivalent in functionality to its byte array counterpart.

The MessageProp object is instantiated by the application and is used by the underlying mechanism to return information to the caller such as the QOP indicating the strength of protection that was applied to the message and other supplementary message state information.

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

Throws GSSException if an error is detected.

Parameters:

tokStream	Stream containing the token generated by peer's getMIC method.
msgStream	Stream containing the application message to verify the cryptographic MIC over.
msgProp	Upon return from the method, this object will contain the applied QOP and supplementary information stating whether the token was a duplicate, old, out of sequence or arriving after a gap. The confidentiality state will be set to "false".

6.7.17. export

```
byte[] export();
```

Provided to support the sharing of work between multiple processes. This routine will typically be used by the context-acceptor, in an application where a single process receives incoming connection requests and accepts security contexts over them, then passes the established context to one or more other processes for message exchange.

This method deactivates the security context and creates an interprocess token which, when passed to the byte array constructor of the GSSContext interface in another process, will re-activate the context in the second process. Only a single instantiation of a given context may be active at any one time; a subsequent attempt by a context exporter to access the exported security context will fail.

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

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

Throws GSSException if an error is detected.

6.7.18. mutualAuthentication

```
bool mutualAuthentication;
```

Mutual Authentication context property. When the value of this property is set to "true" before the context creation process begins then it indicates to the underlying mechanisms that mutual authentication should be requested during context establishment. The value of this property after context establishment is completed indicates whether or not mutual authentication was performed when the context was established.

Throws GSSException if an error is detected.

6.7.19. replayDetection

```
bool replayDetection;
```

Replay Detection context property. When the value of this property is set to "true" before the context creation process begins then it indicates to the underlying mechanisms that the replay detection service should be requested during context establishment. The value of this property after context establishment is completed or after the value of the isProtReady property becomes "true" indicates whether or not the replay detection service is in effect for the context.

Throws GSSException if an error is detected.

6.7.20. sequenceDetection

```
bool sequenceDetection;
```

Sequence Detection context property. When the value of this property is set to "true" before the context creation process begins then it indicates to the underlying mechanisms that the sequence checking service should be requested during context establishment. The value of this property after context establishment is completed or after the value of the isProtReady property becomes "true" indicates whether or not the sequence checking service is in effect for the context.

Throws GSSException if an error is detected.

6.7.21. credentialDelegation

```
bool credentialDelegation;
```

Credential Delegate context property. When the value of this property is set to "true" before the context creation process begins then it indicates to the underlying mechanisms that credential delegation is desired. The value of this property after context establishment is completed or after the value of the isProtReady property becomes "true" indicates whether or not credential delegation was performed when the context was established.

Throws GSSException if an error is detected.

6.7.22. anonymity

```
bool anonymity;
```

Anonimity context property. When the value of this property is set to "true" before the context creation process begins then it indicates to the underlying mechanisms that anonymity has been requested. The value of this property after context establishment is completed or after the value of the isProtReady property becomes "true" indicates whether or not anonymity was used during the establishment of the context.

Throws GSSException if an error is detected.

6.7.23. confidentiality

```
bool confidentiality;
```

Confidentiality context property. When the value of this property is set to "true" before the context creation process begins then it indicates to the underlying mechanisms that the confidentiality service should be requested during context establishment. The value of this property after context establishment is completed or after the value of the isProtReady property becomes "true" indicates whether or not the confidentiality service is in effect for the context.

Throws GSSException if an error is detected.

6.7.24. integrity

```
bool integrity;
```

Integrity context property. When the value of this property is set to "true" before the context creation process begins then it indicates to the underlying mechanisms that the integrity service should be requested during context establishment. The value of this property after context establishment is completed or after the value of the isProtReady property becomes "true" indicates whether or not the integrity service is in effect for the context.

Throws GSSException if an error is detected.

