

# SGSS-API Documentation and Developers' Guide

*Report No. UIUCDCS-R-99-2141, UILU-ENG-99-1760*

August 1999



Jalal Al-Muhtadi  
M. Dennis Mickunas  
Roy H. Campbell

# Abstract

SESAME - *Secure European System for Applications in a Multi-vendor Environment* – is an authentication system that extends Kerberos by supporting public-key cryptography, role-based access control, attribute certificates, delegation, and extensive auditing. Additionally, SESAME incorporates a standard interface for program developers which is based on the Internet standard *Generic Security Services Application Programming Interface (GSS-API)* (followed by RFC 1508 [1], and finally RFC 2078 [3]).

UIUC SESAME [10] is a portable version of SESAME fully implemented in Java. Accordingly, a Java implementation of SESAME GSS-API is needed. This will act as an interface for applications utilizing UIUC SESAME, as well as a way to hide UIUC SESAME's implementation details from callers. This document is concerned with the use and implementation details of SGSS-API (SESAME GSS-API).

# Table of Contents

<b>TABLE OF CONTENTS.....</b>	<b>2</b>
<b>1. FOREWORD .....</b>	<b>4</b>
GSS-API.....	4
SESAME.....	5
<b>2. USING SGSS-API.....</b>	<b>7</b>
2.1 REQUIREMENTS FOR USING SGSS-API.....	7
2.2 RUNNING THE DEMO APPLICATIONS .....	8
<b>3. SGSS-API USER'S GUIDE .....</b>	<b>10</b>
3.1 GSS-API OPERATIONAL PARADIGM.....	10
3.2 SECURITY MECHANISMS .....	13
3.3 SECURITY NAMING .....	13
3.4 QUALITY OF PROTECTION (QOP).....	14
3.5 PER-MESSAGE SERVICES .....	14
3.6 PRIVILEGE ATTRIBUTES .....	16
3.7 GSS-API OVER UIUC SESAME .....	16
3.8 SAMPLE CODE.....	17
<b>4. SGSS-API INTERFACE DESCRIPTION.....</b>	<b>21</b>
4.1 CLASS OID .....	21
4.2 CLASS GSSMANAGER .....	22
4.3 CLASS GSSNAME .....	23
4.4 CLASS MESSAGEPROP.....	26
4.5 CLASS GSSCREDENTIAL.....	28
4.6 CLASS CHANNELBINDING.....	35
4.7 CLASS GSSCONTEXT.....	37
4.8 CLASS GSSEXCEPTION EXTENDS EXCEPTION .....	47
4.9 CLASS GSSATTRIBUTE .....	51
<b>5. SGSS-API DEVELOPERS' GUIDE .....</b>	<b>53</b>
5.1 MAPPING OF UIUC SESAME'S OBJECTS.....	53
5.2 GENERAL FORMAT OF TOKENS .....	54
5.3 INITIAL CONTEXT TOKEN.....	55
5.4 TARGET RESULT TOKEN .....	61
5.5 ERROR TOKEN.....	62
5.6 WRAP TOKEN.....	63
5.7 MIC TOKEN .....	64
5.8 FUTURE WORK.....	64
<b>REFERENCES.....</b>	<b>66</b>
<b>INDEX .....</b>	<b>67</b>

# 1. Foreword

**I**t is a dangerous world. Around the clock, hackers are trying to crack systems and gain unauthorized privileges. Security is a real concern in today's information systems, particularly distributed ones. Since client/server applications and distributed objects have components spread over different locations, there are more places that can be breached. Fortunately, numerous security mechanisms and technologies have been devised. However, with so many different security mechanisms and technologies, porting existing applications to different security environments and coping with the rapid development of the underlying mechanisms is a difficult and tedious task. Therefore, it is essential to introduce a standard interface to some generic security services, which would allow an application to benefit from security services independently from the underlying mechanism and programming language environment. This is where the Generic Security Services Application Programming Interface (GSS-API) comes into play.

## **GSS-API**

The advantages attained by using GSS-API include:

- **Mechanism independence:** The GSS-API defines a generic interface for acquiring security credentials, establishing security contexts, and requesting for confidentiality and integrity services for messages.
- **Communication protocol independence:** Existing network-enabled applications may use their own communication protocol, and, in conjunction, invoke GSS-API services.
- **Programming language environment independence.**
- **Separating cryptographic security from applications:** By using GSS-API, applications do not need to implement or link to cryptographic libraries, which leads to smaller application sizes and rapid development while reducing cryptographic awareness on the application side.
- **Easy and rapid deployment of security services to current and future applications.**

- Cryptographic algorithms' strength could be improved without any need to change the applications.
- Fulfills varying protection requirements in different environments (e.g. commercial, educational, governmental) and accommodates cryptography export restrictions.

## SESAME

Kerberos - an authentication protocol based on *Needham* and *Schroeder* secret-key protocol - became well-established in many security environments, however, there is a demand in some applications to extend Kerberos by supporting public key technology and providing security privileges instead of just a single identity. SESAME - *Secure European System for Applications in a Multi-vendor Environment* - extends Kerberos and provides additional services that include:

- Handling access control privileges for users, which are part of a data structure referred to as the "PAC" (Privilege Attribute Certificate) that is sent to the peer upon establishing a security context. The PAC is digitally signed by the initiator.
- Support for different key management and distribution schemes. Administrators can choose to use public or secret key technologies. Further, SESAME enables multiple-domain services with different policies.
- Enhanced support for delegation where access privileges can be transmitted and delegated to selected targets.

SESAME is accessed through the GSS-API. However, the GSS-API should be extended to handle access control privileges.

"*Niphilim*" is a research project undertaken by the System Software Research Group at UIUC, Computer Science Department. The project is implementing CORBA Security Services using Java. Part of the project is to re-implement a subset of SESAME entirely in Java [10]. Consequently, a Java implementation of SESAME GSS-API is needed. This document is concerned with the use and implementation details of SGSS-API (SESAME GSS-API).

This report is divided as follows: *Chapter two* discusses the setting-up of SGSS-API and the running of the demo applications included in the distribution, it also highlights the system requirements needed to use SGSS-API over UIUC SESAME. *Chapter three* is oriented towards users of the SGSS-API who would like to port their existing applications to a secure environment while keeping the details of the underlying security mechanism transparent. *Chapter four* provides a full documentation for the SGSS-API version 1.0a programming interface, including a detailed explanation of all the classes and public methods available. Finally, *chapter five* is meant to be a guide for developers who are interested in upgrading or extending the functionality of the SGSS-API, the chapter is also concerned with the implementation details of SGSS-API.

## 2. Using SGSS-API

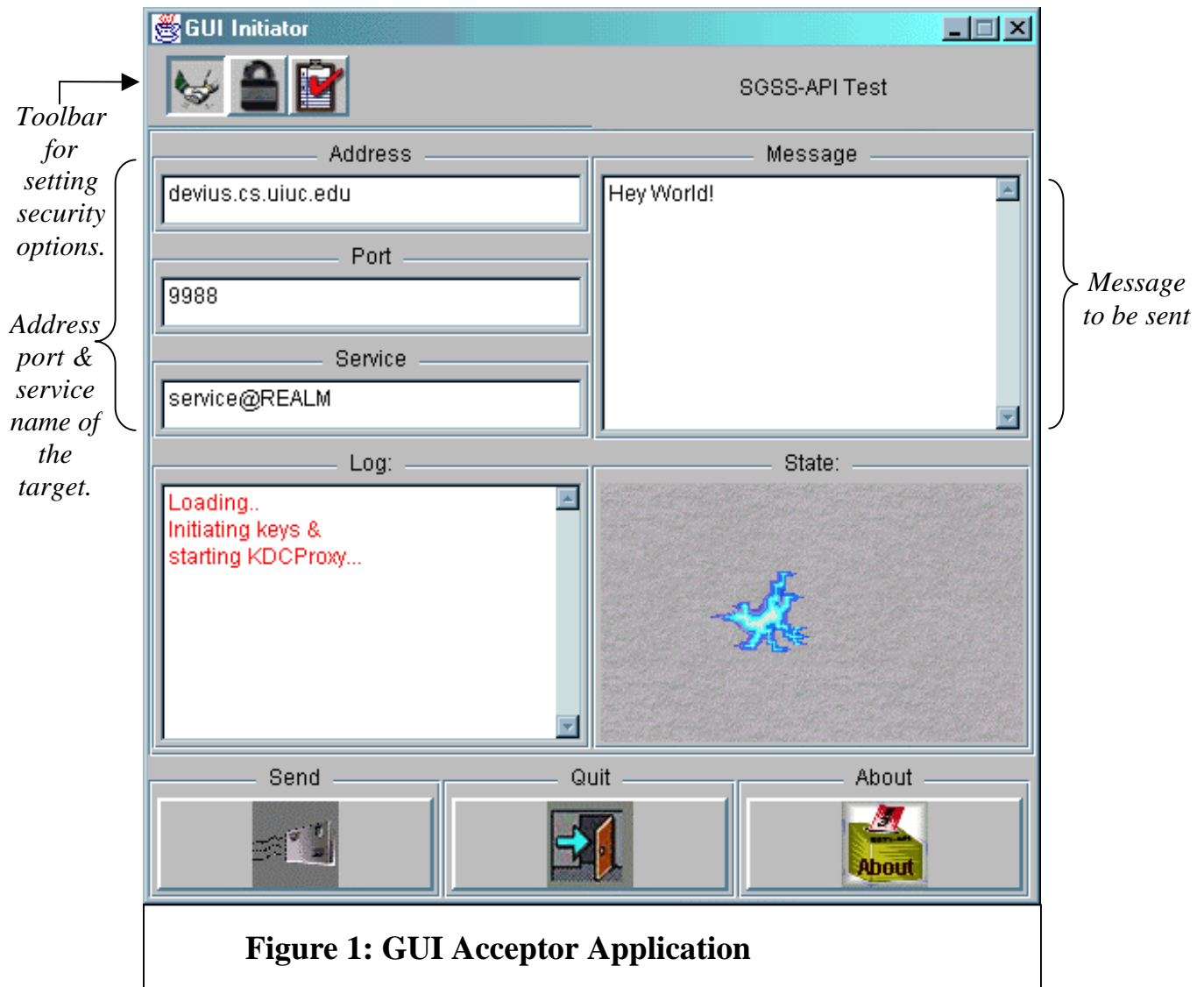
The general GSS-API interface is defined in [1] and [3]. However, for different programming environments specific parameter bindings are proposed. As for Java, the GSS-API bindings are described in two different Internet drafts. We decided to adhere to the Internet draft [5] (which defines Java bindings for GSS-API version 2). We also added a few extensions over [5] to accommodate SESAME's access control privileges. Chapter 3 contains full documentation for using SGSS-API. This chapter will discuss the system requirements and setup instructions for running the SGSS-API demo programs.

### 2.1 Requirements for Using SGSS-API

- SGSS-API can run on any platform with JDK 1.1+. SGSS-API was tested on Solaris 2.5+, Windows 98, and Windows NT.
- SGSS-API uses IAIK-JCE 2.5 for cryptographic services, which is available at <http://jcewww.iaik.tu-graz.ac.at/>
- SGSS-API uses IBM's Snacc for Java, which is a compiler for ASN.1 specifications. At the time of this writing, Snacc is available for download at <http://www.alphaworks.ibm.com/tech/snaccforjava>. The user will need to download Snacc classes.
- The user should download and install UIUC SESAME, over which the SGSS-API will run. Information about UIUC SESAME and its setup and distribution can be found at <http://choices.cs.uiuc.edu/Security/nephilim/> and in UIUC SESAME's documentation.
- To demonstrate the use of SGSS-API the distribution has two different demo applications, one is text-based and the other uses a GUI. The GUI application requires GJT v1.1 (Graphic Java Toolkit) which is included in the SGSS-API distribution.

For more information on setting up SGSS-API, please refer to the "readme.txt" file included in the SGSS-API distribution "**sgss-api.tar.gz**".

## 2.2 Running the Demo Applications



To run the GUI demo application, the CLASSPATH environment variable should be adjusted properly to point to the necessary classes mentioned above. More information about this is available in the SGSS-API distribution “**sgss-api.tar.gz**”. The GUI demo consist of two applications: the client (or the initiator) “GInitiator” (figure 1), and a server (or an acceptor): “GAcceptor” (figure 2). Here are the steps necessary to run this demo:

1. UIUC SESAME requires a KDC Server to be running (see next chapter). In case no such server is running, it could be activated by:

```
java TestKDC
```

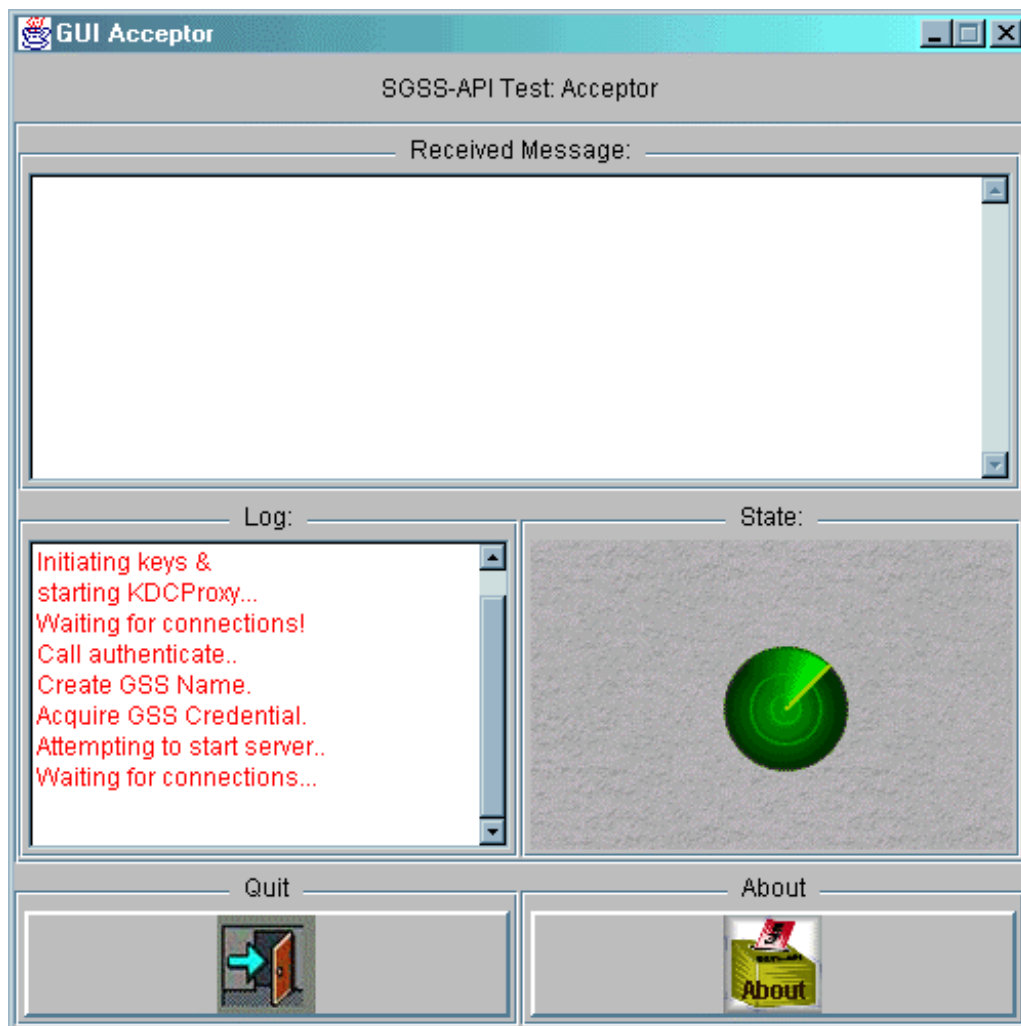
2. The initiator could be run by:

```
java GInitiator
```

3. Finally, the acceptor application should be invoked:

```
java GAcceptor
```

The toolbar at the initiator application allows the user to activate/deactivate mutual authentication, confidentiality services, and integrity services. The user is expected to provide the address, port number, and service name of the acceptor application. Upon clicking on the “send” button the “initiator” will establish a security context with the specified target application, and send the user’s message over the context.



**Figure 2: GUI Acceptor Application**

### 3. SGSS-API User’s Guide

This section is oriented towards users of the SGSS-API version 1.0a; i.e. the developers who plan to add security services to their applications through SGSS-API.

#### 3.1 GSS-API Operational Paradigm

At the outset, it is important to understand the GSS-API operational paradigm. GSS-API is not responsible for establishing communication channels between the client and server application. Instead, it is the GSS-API caller’s responsibility to establish a communication link with a peer application. Typically, as illustrated in figure 3, when an application uses GSS-API, it has access to a local GSS-API implementation. The local GSS-API implementation hides beneath it one or more security mechanisms. Thanks to GSS-API, the security mechanisms are transparent to the application. In fact, it is possible to “plug-in” a totally different security mechanism without affecting the application.

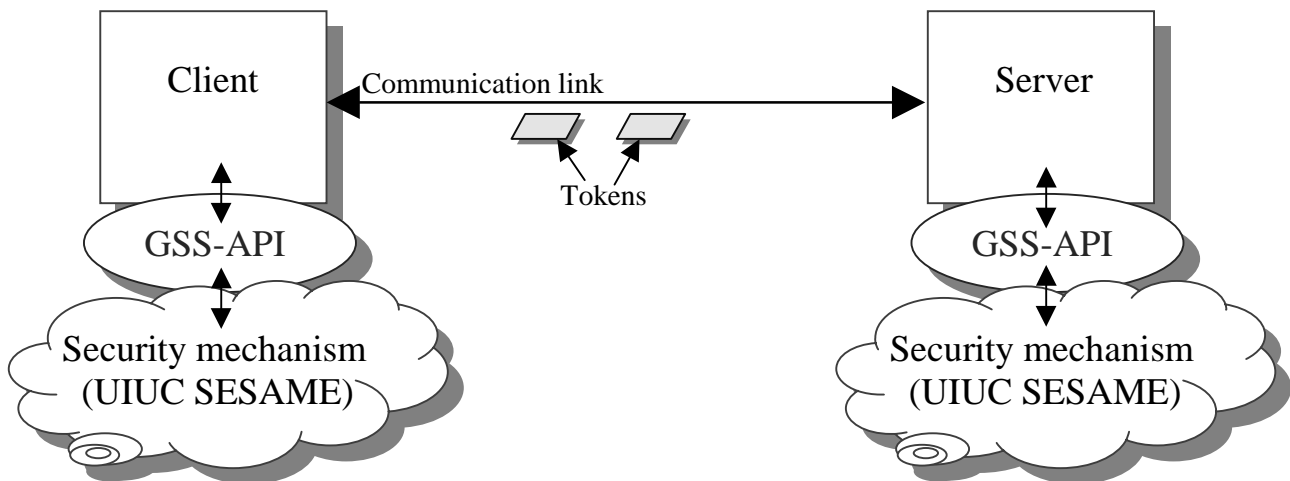


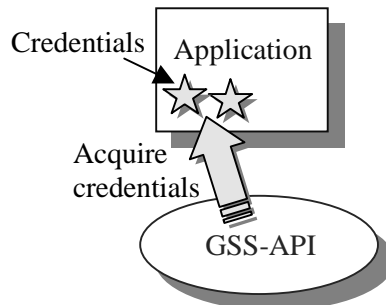
Figure 3: Using GSS-API

The GSS-API caller employs GSS-API services to protect its communication with integrity and/or confidentiality security services. This is accomplished through opaque data structures, or *tokens*, which are passed from a local GSS-API implementation to the caller. Upon the receipt of the tokens, the caller transfers the

tokens through a communication link to a remote peer. The remote peer, in turn, passes the tokens to its local GSS-API implementation for processing (figure 3).

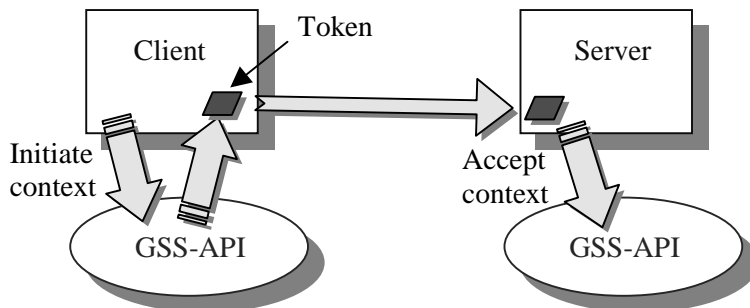
Figure 4 illustrates the data-flow between the GSS-API and the caller applications in more detail. The data-flow proceeds as follows:

a. Both the “client” and “server” applications acquire one or more credentials from the local GSS-API. A credential is a data structure that identifies a single entity and stores necessary information for establishing a security context on behalf of that entity.

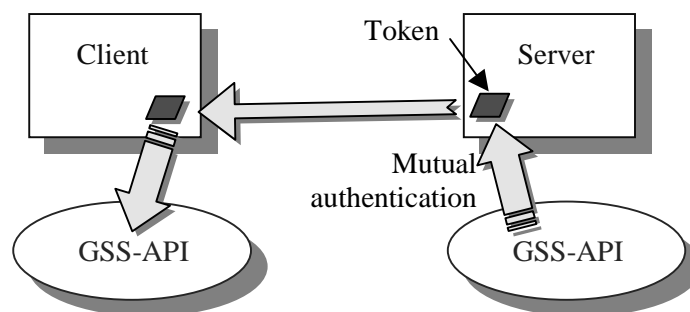


(a) Acquiring credentials.

b. Next, it is necessary to establish a “*security context*” between the communicating peers. The client “*initiates*” the security context establishment by invoking the “*context-initiate*” method in GSS-API and passing it the name of the target application. At this point, the client may optionally choose to request “*mutual authentication*”, in which the server application is expected to authenticate itself to the client as well. In either case, the GSS-API returns a token to the caller. This token should be sent by the client to the server over the communication link.



(b) Initiating a security context.



(c) Mutual authentication (if requested).