6.7.25. lifetime

```
int lifetime;
```

Lifetime context property. The value of this property indicates the lifetime in seconds for the context. This property can only be set by the initiator before the context creation process is started. Set the property to GSSConstants.INDEFINITE_LIFETIME and GSSConstants.DEFAULT_LIFETIME to request indefinite or default context lifetime. The value of this property after context establishment is completed or after the value of the isProtReady property becomes "true" indicates the remaining lifetime for the context.

Throws GSSException if an error is detected.

6.7.26. channelBinding

```
ChannelBinding channelBinding;
```

Channel Binding context property. This value of this property can be set to specify a Channel Binding to be used during context establishment. This property can only be set before the context creation process begins.

Throws GSSException if an error is detected.

6.7.27. isTransferable

```
bool isTransferable;
```

Property which indicates whether or not the context can be transferred to other processes. The value of this property is only valid on fully established contexts.

Throws GSSException if an error is detected.

6.7.28. isProtReady

```
bool isProtReady;
```

Property which indicates whether or not per message operations can be applied over the context. Some mechanisms may allow the usage of per-message operations before the context is fully established.

Throws GSSException if an error is detected.

6.7.29. srcName

GSSName srcName;

Property which contains the name of the context initiator. The value of this property is valid only after the context is fully established or the isProtReady property is set to "true". The property is guaranteed to be set to an MN after it becomes valid.

Throws GSSException if an error is detected.

6.7.30. targName

GSSName targName;

Property which contains the name of the context target (acceptor). The value of this property is valid only after the context is fully established or the isProtReady property is set to "true". The property is guaranteed to be set to an MN after it becomes valid.

Throws GSSException if an error is detected.

6.7.31. mechanism

Oid mechanism;

Property which contains the mechanism oid for the context. The value of this property may change from time to time if mechanism negotiation takes place.

Throws GSSException if an error is detected.

6.7.32. delegatedCredential

GSSCredential delegatedCredential;

Property which contains the credential delegated by the context initiator. The value of this property is only valid on the acceptors side after the context has been fully established and if the credentialDelegation property is set to "true".

Throws GSSException if an error is detected.

6.7.33. isInitiator

```
bool isInitiator;
```

Property which contains indication of whether or not the context creation process started on this side. The value of this property is only valid after the context creation process has started.

Throws GSSException if an error is detected.

6.8. public class MessageProp

This is a utility class used within the per-message GSSContext methods to convey per-message properties.

When used with the GSSContext interface's wrap and getMIC methods, an instance of this class is used to indicate the desired QOP and to request if confidentiality services are to be applied to caller supplied data (wrap only). To request default QOP, the value of 0 should be used for QOP.

When used with the unwrap and verifyMIC methods of the GSSContext interface, an instance of this class will be used to indicate the applied QOP and confidentiality services over the supplied message. In the case of verifyMIC, the confidentiality state will always be "false". Upon return from these methods, this object will also contain any supplementary status values applicable to the processed token. The supplementary status values can indicate old tokens, out of sequence tokens, gap tokens or duplicate tokens.

6.8.1. Constructors

```
public MessageProp(bool privState);
```

Constructor which sets QOP to 0 indicating that the default QOP is requested.

Throws GSSException if an error is detected.

Parameters:

privState	The desired privacy state. "true" for privacy and "false" for integrity only.

```
public MessageProp(int qop, bool privState);
```

Constructor which sets the values for the qop and privacy state.

Throws GSSException if an error is detected.

Parameters:

qop	The desired QOP. Use 0 to request a default QOP.
privState	The desired privacy state. "true" for privacy and "false" for integrity only.

6.8.2. QOP

```
public int QOP;
```

QOP property. Use this to set or get the value of the QOP property. Set to 0 to request the default value.

Throws GSSException if an error is detected.

6.8.3. privacy

```
public bool privacy;
```

Privacy property. Use this to set or get the value of the privacy property.

Throws GSSException if an error is detected.

6.8.4. minorStatus

```
public int minorStatus;
```

Minor Status property. This property maintains the minor status that the underlying mechanism might have set.