**Figure 4: GSS-API Operational paradigm.**

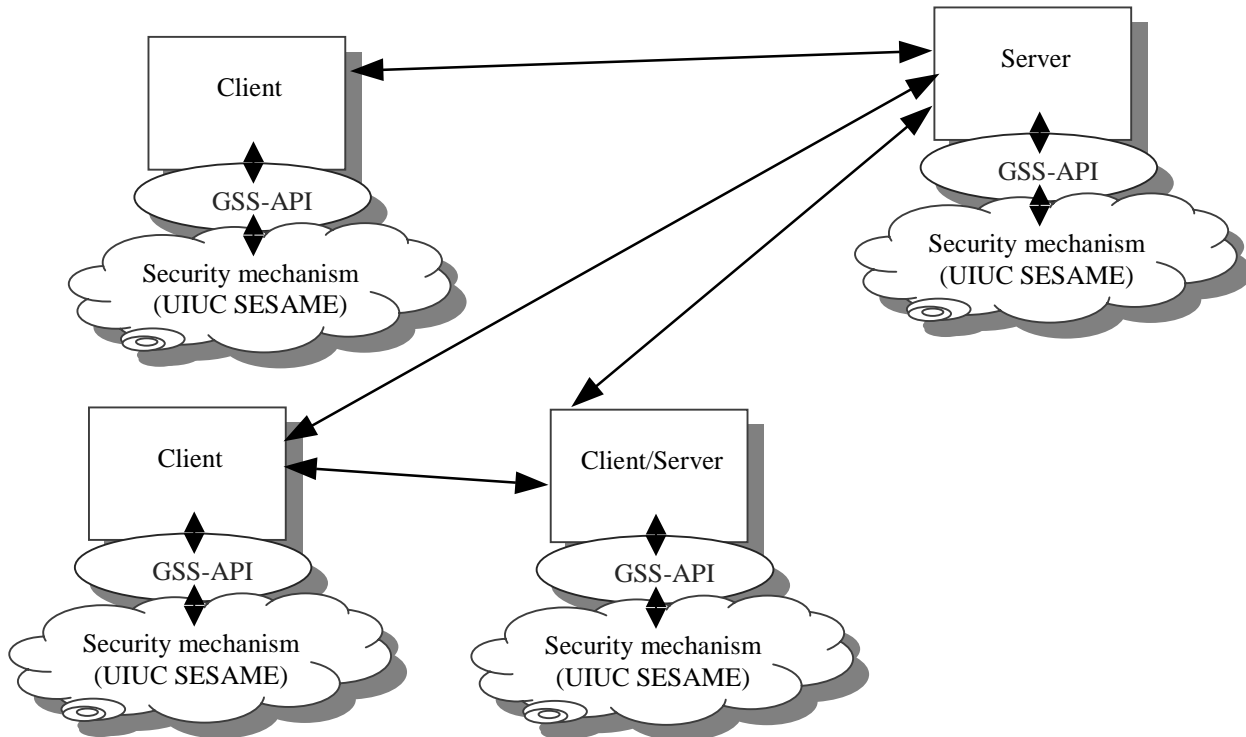
tokens through a communication link to a remote peer. The remote peer, in turn, passes the tokens to its local GSS-API implementation for processing (figure 3).

- c. Upon the receipt of the token, the server application passes it to the “*context-accept*” method in its local GSS-API for processing. If the client’s credentials permit it, and mutual authentication is not requested, a security context is established between the client and server, and security services become available for messages sent between them. Otherwise, if mutual authentication is requested, the server’s local GSS-API will return a token that needs to be sent by the server to the client to fulfill the mutual authentication requirement.

It should be noted however, that some security mechanisms may require passing additional tokens between the two peers before the actual establishment of the security context, therefore, steps *b* and *c* above may be repeated.

Once a security context is established between the client and server applications, it is possible to use per-message security services (which include integrity and confidentiality services) for messages sent back and forth between the communicating peers. The GSS-API operational paradigm is further demonstrated in the sample code provided later in this chapter.

Naturally, it is possible for an application to establish multiple security contexts with the same application or different applications (figure 5).



**Figure 5: Multiple clients and servers**

## 3.2 Security Mechanisms

In order to establish a security context between two peers, it is necessary to identify an underlying security mechanism that is supported by the two peers. The mechanism defines the cryptographic technologies to be used as well as the format of tokens and data elements to be passed between the peers.

It is possible for the GSS-API to exist atop several different security mechanisms (see [1], [3] and [5]), acting as a unified interface for accessing the services of different cryptographic technologies. In such a scenario, the GSS-API identifies a “*default*” mechanism that is used whenever the calling application does not explicitly request a certain mechanism. Alternatively, an application may query the local GSS-API for the supported mechanism(s) and choose a particular one if the application developer has a preference. In fact, the GSS-API interface includes the “GSSManager” class that could be used to query about the mechanisms supported and the default mechanism (see next chapter).

As SGSS-API was intended to interface with UIUC SESAME, SGSS-API version 1.0a only supports the SESAME v4.0 mechanism, using fully asymmetric security scheme with an exportable cryptographic profile. This is denoted by the object identifier: *{1.3.12.1.46.1.4.6.3}* (see [4] and [6] for more details). This mechanism is designated as the default mechanism for SGSS-API v1.0a.

## 3.3 Security Naming

Security names are used to uniquely identify entities. A name of an entity is transferred across the interface to initiate and accept security contexts. The GSS-API can be implemented to support a range of different name forms as various underlying mechanisms may support diverse naming schemes.

At the time of this writing, UIUC SESAME only supports Kerberos security names. Hence, SGSS-API v1.0a supports only this naming scheme.

The Kerberos name of an entity consists of a *name* and a *realm*. The printable format looks as follows:

```
principal_name@realm_name
```

## 3.4 Quality of Protection (QOP)

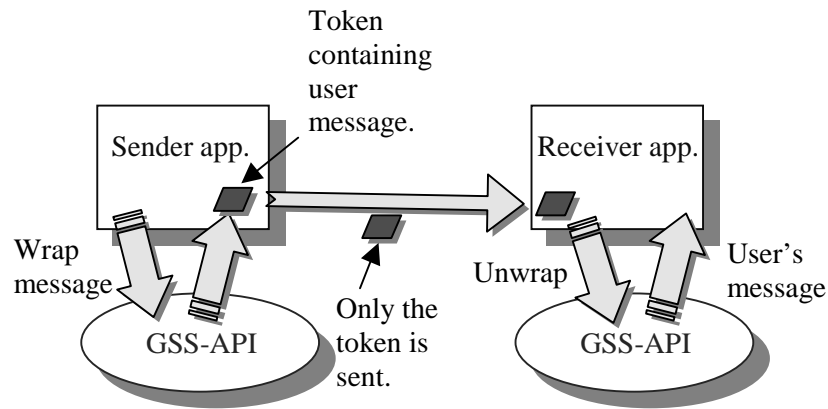
Some mechanisms provide the users with better control over the quality and power of per-message protection. Usually, higher quality implies extra processing and overhead while encrypting and decrypting. The GSS-API provides facilities which allow the user to manually select a particular QOP.

In SESAME, cryptographic and hashing algorithms are categorized according to their use. The categorization is then refined into cryptographic profiles. The mechanism type in SESAME incorporates the cryptographic profile that will be used [4]. SGSS-API v1.0a supports the “*exportable cryptographic profile*”. Accordingly, the QOP values supported in SGSS-API are:

- 0 - Use default.
- 1 - Use exportable cryptographic profile.

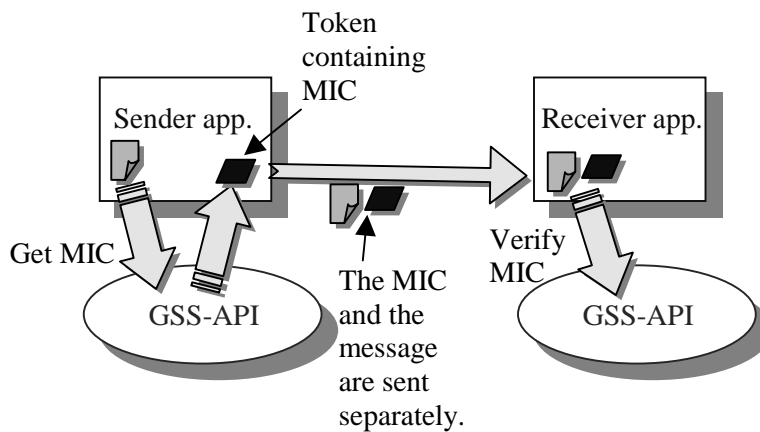
## 3.5 Per-Message Services

After establishing a security context between the two concerned parties, they can employ the per-message security services. The per-messages services (as defined by [3]) include two different flavors. In *message wrapping* (illustrated in figure 6) the user’s message is passed to the local GSS-API, which in turn would return a token that encapsulates the user’s message (which may be optionally encrypted) along with a cryptographic MIC for message integrity check. The sender application needs only to send the token. The receiver application invokes the “*unwrap*” method on the token, which would check the token’s integrity and retrieve the user’s message (after decrypting it, if necessary.).



**Figure 6: Per-Message Services - Wrapping Messages.**

The other per-message service is invoked by using the “*get MIC*” call. Upon passing the user’s data to this method, a token with a cryptographic MIC, which could be used for message integrity check, is returned. This token, however, does not encapsulate the user’s message; therefore, the sender application has to send the user’s message and the token separately. Furthermore, it is the receiver’s responsibility to associate the received message with the received token. Upon the receipt of the message and MIC token, the receiver application can invoke “*verify MIC*” method to check the integrity of the received message. This process is demonstrated in figure 7 below.



**Figure 7: Per-Message Services - MIC Tokens.**

## 3.6 Privilege Attributes

Among the advantages gained by employing SESAME is its support for privilege attributes. This is useful in circumstances where applications are security-aware, and would like to protect their own objects by controlling access to their own resources or audit their own security policies.

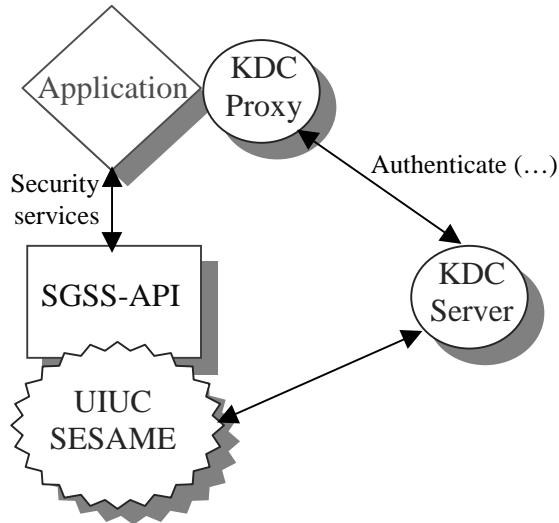
SGSS-API v1.0a provides the applications with methods to set and get the privilege attributes associated with a given credential.

At the time of this writing, UIUC SESAME only supports a single “role name” as a security attribute. For more information about utilizing the privilege attribute feature refer to SGSS-API interface description (chapter 4) and UIUC SESAME documentation.

## 3.7 GSS-API over UIUC SESAME

As far as the user of SGSS-API is concerned, there is no need to understand the details of the underlying UIUC SESAME mechanism, as GSS-API acts as a standardized interface. It should be noted, however, that it is necessary for a principal to be authenticated before it can employ GSS-API services. This principal authentication process is *not* part of GSS-API. Rather, either the security-aware application authenticates its user and passes the *authentication ticket* to the local SGSS-API to obtain one or more credentials, or, users are required to get authenticated through a system-dependent log-in process. In the latter case, if the user is successfully authenticated, his or her authentication information is stored at a cache that exists within the local UIUC SESAME implementation. Information in this cache, particularly the *authentication ticket*, is used by the local SGSS-API to grant credentials to the application.

If the application decides to authenticate its user, it should obtain a reference to a “KDCProxy” object (see figure 8). This object provides a method for authenticating principals. To further illustrate this, the next section provides some sample code. Additional information about the UIUC SESAME authentication process is available in UIUC SESAME’s documentation.



**Figure 8: UIUC SESAME uses a KDC Server. An application can authenticate an entity using a KDC Proxy object.**

### 3.8 Sample Code

The following is some sample code for an application that uses SGSS-API services to send a message securely to a server. Only the program's skeleton is given here. Full-functioning sample code is provided in SGSS-API distribution.

- Client Application:

```

import security.SGSS.*;
...

try {
    // start by creating a GSSName containing the name of the client application
    // use the default name type.
    GSSName myName = new GSSName("client@REALM", null);

    // acquire a GSSCredential. To do this, the application will authenticate
    // the user by calling the 'authenticate' method, then the returned
    // authentication ticket should be passed to create a credential.
    ClientTicket tgt;
    KDCProxy kdcProxy;

    ... //user authentication goes here, see the demo in the distribution.

    // use the authentication ticket to acquire a credential
  
```

```

GSSCredential myCred = new GSSCredential(myName, tgt, kdcProxy,
                                         GSSCredential.INITIATE_AND_ACCEPT);

//now, create the name for the service entity
GSSName targetName = new GSSName("target@REALM",
                                  GSSName.NT_HOSTBASED_SERVICE );

//create a context using the acquired credential for the
//default mechanism (SESAME)
GSSContext aCtxt = new GSSContext(targetName,
                                   null, /* default mechanism */
                                   myCred,
                                   GSSContext.INDEFINITE); /* desired lifetime */

//set desired context options - all others are false by default
aCtxt.requestConf(true);
aCtxt.requestMutualAuth(true);
aCtxt.requestReplayDet(true);
aCtxt.requestSequenceDet(true);

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

do {
    byte[] outTok = aCtxt.init(inTok, 0, inTok.length);
        /* for first call, inTok is ignored */

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

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

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

//context established, now display context information
System.out.println("Remaining lifetime in seconds = " + aCtxt.getLifetime());
System.out.println("Context mechanism = " + aCtxt.getMech());
System.out.println("Initiator = " + aCtxt.getSrcName());
System.out.println("Acceptor = " + aCtxt.getTargName());

if (aCtxt.getConfState())
    System.out.println("Confidentiality security service available");

if (aCtxt.getIntegState())
    System.out.println("Integrity security service available");

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

MessageProp mProp = new MessageProp(0, true);

```

```

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

if (mProp.getPrivacy())
    System.out.println("Message protected with privacy.");

sendToken(tok);

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

//also release the credential
myCred.dispose();

} catch (GSSEException e) {
    //display a detailed err msg
    System.err.println("GSSAPI error: " + e.getMessage());
}

```

- **Server Application:**

```

import security.SGSS.*;
...

try {

    // create a GSSName for the server using the default name type
    GSSName myName = new GSSName("target@REALM",null);

    // acquire a GSSCredential. To do this, the application will authenticate
    // the user by calling the 'authenticate' method, then the returned
    // authentication ticket should be passed to create a credential. (as
    // the case with the client application).
    ClientTicket tgt;
    KDCProxy kdcProxy;

    ... //user authentication goes here, see the demo in the distribution.

    // use the authentication ticket to acquire a credential
    GSSCredential myCred = GSSCredential(myName, tgt, kdcProxy,
        GSSCredential.INITIATE_AND_ACCEPT);

    //create acceptor GSS-API context
    GSSContext aCtxt = new GSSContext(myCred);

    do {

        byte [] inTok = readToken();

        byte []outTok = aCtxt.accept(inTok, 0,
            inTok.length);

        //if there's a output token, send it to peer
        if (outTok != null)
            sendToken(outTok);
    }
}

```

```

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

    // we're expecting a wrapped msg, get a token and unwrap it...
    byte [] myTok = readToken();

    MessageProp mProp = new MessageProp(0, true);

    byte []myMsg = aCtxt.unwrap(myTok, 0, myTok.length, mProp);
    // process the received myMsg ...

    //release the local-end of the context when done
    aCtxt.dispose();
} catch (GSSException e) {
    //display detailed error message.
    System.err.println("GSS-API error: " + e.getMessage());
}

```

- **Notes:**

- “readToken” and “sendToken” methods are implemented by the application. The methods simply read a token from a communication link and send a token over a communication link, respectively.
- The above applications assume that authentication is done within the code. As pointed out in section 3.7, if the system has a log-in process for users, then authentication part could be removed from the applications. In such a situation, the above code portions do not need to obtain references to “TicketClient” and “KDCProxy” objects. As a result, obtaining a GSS credential would be done through the following code portion:

```

GSSCredential myCred = GSSCredential(myName,
                                     GSSCredential.INITIATE_AND_ACCEPT);

```

The cache implemented at the local UIUC SESAME implementation should hold other required authentication information.

## 4. SGSS-API Interface Description

In this section, a full description of the SGSS-API interface (version 1.0a) is given. This is based on [5], with more methods added to support the handling of security attributes.

### 4.1 Class Oid

This is used to represent Object Identifiers (OIDs) and their associated operations.

- Constructors

- public Oid(String strOid) throws GSSEException

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

Parameter: strOid The string representation for the Oid, whitespaces are ignored.

- public Oid(InputStream derOid) throws GSSEException

Creates a complete Oid object from its DER encoding.

Parameter: derOid Stream containing the DER encoded oid.

- public Oid(byte[] DERoid) throws GSSEException

Creates an Oid object from its full DER encoding.

Parameter: derOid Byte array containing a DER encoded Oid.

- Public Methods:

- public 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”).

- `public String toRFC2078String()`

Returns a string representation of the Oid's in the format specified within RFC 2078 (e.g. "{ 1 2 840 113554 1 2 2 }").

- `public boolean equals(Object Obj)`

Returns "true" if the two Oid objects represent the same Oid value.

Parameter: obj Another Oid object to compare with.

- `public byte[] getDER()`

Returns the full ASN.1 DER encoding for this Oid object, which includes the tag and length.

- `public boolean containedIn(Oid[] oids)`

A utility method to test if an Oid object is contained within the supplied Oid object array.

Parameter: oids: An array of Oid to search.

## 4.2 Class GSSManager

This class implements functionality common to the entire SGSS-API package. All of its methods are static. It does not define any public constructors.

- Public Methods

- `public static Oid[] getMechs()`

Returns an array of Oid objects, one for each mechanism supported within this version of the SGSS-API package. A "null" value is returned when no mechanisms are supported. SGSS-API version 1.0a only supports the SESAME v4.0 mechanism, using fully asymmetric security scheme with an exportable cryptographic profile. This is denoted by the object identifier: {1.3.12.1.46.1.4.6.3}.

- `public static Oid[] getNamesForMech(Oid mech) throws GSSException`

Returns an array of Oids representing name types supported by the specified mechanism. SGSS-API version 1.0a only supports Kerberos names.

Parameters: mech The Oid object for the mechanism to query.

- `public static Oid[] getMechsForName(Oid nameType)`

Returns an array of Oid objects, one for each mechanism that support the specific name type. “null” is returned when no mechanisms are found to support the specified name type.

Parameters: nameType The Oid object for the name type to query.

SGSS-API v1.0a recognizes only Kerberos names: { 1.3.12.1.46.6.0 }.

- `public static Oid getDefaultMech()`

Returns the default mechanism Oid that will be used when a “null” Oid type is specified in the mechanism parameter within GSSCredential and GSSContext methods. SGSS-API v1.0a returns the SESAME v4.0 mechanism, using fully asymmetric security scheme with an exportable cryptographic profile.

### 4.3 Class GSSName

This class represents a single GSS-API principal. Different name formats and their definitions are identified with Oids.

- Constants

- `public static final Oid NT_HOSTBASED_SERVICE`

Oid indicating a host-based service whose name form is “service@hostname”. SGSS-API v1.0a supports only this type of services.

- `public static final Oid NT_USER_NAME`

An Oid indicating a named user on a local system.

- `public static final Oid NT_MACHINE_UID_NAME`

Oid indicating a name consisting of a numeric user identifier (UID) for a user on a local system.

- `public static final Oid NT_STRING_UID_NAME`

Oid indicating a name consisting of a string of digits representing the UID of a user on a local system.

- `public static final Oid NT_ANONYMOUS`

Oid indicating a name for representing an anonymous entity.

- `public static final Oid NT_EXPORT_NAME`

Oid indicating an exported name produced by the export method.

- Constructors

- `public GSSName(String nameStr, Oid type) throws GSSException`

Converts a string name to a GSSName object of the specified name type. The nameStr parameter is interpreted based on the specified type. SGSS-API v1.0a only supports NT\_HOSTBASED\_SERVICE type. If type is “null”, NT\_HOSTBASED\_SERVICE is used.

- `public GSSName(byte name[], Oid type) throws GSSException`

Converts a contiguous byte name to a GSSName object of the specified type. The name parameter is interpreted based on the specified type. Not supported by SGSS-API v1.0a, if called throws a GSSException.BAD\_NAME.

- public GSSName(String nameStr, Oid nameType, Oid mechType) throws GSSException

Converts a string name to a GSSName object of the specified type. The nameStr parameter is interpreted based on the specified type, “mechType” represents the mechanism for which this name should be created. If “mechType” is “null” the default mechanism is used. SGSS-API only supports the SESAME v4.0 mechanism, using fully asymmetric security scheme with an exportable cryptographic profile. {1.3.12.1.46.1.4.6.3}. SGSS-API v1.0a only supports NT\_HOSTBASED\_SERVICE type. If type is “null”, NT\_HOSTBASED\_SERVICE is used.

- public GSSName(byte name[], Oid nameType, Oid mechType) throws GSSException

Converts a contiguous byte name to a GSSName object of the type specified, the name parameter is interpreted based on the specified type, “mechType” represents the mechanism for which this name should be created. If “mechType” is “null” the default mechanism is used (the default mechanism will be defined in a later version of this document). Not supported by SGSS-API v1.0a, if called throws a GSSException.BAD\_NAME.

- Public Methods

- public boolean equals(Object another)

Returns “true” if two GSSName objects refer to the same principal.

- public GSSName canonicalize(Oid mechOid) throws GSSException

Creates a mechanism name from an arbitrary internal name. “mechOid” is the OID for the authentication mechanism for which the canonical form of the name is requested. SGSS-API only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

- `public byte[] export()` throws `GSSEException`

Returns a contiguous byte representation of a mechanism name. Not supported by SGSS-API v1.0a, if called throws a `GSSEException.UNAVAILABLE`.