Throws GSSException if an error is detected.

6.8.5. minorString

```
public string minorString;
```

Minor String property. This property maintains a string explaining the mechanism specific error code. The string will be empty if no mechanism error code has been set.

Throws GSSException if an error is detected.

6.8.6. isDuplicateToken

```
public bool isDuplicateToken;
```

Is Duplicate Token property. The value of this property indicates whether or not the token is a duplicate of an earlier token.

Throws GSSException if an error is detected.

6.8.7. isOldToken

```
public bool isOldToken;
```

Is Old Token property. The value of this property indicates whether or not the token's validity period has expired.

Throws GSSException if an error is detected.

6.8.8. isUnseqToken

```
public bool isUnseqToken;
```

Is Un-sequenced Token property. The value of this property indicates whether or not the token is being processed out of sequence.

Throws GSSException if an error is detected.

6.8.9. isGapToken

```
public bool isGapToken;
```

Is Gap Token property. The value of this property indicates whether or not an expected per-message was not received.

Throws GSSException if an error is detected.

6.9. public class ChannelBinding

The GSS-API accommodates the concept of caller-provided channel binding information. Channel bindings are used to strengthen the quality with which peer entity authentication is provided during context establishment. They enable the GSS-API callers to bind the establishment of the security context to relevant characteristics like addresses or to application specific data.

The caller initiating the security context must determine the appropriate channel binding values to set in the GSSContext object. The acceptor must provide an identical binding in order to validate that received tokens possess correct channel-related characteristics.

Use of channel bindings is optional in GSS-API. Since channel-binding information may be transmitted in context establishment tokens, applications should therefore not use confidential data as channel-binding components.

6.9.1. Constructors

```
public ChannelBinding(EndPoint initAddr,
                      EndPoint acceptAddr,
                      byte[] appData);
```

Create a ChannelBinding object with user supplied address information and data. "null" values can be used for any fields which the application does not want to specify.

Throws GSSException if an error is detected.

Parameters:

initAddr	The address of the context initiator. "null" value can be supplied to indicate that the application does not want to set this value.
acceptAddr	The address of the context acceptor. "null" value can be supplied to indicate that the application does not want to set this value.
appData	Application supplied data to be used as part of the channel bindings. "null" value can be supplied to indicate that the application does not want to set this value.

```
public ChannelBinding(byte[] appData);
```

Creates a ChannelBinding object without any addressing information.

Throws GSSException if an error is detected.

Parameters:

appData	Application supplied data to be used as part of the channel bindings.

6.9.2. initiatorAddress

```
public EndPoint initiatorAddress;
```

Initiator Address property. The value of this property indicates the initiator's address for the channel binding. The property value will be set to "null" if not set.

Throws GSSException if an error is detected.

6.9.3. acceptorAddress

```
public EndPoint acceptorAddress;
```

Acceptor Address property. The value of this property indicates the acceptor's address for the channel binding. The property value will be set to "null" if not set.

Throws GSSException if an error is detected.

6.9.4. applicationData

```
public byte[] applicationData;
```

Application Data property. The value of this property indicates the application data being used for the channel binding. The property value will be set to "null" if not set.

Throws GSSException if an error is detected.

6.9.8. Equals

```
public override bool Equals(Object obj);
```

Tests whether or not two channel binding objects match.

Throws GSSException if an error is detected.

Parameters:

obj	Another channel binding to compare with.

6.10. public class Oid

This class represents Universal Object Identifiers (Oids) and their associated operations.

Oids are hierarchically globally-interpretable identifiers used within the GSS-API framework to identify mechanisms and name formats.

The structure and encoding of Oids is defined in ISOIEC-8824 and ISOIEC-8825. For example the Oid representation of Kerberos V5 mechanism is "1.2.840.113554.1.2.2"

The GSSName name class contains public static Oid objects representing the standard name types defined in GSS-API.