- `public String toString()`

Returns a string representation of the `GSSName` object.

- `getStringNameType()` throws `GSSEException`

Returns the `Oid` representing the type of name returned in the “`toString`” method.

- `public Object clone()` throws `CloneNotSupportedException`

Creates a duplicate copy of this `GSSName` object.

- `public boolean isAnonymous()`

Returns “`true`” if this `GSSName` represents an anonymous entity, “`false`” otherwise.

## **4.4 Class MessageProp**

This is a helper class used to indicate the desired QOP and confidentiality per message. Upon receiving a token, the `MessageProp` class contains flags that can indicate old tokens, out of sequence tokens, duplicate tokens and/or gaps in

transmission (to use these flags, the corresponding services should be enabled when the security context is established).

- Constructors

- public MessageProp()

Creates a default MessageProp with a QOP (quality of protection) set to 0 and confidentiality to “false”.

- public MessageProp(int qop, boolean privState)

Creates a MessageProp with the specified QOP value and privacy state.

Parameters:

qop The desired QOP value.

privState privacy state

- Public Methods

- getQOP public int getQOP()

Returns the QOP value.

- getPrivacy public boolean getPrivacy()

Returns privacy state.

- public void setQOP(int qopVal)

Sets the QOP value. SGSS-API v1.0a supports QOP values of 0 (default) and 1 (exportable cryptographic profile).

- `public void setPrivacy(boolean privState)`

Sets the Privacy State.

- `public boolean isDuplicateToken()`

Returns “true” if this is a duplicate of an earlier token.

- `public boolean isOldToken()`

Returns “true” if the token’s validity period has expired.

- `public boolean isUnseqToken()`

Returns “true” if a later token has already been processed.

- `public boolean isGapToken()`

Returns “true” if an expected per-message token was not received.

## **4.5 Class GSSCredential**

This class implements GSS-API credentials and their operations. A credential holds all cryptographic information to allow the establishment of security context by the entity it represents.

- Class Constants

- `public static final int INITIATE_AND_ACCEPT`

Credential usage flag, indicating that the credential is to be used for initiating and accepting of security contexts.

- `public static final int INITIATE_ONLY`

Credential usage flag, indicating that the credential is to be used for initiating contexts only.

- `public static final int ACCEPT_ONLY`

Credential usage flag indicating that the credential is to be used for accepting contexts only.

- `public static final int INDEFINITE`

A lifetime constant representing indefinite amount of time for a credential lifetime.

- Constructors

- `public GSSCredential(int usage) throws GSSException`

Creates a default credential using the default mechanism, name, and an INDEFINITE lifetime. SGSS-API v1.0a supports only the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}). SGSS-API v1.0a does NOT assign a default name. Applications are recommended to pass their names.

Parameter:

usage The usage flag for this credential, this must be GSSCredential.ACCEPT\_AND\_INITIATE, GSSCredential.ACCEPT\_ONLY, or GSSCredential.INITIATE\_ONLY.

- `public GSSCredential(GSSName aName, int usage) throws GSSException`

Creates a default credential for the principal “aName” using the default mechanism and an INDEFINITE lifetime. SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

Parameters:

aName Name of the principal for whom this credential is to be acquired.  
usage the usage flag for this credential. The value should be GSSCredential.ACCEPT\_AND\_INITIATE, GSSCredential.ACCEPT\_ONLY, or GSSCredential.INITIATE\_ONLY.

- public GSSCredential(GSSName aName, int lifetime, Oid mechOid, int usage) throws GSSException

Creates a credential for a single mechanism,

Parameters:

aName Name of the principal for whom this credential is to be acquired. If it’s “null” the default name is used.

lifetime Lifetime of the credential in seconds. GSSCredential.INDEFINITE can be used to designate the maximum permitted lifetime.

mechOid The Oid of the desired mechanism. If it’s “null” the default mechanism is used.

usage the usage flag for this credential. The value should be GSSCredential.ACCEPT\_AND\_INITIATE, GSSCredential.ACCEPT\_ONLY, or GSSCredential.INITIATE\_ONLY.

SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

- public GSSCredential(GSSName aName, int lifetime, Oid mechs[], int usage) throws GSSException

Creates a credential over a set of mechanisms acquiring credentials for each mechanism in the set. To determine which mechanism acquisition of the credential was successful, the user should call the getMechs method. Calling this constructor is equivalent to creating a single mechanism credential and using addCred to extend the credential over other mechanisms.

Parameters:

aName Name of the principal for whom this credential is to be acquired. If it's "null" default name is used.

lifetime the lifetime of the credentials in seconds, value `GSSCredential.INDEFINITE` represents the maximum permitted lifetime.

mechOid an array of mechanisms over which the credential is to be acquired.

usage the usage flag for this credential. The value should be `GSSCredential.ACCEPT_AND_INITIATE`, `GSSCredential.ACCEPT_ONLY`, or `GSSCredential.INITIATE_ONLY`.

SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile (`{1.3.12.1.46.1.4.6.3}`).

- `public GSSCredential(GSSName aName, ClientTicket tkt, KDCProxy kdcProxy, int usage) throws GSSEException`

Creates a credential for the default mechanism, and passes the authentication ticket, and a reference to `KDCProxy` object for communicating with UIUC SESAME's KDC Server. For more information about this, check the UIUC SESAME documentation or the sample code. The credential lifetime will be set to `INDEFINITE`.

Parameters:

aName Name of the principal for whom this credential is to be acquired. If it's "null" default name is used.

tk authentication ticket of the user or client application.

kdcProxy a reference to a `KDCProxy` object used by UIUC SESAME to communicate with the KDC Proxy.

usage the usage flag for this credential. The value should be `GSSCredential.ACCEPT_AND_INITIATE`, `GSSCredential.ACCEPT_ONLY`, or `GSSCredential.INITIATE_ONLY`.

SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile (`{1.3.12.1.46.1.4.6.3}`).

Note: this constructor is not part of the Internet draft document.

- `public GSSCredential(GSSName aName, ClientTicket tkt, KDCProxy kdcProxy, int lifetime, int usage) throws GSSEException`

This constructor is identical to above, except that the lifetime is passed as a parameter.

Note: this constructor is not part of the Internet draft document.

- Public Methods

- public void dispose() throws GSSException

Releases the credential, and destroys sensitive data that may be stored within.

- public GSSName getGSSName() throws GSSException

Returns the name of the entity that acquired the credential.

- public GSSName getGSSName(Oid mechOID) throws GSSException

Returns per-mechanism name of the entity that acquired the credential. SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

Parameter: mechOID The mechanism Oid for which the name should be returned.

- public int getRemainingLifetime() throws GSSException

Returns the remaining lifetime in seconds for a credential. If more than one mechanism exists in the credential, the minimum lifetime of the mechanisms is returned. A return value of GSSCredential.INDEFINITE indicates that the credential does not expire. A return value of 0 indicates that the credential has already expired.

- public int getRemainingInitLifetime(Oid mech) throws GSSException

Returns the remaining lifetime in seconds for the credential to remain capable of initiating security contexts under the specified mechanism. A return value of `GSSCredential.INDEFINITE` indicates that the credential does not expire for context initiation. A return value of 0 indicates that the credential has already expired.

Parameter: mechOID The mechanism Oid for which the remaining lifetime is to be returned.

SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

- `public int getRemainingAcceptLifetime(Oid mech) throws GSSException`

Returns the remaining lifetime in seconds for the credential to remain capable of accepting security contexts under the specified mechanism. A return value of `GSSCredential.INDEFINITE` indicates that the credential does not expire for context acceptance. A return value of 0 indicates that the credential has already expired.

Parameter: mechOID: The mechanism Oid for which the remaining lifetime is to be returned.

SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

- `public int getUsage() throws GSSException`

Returns the credential usage flag. The return value could be `GSSCredential.INITIATE_ONLY`, `GSSCredential.ACCEPT_ONLY`, or `GSSCredential.INITIATE_AND_ACCEPT`.

- `public int getUsage(Oid mechOID) throws GSSException`

Returns the credential usage flag for the specified mechanism. The return could be `GSSCredential.INITIATE_ONLY`, `GSSCredential.ACCEPT_ONLY`, or `GSSCredential.INITIATE_AND_ACCEPT`.

Parameter: mechOID: The mechanism Oid for which the usage flag is returned.

SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

- `public Oid[] getMechs()` throws `GSSException`

Returns an array of mechanisms supported by this credential.

SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

- `public void add(GSSName aName, int initLifetime, int acceptLifetime, Oid mech, int usage)` throws `GSSException`

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

Parameters:

aName Name of the principal for whom this credential is to be acquired. If it's "null" the default name will be used.

initLifetime the lifetime in seconds for the credential to remain valid for the initiating of security contexts. The value `GSSCredential.INDEFINITE` could be used to represent the maximum permitted lifetime.

acceptLifetime the lifetime in seconds for the credential to remain valid for the acceptance of security contexts. The value `GSSCredential.INDEFINITE` could be used to represent the maximum permitted lifetime.

mechOid The mechanism Oid for which the credential is to be acquired.

usage the usage flag for this credential. The value should be `GSSCredential.ACCEPT_AND_INITIATE`, `GSSCredential.ACCEPT_ONLY`, or `GSSCredential.INITIATE_ONLY`.

SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

- `public boolean equals(Object another)`

Returns "true" if this credential is the same as "another", both must be acquired over the same mechanisms and refer to the same principal.

- Extensions to GSS-API

In XGSS-API the user may modify or retrieve security attribute(s) associated with a given credential. The following methods are added to this object to support this.

Note: these methods are not part of the Internet draft.

- `public GSSAttribute getAttributes()` throws `GSSEException`

Allows callers to retrieve attribute(s) associated with this credential object. Returns “null” if no attribute(s) are associated with this object.

SGSS-API v1.0a only supports a single “role name” with a list of target names.

- `public void setAttributes(GSSAttribute attrib)` throws `GSSEException`

Associates attribute(s) with this credential object. Note: sometimes some requested attributes are not granted. The user should check the actual attribute(s) assigned by calling “getAttributes”. A “null” value for “attrib” indicates no attributes.

SGSS-API v1.0a only supports a single “role name” with a list of target names.

- `public Object clone()` throws `CloneNotSupportedException`

Creates a duplicate copy of this credential object.

## 4.6 Class ChannelBinding

This class allows the caller to provide its own channel binding for establishing the security context. The initiator is responsible for determining the appropriate channel binding values. Similarly, the acceptor must provide an identical binding in order to be validated. Using channel bindings is optional in GSS-API. Since “InetAddress” is the only address type defined in Java, therefore it’s the only one supported for channel binding. SGSS-API v1.0a does not support channel binding.

- Constructors

- `public ChannelBinding(InetAddress initAddr, InetAddress acceptAddr, byte[] appData)`

Create a ChannelBinding object with the given address information and application data. “null” values can be used for any unspecified fields.

Parameters:

initAddr The address of the context initiator.

acceptAddr The address of the context acceptor.

appData Application supplied data to be used as part of the channel bindings. “null” can be used if none is needed.

- `public ChannelBinding(byte[] appData)`

Creates a ChannelBinding with no addressing information, defines some data to be used as part of the channel binding “appData”.