6.10.1. Constructors

public Oid(string strOid);

Creates an Oid object from a string representation of its integer components (e.g. "1.2.840.113554.1.2.2").

Throws GSSException if an error is detected.

Parameters:

strOid The string representation for the oid.

public Oid(Stream derOid);

Creates an Oid object from its DER encoding. This refers to the full encoding including tag and length. The structure and encoding of Oids is defined in ISOIEC-8824 and ISOIEC-8825. This method is identical in functionality to its byte array counterpart.

Throws GSSException if an error is detected.

Parameters:

derOid Stream containing the DER encoded oid.

public Oid(byte[] DEROid);

Creates an Oid object from its DER encoding. This refers to the full encoding including tag and length. The structure and encoding of Oids is defined in ISOIEC-8824 and ISOIEC-8825. This method is identical in functionality to its byte array counterpart.

Throws GSSException if an error is detected.

Parameters:

derOid Byte array storing a DER encoded oid.

6.10.2. ToString

public override string ToString();

Returns a string representation of the oid's integer components in dot separated notation (e.g. "1.2.840.113554.1.2.2").

Throws GSSException if an error is detected.

6.10.3. Equals

```
public override bool Equals(Object Obj);
```

Tests if two OID objects represent the same oid value.

Throws GSSException if an error is detected.

Parameters:

obj	Another Oid object to compare with.

6.10.4. DER

```
public byte[] DER;
```

DER property. Property of the object which refers to its full ASN.1 DER encoding, the property includes the tag and length.

Throws GSSException if an error is detected.

6.10.5. containedIn

```
public bool containedIn(Oid[] oids);
```

Method to test if an Oid object is contained within the supplied Oid object array.

Throws GSSException if an error is detected.

Parameters:

oids	An array of oids to search.

6.11. public class GSSException : Exception

This exception is thrown whenever a fatal GSS-API error occurs including mechanism specific errors. It may contain both, the major and minor, GSS-API status codes. The mechanism implementers are responsible for setting appropriate minor status codes when throwing this exception. Aside from delivering the numeric error code(s) to the caller, this class performs the mapping from their numeric values to textual representations. All C# GSS-API methods and properties are declared throwing this exception.

6.11.1. Constants

All valid major GSS-API error code values are declared as constants in this class.

public const int BAD_BINDINGS

Channel bindings mismatch error.

public const int BAD_MECH

Unsupported mechanism requested error.

public const int BAD_NAME

Invalid name provided error.

public const int BAD_NAMETYPE

Name of unsupported type provided error.

public const int BAD_STATUS

Invalid status code error - this is the default status value.

public const int BAD_MIC

Token had invalid integrity check error.

public const int CONTEXT_EXPIRED

Specified security context expired error.

public const int CREDENTIALS_EXPIRED

Expired credentials detected error.

public const int DEFECTIVE_CREDENTIAL

Defective credential error.

public const int DEFECTIVE_TOKEN

Defective token error.

public const int FAILURE

General failure, unspecified at GSS-API level.

public const int NO_CONTEXT

Invalid security context error.

public const int NO_CRED

Invalid credentials error.

public const int BAD_QOP

Unsupported QOP value error.

public const int UNAUTHORIZED

Operation unauthorized error.

public const int UNAVAILABLE

Operation unavailable error.

public const int DUPLICATE_ELEMENT

Duplicate credential element requested error.

public const int NAME_NOT_MN

Name contains multi-mechanism elements error.

public const int DUPLICATE_TOKEN

The token was a duplicate of an earlier token. This is a fatal error code that may occur during context establishment. It is not used to indicate supplementary status values. The MessageProp object is used for that purpose.

public const int OLD_TOKEN

The token's validity period has expired. This is a fatal error code that may occur during context establishment. It is not used to indicate supplementary status values. The MessageProp object is used for that purpose.

public const int UNSEQ_TOKEN

A later token has already been processed. This is a fatal error code that may occur during context establishment. It is not used to indicate supplementary status values. The MessageProp object is used for that purpose.

```
public const int GAP_TOKEN
```

An expected per-message token was not received. This is a fatal error code that may occur during context establishment. It is not used to indicate supplementary status values. The MessageProp object is used for that purpose.

6.11.2. Constructors

```
public GSSException(int majorCode);
```

Creates a GSSException object with a specified major code.

Parameters:

majorCode The GSS error code causing this exception to be thrown.

```
public GSSException(int majorCode, int minorCode,
                    string minorString);
```

Creates a GSSException object with the specified major code, minor code, and minor code textual explanation. This constructor is to be used when the exception is originating from the security mechanism. It allows to specify the GSS code and the mechanism code.

Parameters:

majorCode	The GSS error code causing this exception to be thrown.
minorCode	The mechanism error code causing this exception to be thrown.
minorString	The textual explanation of the mechanism error code.

6.11.3. major

```
public int major;
```

Major code property. Property of the object representing the GSS error code that caused the exception to be thrown.

6.11.4. minor

```
public int minor;
```

Minor code property. Property of the object representing the mechanism error code that caused the exception to be thrown. The minor code property is set by the underlying mechanism. A setting of 0 for this property indicates that the mechanism error code is not set.

6.11.5. majorString

```
public string majorString;
```

Major string property. Property of the object explaining the GSS major error code causing the exception to be thrown.

6.11.6. minorString

```
public string minorString;
```

Minor string property. Property of the object explaining the mechanism specific error code causing the exception to be thrown. The string will be empty if no mechanism error code has been set.

6.11.7. ToString

```
public override string ToString();
```

Returns a textual representation of both the major and minor status codes.

6.11.8. Message

```
public string Message;
```

Message property. Error message explaining the reason for the exception.