- Public Methods

- `public InetAddress getInitiatorAddress()`

Returns the initiator’s address for this channel binding. “null” is returned if the address has not been set.

- `public InetAddress getAcceptorAddress()`

Returns the acceptor’s address for this channel binding. “null” is returned if the address has not been set.

- `public byte[] getApplicationData()`

Returns application data being used as part of the ChannelBinding. “null” is returned if none exists.

- `public boolean equals(Object obj)`

Returns “true” if two channel bindings are the same, “false” otherwise.

Parameter: obj: Another channel binding to compare with.

## 4.7 Class GSSContext

This class represents the SGSS-API security context and its associated operations.

- Constructors

- `public GSSContext(GSSName peer, Oid mechOid, GSSCredential myCred, int lifetime) throws GSSException`

Creates a security context on the initiator’s side. The parameters include the peer’s name, the mechanism’s Oid (or null for default mechanism), the credential (or null for default credential) and lifetime of the context in seconds.

SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

SGSS-API v1.0a does not predefine a default credential. Applications must acquire a credential before creating a GSSContext object.

- `public GSSContext(GSSCredential myCred) throws GSSException`

Creates a security context on the acceptor’s side. The context properties will be determined from the token sent from the initiator. If “myCred” is null, a default credential is used. However, SGSS-API v1.0a does not predefine a default credential. Applications should acquire a credential before creating a GSSContext object.

- Public Methods

- `public byte[] init(byte inputBuf[], int offset, int len) throws GSSEException`

Initiates a security context. This method may return a token that needs to be sent to the peer for processing by the accept call. If “null” is returned no token needs to be returned to the acceptor. To determine whether the context is established, the application can call “isEstablished” method, which will return “false” if more tokens are expected to be supplied to the “init” method.

Parameters:

inputBuf token generated by the peer, this parameter is ignored on the 1st call.

offset the offset within inputBuf where the token begins

len length of the token, starting from offset.

- `public int init(InputStream inputBuf, OutputStream outputBuf) throws GSSEException`

Initiates a security context. This call is equivalent to the byte array based method (above). This method may write an output token to outputBuf which the application should send to the peer for processing. 0 bytes written to the output stream indicate that no more tokens need to be sent to the peer. The method will return either `GSSContext.COMPLETE` or `GSSContext.CONTINUE_NEEDED` indicating the status of the current context. A return value of `GSSContext.COMPLETE` indicates that the context establishment phase is complete for this peer, while `GSSContext.CONTINUE_NEEDED` means that another token is expected from the peer.

Parameters:

inputBuf Contains the token generated by the peer. This parameter is ignored on the first call.

outputBuf output stream where the output token will be written.

- `public byte[] accept(byte inTok[], int offset, int len) throws GSSEException`

Called by the context acceptor upon receiving a token from the peer. This may return an output token, which the application needs to send to the initiator for further processing by the init method. “null” indicates no more tokens need to be sent. To determine if the context establishment phase is complete the application can call “isEstablished” method.

Parameters:

inTok Token generated by the initiator,

offset the offset within the inTok where the actual token starts,

len the length of the token starting from ‘offset’.

- `public int accept(InputStream inputBuf, OutputStream outputBuf)`  
throws `GSSEException`

Called by the context acceptor upon receiving a token from the peer. This call is equivalent to the byte array method. This may write a token to the outputBuf which the application should send to the acceptor for processing. 0 bytes written to OutputBuf indicate no token needs to be sent to the peer. The method will return either `GSSContext.COMPLETE` or `GSSContext.CONTINUE_NEEDED` indicating the status of the current context. A return value of `GSSContext.COMPLETE` indicates that the context establishment phase is complete for this peer, while `GSSContext.CONTINUE_NEEDED` means that another token is expected from the peer.

- `public boolean isEstablished()`

Returns “true” if the context is fully established else returns false.

- `public void dispose()` throws `GSSEException`

Releases the context and any system resources or sensitive data stored in it.

- `public byte[] wrap(byte inBuf[], int offset, int len, MessageProp msgProp)` throws `GSSEException`

Method to apply per-message security services over a given context. This will return a token with a cryptographic MIC and may optionally encrypt the specified inBuf. The returned token will contain both the MIC and the message. “msgProp” object is used to specify the QOP and privacy status. The application is responsible for sending the tokens to the peer. Upon completion of this method, “msgProp” will contain the actually applied QOP and privacy status.

Parameters:

inBuf application message to be protected,

offset: offset where the data begins in inBuf,

len length of the data,

msgProp specifies the QOP and privacy status,

- public void wrap(InputStream inBuf, OutputStream outBuf, MessageProp msgProp) throws GSSEException

Method to apply per-message security services over a given context. This will write to ‘outBuf’ a token with a cryptographic MIC and may optionally encrypt the provided application message. The supplied message will be read from inBuf. “msgProp” object is used to specify the QOP and privacy status. The application is responsible for sending the tokens to the peer. Upon completion of this method, “msgProp” will contain the actually applied QOP and privacy status.

- public byte [] unwrap(byte[] inBuf, int offset, int len, MessageProp msgProp) throws GSSEException

Used by the peer to process tokens generated with the wrap method. This method will return the actual message from the token that was produced by the wrap function from the peer. The “msgProp” instance will indicate the applied QOP and the privacy status of the received message.

Parameters:

inBuf token received from peer containing the wrapped message,

offset the offset within the inBuf where the token begins,

len the length of the token starting from offset.

- `public void unwrap(InputStream inBuf, OutputStream outBuf, MessageProp msgProp)` throws `GSSEException`

Used by the peer to process tokens generated with the `wrap` method. This is similar to the above call, but expects to read the token from `inBuf`. This method will return the actual message from the token that was produced by the `wrap` function from the peer. The “`msgProp`” instance will indicate the applied QOP and the privacy status of the received message. The output is written to `outBuf`.

- `public byte[] getMIC(byte []inMsg, int offset, int len, MessageProp msgProp)` throws `GSSEException`

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

Parameters:

`inMsg` Message to generate over which MIC will be generated.

`offset` The offset within the `inMsg` where the token begins.

`len` The length of the token starting from “`offset`”.

`msgProp` Indicates the desired QOP.

Upon return from the method, this object will contain the actual applied QOP

- `public void getMIC(InputStream inMsg, OutputStream outBuf, MessageProp msgProp)` throws `GSSEException`

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 but uses streams for input and output. Note that privacy can only be applied through the `wrap` call. Upon return from the method, this object will contain the applied QOP.

- void verifyMIC(byte []inTok, int tokOffset, int tokLen, byte[] inMsg, int msgOffset, int msgLen, MessageProp msgProp) throws GSSEException

Verifies the cryptographic MIC, contained in the token parameter, over the supplied message. The msgProp parameter will contain the QOP indicating the strength of protection applied to the message.

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 starting from the offset,

inMsg Application message over which the cryptographic MIC is verified.

msgOffset The offset within the inMsg where the message begins.

msgLen The length of the message starting from the offset .

Upon return from the method, this object will contain the applied QOP and supplementary status values for the supplied token. The confidentiality state will be always set to "false".

- public void verifyMIC(InputStream inTok, InputStream inMsg, MessageProp msgProp) throws GSSEException

Verifies the cryptographic MIC, contained in the token parameter, over the supplied message. The msgProp parameter will contain the QOP indicating the strength of protection that was applied to the message. This method is equivalent in functionality to its byte array counterpart but uses streams for input.

Parameters:

inTok Contains the token generated by peer's getMIC method.

inMsg Contains application message over which the cryptographic MIC is verified.

msgProp Upon return from the method, this object will contain the applied QOP and supplementary status values for the supplied token.

- public byte [] export() throws GSSEException

Provided to support the sharing of work between multiple processes. Not supported by SGSS-API v1.0a, if called throws a GSSException.UNAVAILABLE.

- public void requestMutualAuth(boolean state) throws GSSException

Sets the request state of the mutual authentication flag for the context. Only valid before the context creation process on the initiator side.

- public void requestReplayDet(boolean state) throws GSSException

Sets the request state of the replay detection service for the context. Only valid before the context creation process on the initiator side.

- public void requestSequenceDet(boolean state) throws GSSException

Sets the request state for the sequence checking service of the context. Only valid before the context creation process on the initiator side.

- public void requestCredDeleg(boolean state) throws GSSException

Sets the request state for the credential delegation flag for the context. Only valid before the context creation process on the initiator side. Not supported by SGSS-API v1.0a, if called throws a GSSException.UNAVAILABLE.

- public void requestAnonymity(boolean state) throws GSSException

Requests anonymous support over the context. Only valid before the context creation process on the initiator side.

- public void requestConf(boolean state) throws GSSException

Requests that confidentiality service be available over the context. Only valid before the context creation process on the initiator side.

- `public void requestInteg(boolean state) throws GSSEException`

Requests that integrity services be available over the context. Only valid before the context creation process on the initiator side.

- `public void requestLifetime(int lifetime) throws GSSEException`

Sets the desired lifetime for the context in seconds. Only valid before the context creation process on the initiator side.

- `public void setChannelBinding(ChannelBinding cb) throws GSSEException`

Sets the channel bindings to be used during context establishment. Only valid before the context creation process.

- `public boolean getCredDelegState()`

Returns the state of the delegated credentials for the context. When issued before context establishment is completed or when the `isProtReady` method returns “false”, it returns the desired state, otherwise it will indicate the actual state over the established context.

- `public boolean getMutualAuthState()`

Returns the state of the mutual authentication option for the context. When issued before context establishment completes or when the `isProtReady` method returns “false”, it returns the desired state, otherwise it will indicate the actual state over the established context.

- `public boolean getReplayDetState()`

Returns the state of the replay detection option for the context. When issued before context establishment completes or when the `isProtReady` method returns “false”, it returns the desired state, otherwise it will indicate the actual state over the established context.

- `public boolean getSequenceDetState()`

Returns the state of the sequence detection option for the context. When issued before context establishment completes or when the `isProtReady` method returns “false”, it returns the desired state, otherwise it will indicate the actual state over the established context.

- `public boolean getAnonymityState()`

Returns “true” if this is an anonymous context. When issued before context establishment completes or when the `isProtReady` method returns “false”, it returns the desired state, otherwise it will indicate the actual state over the established context.

- `public boolean isTransferable() throws GSSException`

Returns “true” if the context is transferable to other processes through the use of the `export` method. This call is only valid on fully established contexts.

- `public boolean isProtReady()`

Returns “true” if the per message operations can be applied over the context.

- `public boolean getConfState()`

Returns the confidentiality service state over the context. When issued before context establishment completes or when the `isProtReady` method returns “false”, it returns the desired state, otherwise it will indicate the actual state over the established context.

- `public boolean getIntegState()`

Returns the integrity service state over the context. When issued before context establishment completes or when the `isProtReady` method returns “false”, it returns the desired state, otherwise it will indicate the actual state over the established context.

- `public int getLifetime()`

Returns the context lifetime in seconds. When issued before context establishment completes or when the `isProtReady` method returns “false”, it returns the desired lifetime, otherwise it will indicate the remaining lifetime for the context.

- `public GSSName getSrcName() throws GSSException`

Returns the name of the context initiator. This call is valid only after the context is fully established or the `isProtReady` method returns “true”. If “null” is returned then the context initiator’s name is not known.