7. Sample Applications

7.1. Simple GSS Context Initiator

```
using org.ietf.gss;

/**
 * This is a partial sketch for a simple client program that acts
 * as a GSS context initiator.  It illustrates how to use the C#
 * bindings for the GSS-API specified in
 * Generic Security Service API Version 2 : C# bindings
 *
 *
 * This code sketch assumes the existence of a GSS-API
 * implementation that supports the mechanism that it will need and
 * is present as a library package (org.ietf.jgss) either as part of
 * the standard JRE or in the CLASSPATH the application specifies.
 */

public class SimpleClient {

   private string serviceName; // name of peer (ie. server)
   private GSSCredential clientCred = null;
   private GSSContext context = null;
   private Oid mech; // underlying mechanism to use

   private GSSManager mgr = GSSManager.getInstance();

   ...
   ...

   private void clientActions() {

      initializeGSS();
      establishContext();
      doCommunication();
   }
```

```
      /**
       * Acquire credentials for the client.
       */
      private void initializeGSS() {

         try {

            clientCred = mgr.createCredential(null /*default princ*/,
                 GSSConstants.INDEFINITE_LIFETIME /* max lifetime */,
                 mech /* mechanism to use */,
                 GSSCredentialUsage.INITIATE_ONLY /* init context */);

            Console.Writeline("GSSCredential created for " +
                              cred.getName().ToString());
            Console.Writeline("Credential lifetime (sec)=" +
                              cred.getRemainingLifetime());

         } catch (GSSException e) {
            Console.Writeline("GSS-API error in credential acquisition:"
                              + e.Message);
                    ...
                    ...
           }

           ...
           ...
      }
```

```
      /**
       * Does the security context establishment with the
       * server.
       */
      private void establishContext() {

         byte[] inToken = new byte[0];
         byte[] outToken = null;

         try {

            GSSName peer = mgr.createName(
                                    serviceName,
                                    GSSNameTypes.NT_HOSTBASED_SERVICE);

            context = mgr.createContext(
                         peer,
                         mech,
                         gssCred,
                         GSSConstants.INDEFINITE_LIFETIME/*lifetime*/);

            // Will need to support confidentiality
            context.confidentiality = true;

            while (!context.isEstablished) {

               outToken = context.initSecContext(inToken,
                                                 0,
                                                 inToken.length);

               if (outToken != null)
                  writeGSSToken(outToken);

               if (!context.isEstablished)
                  inToken = readGSSToken();
            }

            GSSName peer = context.srcName;
            Console.Writeline(
                     "Security context established with " +
                     peer +
                     " using underlying mechanism " + mech.toString());

         } catch (GSSException e) {
            Console.Writeline(
               "GSS-API error during context establishment: " +
               e.Message);
         }
           ...
```

```
      }
```

```
      /**
       * Sends some data to the server and reads back the
       * response.
       */
      private void doCommunication()  {

         byte[] inToken = null;
         byte[] outToken = null;
         byte[] buffer;

         // Container for multiple input-output arguments to and
         // from the per-message routines (e.g., wrap/unwrap).
         MessageProp messgInfo = new MessageProp();

         try {

            /*
             * Now send some bytes to the server to be
             * processed. They will be integrity protected but
             * not encrypted for privacy.
             */

            buffer = readFromFile();

            // Set privacy to false and use the default QOP
            messgInfo.privacy = false;

            outToken = context.wrap(buffer,
                                    0,
                                    buffer.length,
                                    messgInfo);

            writeGSSToken(outToken);

            /*
             * Now read the response from the server.
             */

            inToken = readGSSToken();
            buffer = context.unwrap(inToken,
                                    0,
                                    inToken.length,
                                    messgInfo);
```

```
            // All ok if no exception was thrown!

            GSSName peer = context.srcName;

            Console.Writeline("Message from " +
                              peer.ToString() +
                              " arrived.");
            Console.Writeline("Was it encrypted? " +
                              messgInfo.privacy);
            Console.Writeline("Duplicate Token? " +
                              messgInfo.isDuplicateToken);
            Console.Writeline("Old Token? " +
                              messgInfo.isOldToken);
            Console.Writeline("Unsequenced Token? " +
                              messgInfo.isUnseqToken);
            Console.Writeline("Gap Token? " +
                              messgInfo.isGapToken);

            ...
            ...

         } catch (GSSException e) {
            Console.Writeline("GSS-API error in per-message calls: " +
                              e.Message);
            ...
            ...

         }

         ...
         ...

      } // end of doCommunication method

      ...
      ...

   } // end of class SimpleClient
```