- `public GSSName getTargName() throws GSSException`

Returns the name of the context target (acceptor). This call is valid only after the context is fully established or the `isProtReady` method returns “true”.

- `public Oid getMech() throws GSSException`

Returns the mechanism OID for this context. SGSS-API v1.0a only supports the SESAME v4.0 mechanism using fully asymmetric security scheme with an exportable cryptographic profile ({1.3.12.1.46.1.4.6.3}).

- public GSSCredential getDelegCred() throws GSSEException

Returns the delegated credential object on the acceptor's side. To check for availability of delegated credentials call getDelegCredState. This call is only valid on fully established contexts. SGSS-API v1.0a does not support delegated credentials.

- public boolean isInitiator() throws GSSEException

Returns "true" if this is the initiator of the context. This call is only valid after the context creation process has started.

- Extensions to GSS-API:

In XGSS-API the user may retrieve security attribute(s) associated with a given security context.

- public GSSAttribute getAttributes() throws GSSEException

Allows callers to retrieve attribute(s) associated with this security context object. This call is only valid on fully established contexts. A return value of "null" indicates no attributes are associated with this context.

SGSS-API v1.0a only supports a single "role name" with a list of target names.

## **4.8 Class GSSEException extends Exception**

This exception is thrown whenever a fatal SGSS-API error occurs. Errors could be "generic" GSS-API errors (major error codes) or mechanism-specific errors (minor error codes).

- Constants

All major GSS-API error codes values are declared as constants:

- `public static final int BAD_BINDINGS`  
Channel bindings mismatch error.
- `public static final int BAD_MECH`  
Unsupported mechanism.
- `public static final int BAD_NAME`  
Invalid name provided error.
- `public static final int BAD_NAMETYPE`  
Name of unsupported type.
- `public static final int BAD_STATUS`  
Invalid status code error
- `public static final int BAD_MIC`  
Token had invalid integrity.
- `public static final int CONTEXT_EXPIRED`  
Security context expired.
- `public static final int CREDENTIALS_EXPIRED`  
Credentials expired.
- `public static final int DEFECTIVE_CREDENTIAL`  
Defective credential error.
- `public static final int DEFECTIVE_TOKEN`

Defective token error.

- `public static final int FAILURE`

General failure - unspecified at GSS-API level.

- `public static final int NO_CONTEXT`

Invalid security context.

- `public static final int NO_CRED`

Invalid credentials error.

- `public static final int BAD_QOP`

Unsupported QOP value error.

- `public static final int UNAUTHORIZED`

Operation unauthorized.

- `public static final int UNAVAILABLE`

Operation unavailable error.

- `public static final int DUPLICATE_ELEMENT`

Duplicate credential element

- `public static final int NAME_NOT_MN`

Name contains multi-mechanism elements error

- `public static final int DUPLICATE_TOKEN`

The received token was a duplicate of an earlier token.

- `public static final int OLD_TOKEN`

The received token's validity period has expired.

- `public static final int UNSEQ_TOKEN`

A later token has already been processed.

- `public static final int GAP_TOKEN`

An expected per-message token was not received.

- Public Methods

- `public int getMajor()`

Returns the major code representing the GSS error code that caused this exception.

- `public int getMinor()`

Returns the mechanism-specific error code that caused this exception. Value of '0' means no minor code exists.

- `public String getMajorString()`

Returns a string explaining the GSS major error code causing this exception.

- `public String getMinorString()`

Returns a string explaining the mechanism-specific error code. If there's no mechanism-specific error code, an empty string is returned.

- `public String toString()`

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

- `public String getMessage()`

Returns a detailed message of this exception.

## 4.9 Class GSSAttribute

This class represents a single security attribute.

Notes:

- \* SGSS-API v1.0a only supports a single “role name” with a list of target names.
- \* This class is not part of the Internet draft.

For more information on using attributes, refer to UIUC SESAME documentation.

- Constructors

- `public GSSAttribute (String rolename, int flags, String[] targetNames) throws GSSException`

Creates a single attribute (role name), with a list of target names. `targetNames` may be “null” to represent any target.

SGSS-API v1.0a supports the following flags:

0 - no delegation,

1 - delegation is on.

- `public GSSAttribute (Oid rolename, int flags, String[] targetNames) throws GSSException`

Creates a single attribute (role name), with a list of target names. targetNames may be “null” to represent any target.

SGSS-API v1.0a supports the following flags:

0 - no delegation,

1 - delegation is on.

- Public Methods

- public String getName() throws GSSEException

Returns the role name.

- public String[] getTarget ()

Returns an array of target names, or null if the attribute is to be used at any service.

- public int getFlags()

Returns the flags associated with this attribute object.

SGSS-API v1.0a supports the following flags:

0 - no delegation,

1 - delegation is on.

## 5. SGSS-API Developers' Guide

This chapter is concerned with the implementation details of SGSS-API, and acts as a guide for developers interested in debugging, upgrading or extending the functionality of SGSS-API. Developers who are developing applications that use SGSS-API interface to UIUC SESAME need not know these details.

As described in section 3.1, the operational paradigm of GSS-API relies on passing *tokens* back and forth between the peer applications. A token is an opaque data structure that encodes necessary data to establish/manage security contexts, authenticate entities, and send user data with confidentiality and/or integrity services. It is the application's responsibility to transmit tokens to the peer it wishes to communicate with, however, the application need not know the internal structure of the tokens, rather, the application perceives these tokens as a stream of bytes. Further, the actual token format is different for different underlying mechanisms. For instance, the token format for Kerberos is different from SESAME's token formats.

In SGSS-API the token formats are based on [4]. In this chapter the format of tokens are described. Along the chapter, some familiarity with UIUC SESAME is assumed.

### 5.1 Mapping of UIUC SESAME's Objects

The "GSSCredential" object within GSS-API encapsulates the TGT (Ticket Granting Ticket) structure, which is granted upon authentication. As well as a "KDCProxy" object which is used for communicating with the KDC Server required by UIUC SESAME. These two objects are either passed to the "GSSCredential" upon its construction, or they are retrieved using the cache implemented at UIUC SESAME level.

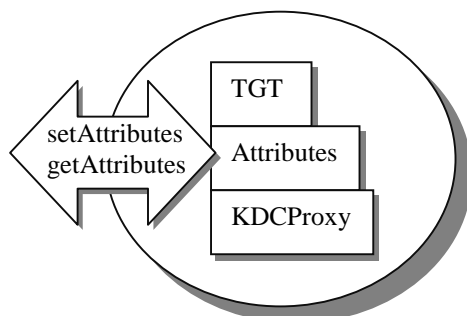


Figure 9: GSSCredential Object

Additionally, the "GSSCredential" object contains a reference to the privilege attributes associated with owner entity. These privilege attributes can be set or queried using the "setAttributes" and "getAttributes" methods. (Figure 9).

A “GSSContext” object contains a reference to the “GSSCredential” object of the creator. Additionally, a PAC (Privilege Attribute Certificate) is obtained from UIUC

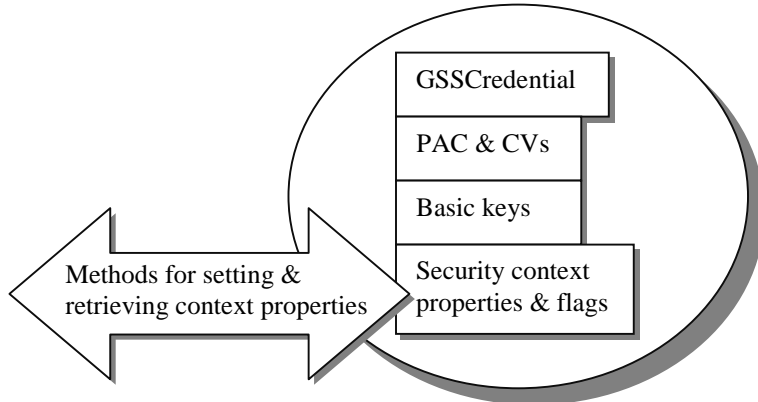


Figure 10: GSSContext Object

SESAME based on the attributes set within the “GSSCredential” of the caller. Basic symmetric keys are also created at SGSS-API level and stored within the “GSSContext” once a security context is established successfully. “GSSContext” object provides methods for managing the security context and controlling its properties.

## 5.2 General Format of Tokens

Tokens are simply the DER encoding of some ASN.1 types. SGSS-API employs IBM’s Snacc for encoding/decoding ASN.1 types. Tokens in SGSS-API are divided into different categories:

- Context establishment tokens: includes Initial Context Token, Target Result Token and Error Token.
- Per-message tokens: includes MIC Token and Wrap Token.

The general format for all these tokens is fixed (figure 11). As for the mechanism type, SGSS-API v1.0a uses the SESAME v4.0 mechanism using fully asymmetric

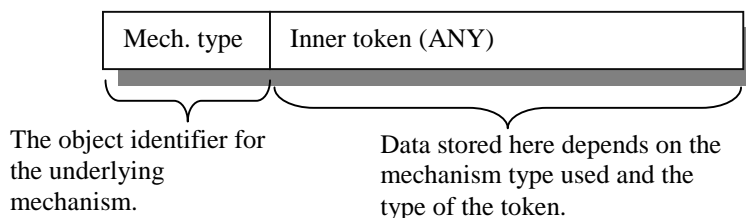


Figure 11: General format of tokens

security scheme with an exportable cryptographic profile, this is denoted with object

identifier *{1.3.12.1.46.1.4.6.3}*. The “inner token” part contains a DER encoding of one of the SESAME tokens defined in the following sections.

The ASN.1 type for the general token format is:

```
Token ::= [APPLICATION 0] IMPLICIT SEQUENCE {
    thisMech          OBJECT IDENTIFIER,
    innerContextToken ANY DEFINED BY thisMech
}
```

### 5.3 Initial Context Token

This token is used to initiate a security context. It is generated by the “init” method within the “GSSContext” object. Figure 12 gives an overview of the Initial Context Token format. Detailed description of the ASN.1 fields is given below.