7.2. Simple GSS Context Acceptor

```
using org.ietf.gss;

/**
 * This is a partial sketch for a simple server program that acts
 * as a GSS context acceptor. It illustrates how to use the C#
 * bindings for the GSS-API specified in
 * Generic Security Service API Version 2 : C# bindings
 *
 * This code sketch assumes the existence of a GSS-API
 * implementation that supports the mechanisms that it will need and
 * is present as a library package (org.ietf.jgss) either as part of
 * the standard JRE or in the CLASSPATH the application specifies.
 */

public class SimpleServer {

   private string serviceName;
   private GSSName name;
   private GSSCredential cred;

   private GSSManager mgr;

   ...
   ...

   /**
    * Wait for client connections, establish security contexts and
    * provide service.
    */
   private void loop() {

      ...
      ...

      mgr = GSSManager.getInstance();

      name = mgr.createName(serviceName,
                            GSSNameTypes.NT_HOSTBASED_SERVICE);

      cred = mgr.createCredential(name,
                                  GSSConstants.INDEFINITE_LIFETIME,
                                  null,
                                  GSSCredentialUsage.ACCEPT_ONLY);
```

```
         // Loop infinitely
         while (true) {

            Socket s = serverSock.accept();

            // Start a new thread to serve this connection
            Thread serverThread = new ServerThread(s);
            serverThread.start();

         }
      }

      /**
       * Inner class ServerThread whose run() method provides the
       * secure service to a connection.
       */

      private class ServerThread extends Thread {

         ...
         ...

         /**
          * Deals with the connection from one client. It also
          * handles all GSSException's thrown while talking to
          * this client.
          */
         public void run() {

            byte[] inToken = null;
            byte[] outToken = null;
            byte[] buffer;

            GSSName peer;

            // Container for multiple input-output arguments to and
            // from the per-message routines (ie. wrap/unwrap).
            MessageProp supplInfo = new MessageProp();

            GSSContext secContext = null;

            try {

               // Now do the context establishment loop

               GSSContext context = mgr.createContext(cred);
```

```
while (!context.isEstablished) {

   inToken = readGSSToken();

   outToken = context.acceptSecContext(inToken,
                                       0,
                                       inToken.length);

   if (outToken != null)
      writeGSSToken(outToken);

}

// SimpleServer wants confidentiality to be
// available. Check for it.
if (!context.confidentiality) {
   ...
   ...
}

GSSName peer = context.srcName;
Oid mech = context.getMech();
Console.Writeline("Security context established with " +
                  peer.ToString() +
                  " using underlying mechanism " +
                  mech.ToString());

// Now read the bytes sent by the client to be
// processed.
inToken = readGSSToken();

// Unwrap the message
buffer = context.unwrap(inToken,
                        0,
                        inToken.length,
                        supplInfo);

// All ok if no exception was thrown!
```

```
            // Print other supplementary per-message status
            // information
            Console.Writeline("Message from " +
                              peer.ToString() +
                              " arrived.");
            Console.Writeline("Was it encrypted? " +
                              supplInfo.privacy);
            Console.Writeline("Duplicate Token? " +
                              supplInfo.isDuplicateToken);
            Console.Writeline("Old Token? "  + supplInfo.isOldToken);
            Console.Writeline("Unsequenced Token? " +
                              supplInfo.isUnseqToken);
            Console.Writeline("Gap Token? "  + supplInfo.isGapToken);

            /*
             * Now process the bytes and send back an encrypted
             * response.
             */

            buffer = serverProcess(buffer);

            // Encipher it and send it across
            supplInfo.privacy = true; // privacy requested
            supplInfo.QOP = 0; // default QOP
            outToken = context.wrap(buffer,
                                    0,
                                    buffer.length,
                                    supplInfo);
            writeGSSToken(outToken);

         } catch (GSSException e) {

            Console.Writeline("GSS-API Error: " + e.Message);

            // Alternatively, could read the e.majorString
            // and the e.minorString properties.

            Console.Writeline("Abandoning security context.");

            ...
            ...
         }

         ...
         ...

      } // end of run method in ServerThread

   } // end of inner class ServerThread
```

```
    ...
    ...

} // end of class SimpleServer
```

8. Security Considerations

The level of security that can be obtained by using GSS-API is dependent on the following factors:

- The integrity of the libraries being utilized.
- The integrity of the systems where the application executes.
- The GSS-API mechanism utilized.
- The GSS-API services utilized by the application.
- The way that the application utilizes GSS-API.

Application as well as system installers need to be aware of the factors mentioned above to avoid security vulnerabilities.

9. IANA Considerations

This document has no actions for IANA.

10. Acknowledgments

The author would like to thank the following:

Kabat, J. and Upadhyay, M. for writing the Generic Security Service API Version 2 : Java Bindings specification [RFC2853] that constitutes the basis of this work.

Jeff Altman for his support and suggestions.

Corby Morris for his initial implementation.

Funding for the RFC Editor function is currently provided by the Internet Society.

11. Normative References

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC2853] Kabat, J. and Upadhyay, M., "Generic Security Service API Version 2 : Java Bindings", RFC 2853, June 2000.

[RFC1509] Wray, J., "Generic Security Service API : C-bindings", RFC 1509, September 1993.

12. Authors' Addresses

Juan Carlos Luciani
Novell, Inc.
1800 South Novell Place
Provo, Utah 84606
US

EMail: jluciani@novell.com

13. Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

14. Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

15. Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.