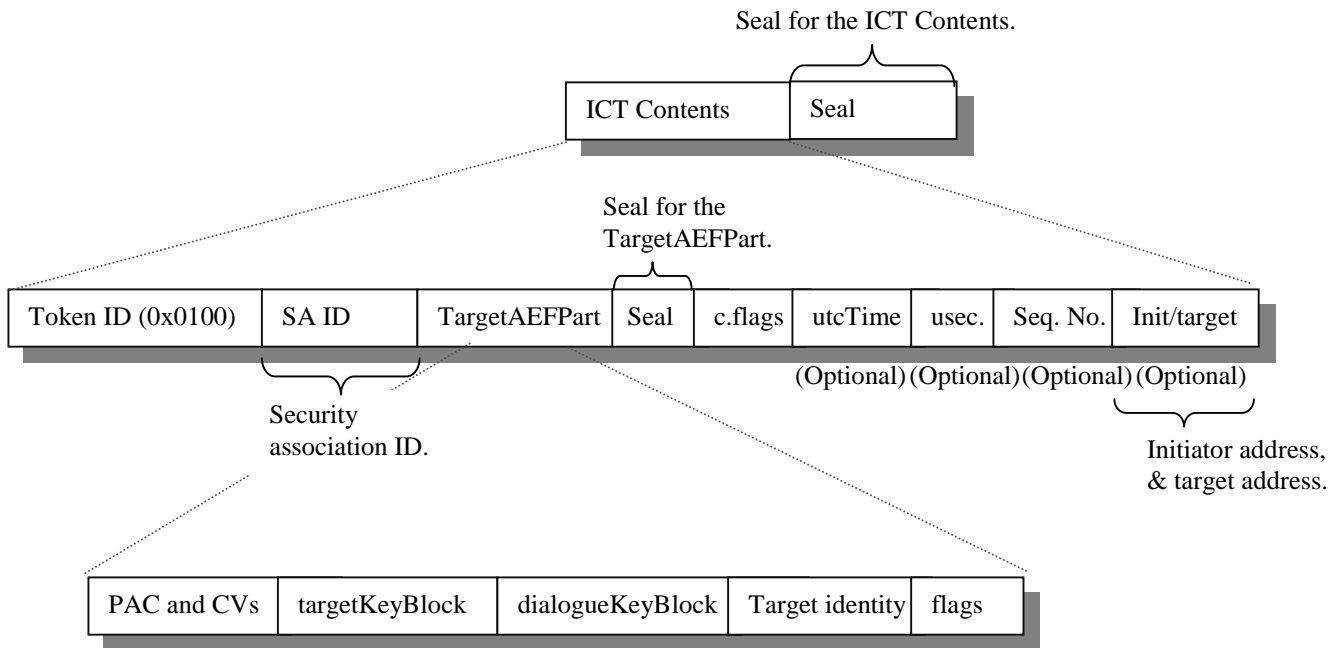


Figure 12: Initial Context Token.

The ASN.1 type for the Initial Context Token is:

```
InitialContextToken ::= SEQUENCE {
    ictContents      [0] ICTContents,
    ictSeal          [1] Seal
}
```

```

ICTContents ::= SEQUENCE {
    tokenId          [0] INTEGER,
    saId             [1] OCTET STRING,
    targetAEFFPart   [2] TargetAEFFPart,
    targetAEFFPartSeal [3] Seal,
    contextFlags     [4] BIT STRING {
        delegation (0),
        mutual-auth (1),
        replay-detect (2),
        sequence (3),
        conf-avail (4),
        integ-avail (5)
    },
    utcTime          [5] UTCTime OPTIONAL,
    usec             [6] INTEGER OPTIONAL,
    seq-number       [7] INTEGER OPTIONAL,
    initiatorAddress [8] ANY OPTIONAL,
    targetAddress    [9] ANY OPTIONAL
}

```

The values for the above fields are:

Token ID	The token ID for the Initial Context Token is 0x0100.
SA ID	<p>The security association ID. This is a random number for identifying the security association. Initially, this number is generated by the context initiator, and is processed by the acceptor as follows:</p> <ul style="list-style-type: none"> <li>• If mutual-authentication is not requested, thus, no Target Result Token is expected, then this SA ID value is taken to be the security association ID for the security context being established.</li> <li>• If mutual-authentication is requested, consequently, a Target Result Token is expected, then the acceptor generates its own random number and concatenates it to the SA ID generated by the initiator. The new concatenated value becomes the SA ID of the context.</li> </ul>
targetAEFFPart	This structure is sent to the Target AEF, its description is given below.
targetAEFFPart Seal	Seal of the TargetAEFFPart. The “Seal” structure will be described below. The seal is computed over the DER encoding of “targetAEFFPart” and the value is encrypted using the basic key.
contextFlags	Flags that indicate which services are requested over the security context. These flags include:

	<ul style="list-style-type: none"> <li>• <b>delegation</b>: if 0 delegation is forbidden.</li> <li>• <b>mutual-auth</b>: indicates whether mutual-authentication is requested.</li> <li>• <b>replay-detect</b>: indicates whether replay detect feature is requested.</li> <li>• <b>sequence</b>: indicates whether message sequencing is requested.</li> <li>• <b>conf-avail</b>: indicates whether confidentiality service is available at the initiator's side.</li> <li>• <b>integ-avail</b>: indicates whether integrity service is available at the initiator's side.</li> </ul>
utcTime	The initiator's UTC Time (optional).
usec.	Microsecond part of the initiator's UTC Time (optional).
seq-number	If present, specifies the initiator's initial sequence number. (Not used in SGSS-API v1.0a).
initiator Address	Initiator's network address, this is part of the optional channel bindings. (Not used in SGSS-API v1.0a)
targetAddress	Target's network address, this is part of the optional channel binding (not used in SGSS-API v1.0a).

The "Seal" structure is described below.

```

Seal ::= SEQUENCE {
    sealValue          [0] BIT STRING,
    symmetricAlgId    [1] ANY          OPTIONAL,
    hashAlgId         [2] ANY          OPTIONAL,
    targetName        [3] ANY          OPTIONAL,
    keyId             [4] INTEGER     OPTIONAL
}

```

sealValue	The value of the seal. This is computed by applying a one-way hash function on the DER encoding of some ASN.1 type. The hash value is then symmetrically encrypted and stored here.
symmetricAlgId	An indicator of the encryption algorithm (optional).
hashAlgId	An indicator of the hashing algorithm used (optional).
targetName	Identifies the target that shares the symmetric key used for encrypting the hash value (optional).
keyId	A serial number to uniquely identify the symmetric key used in encrypting the hash value (optional).

As for the “targetAEFPart” its contents are illustrated in figure 12. Its ASN.1 specification is:

```
TargetAEFPart ::= SEQUENCE {
    pacAndCVs      [0] OCTET STRING,
    targetKeyBlock [1] TargetKeyBlock,
    dialogueKeyBlock [2] ANY          OPTIONAL,
    targetIdentity [3] ANY          OPTIONAL,
    flags          [4] BIT STRING {
        delegation      (0)
    }
}
```

pacAndCVs	The initiator’s privilege and security attributes to be used for this security association. This contains the PAC along with PAC protection information. This structure will be constructed at the underlying UIUC SESAME mechanism and will be passed to SGSS-API as DER encoding.
targetKeyBlock	This structure carries the basic key that will be used in this security association. This is described later in this section.
dialogueKeyBlock	Used to establish an integrity and confidentiality dialogue keys. (Optional in SGSS-API v1.0a).
targetIdentity	The identity of the intended target of the security association.
flags	Only has a delegation flag to indicate whether delegation is permitted or not.

In SESAME, two separate symmetric keys are established during a security context. The *basic key* is established between the initiator and the target AEF and is used for protecting the PAC and deriving the dialogue keys. The integrity and confidentiality *dialogue keys* are established between the initiator and the target to protect application data. The basic key is transmitted through the “targetKeyBlock” structure. As of the time of this writing, UIUC SESAME assumes that every target is its own target AEF, thus, SGSS-API v1.0a only establishes a basic key (or a *session key*) for protecting the PAC as well as user’s messages.

In SESAME, there are different schemes for distributing the basic key. SGSS-API v1.0a and current UIUC SESAME version implements the *full asymmetric key distribution scheme*, where both the initiator and acceptor possess a public/private key

pair. The “targetKeyBlock” in this case should include the basic key encrypted using the target’s public key. To achieve this, a modified SPKM code is used [4].

The “targetKeyBlock” contains:

```
TargetKeyBlock ::= SEQUENCE {
    kdSchemeOID    [2]    OBJECT IDENTIFIER,
    targetKDSpart  [3]    ANY    OPTIONAL,
    targetPart     [4]    ANY    OPTIONAL
}
```

kdSchemeOID	Identifies the key distribution scheme. In SGSS-API v1.0a this contains the object identifier for the full asymmetric scheme.
targetKDSpart	This is not present in case of a full asymmetric key distribution scheme.
targetPart	In case of a full asymmetric key distribution, this contains a modified SPKM_REQ structure. (Described below).

The SPKM\_REQ code in SGSS-API is based on [4] and consists of:

```
SPKM-REQ ::= SEQUENCE {
    requestToken      REQ-TOKEN,
    req-integrity     [1]    ANY    OPTIONAL,
    certif-data       [2]    ANY    OPTIONAL,
    auth-data         [3]    ANY    OPTIONAL
}

REQ-TOKEN ::= SEQUENCE {
    tok-id            INTEGER,
    context-id        BIT STRING,
    pvno              BIT STRING,
    timestamp         UTCTime    OPTIONAL,
    randSrc           BIT STRING,
    targ-name         ANY,
    src-name          ANY,
    req-data          Context-Data,
    validity          [0] ANY    OPTIONAL,
    key-estb-set      [1] KeyEstAlgs,
    key-estb-req      BIT STRING    OPTIONAL,
    key-src-bind      SEQUENCE {
        src-name ANY,
        symm-key BIT STRING
    } OPTIONAL
}

Context-Data ::= SEQUENCE {
    channelId         OCTET STRING,
    seq-number        INTEGER OPTIONAL,
```

```

        options          BIT STRING,
        conf-alg         Conf-Algs,
        intg-alg         Intg-Algs
    }

KeyEstAlgs ::= SEQUENCE OF ANY

Conf-Algs ::= CHOICE {
    NULL
}

Intg-Algs ::= SEQUENCE OF ANY

KeyEstablishmentData ::= SEQUENCE {
    encryptedPlainKey    [0] BIT STRING,
    targetName           [1] ANY          OPTIONAL,
    nameHashingAlg       [2] ANY          OPTIONAL
}

```

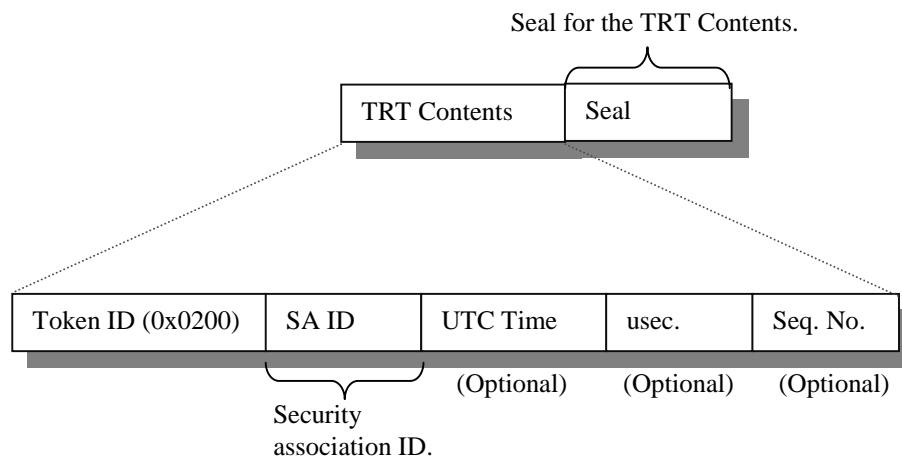
Based on [4] the following values are assigned to the various SPKM\_REQ fields:

requestToken:	
- tok-id	Not used: fixed value of 0.
- context-id	Not used: fixed value of a bit string containing one zero bit.
- pvno	Not used: fixed value of a bit string containing one zero bit.
- timestamp	Creation time of SPKM_REQ.
- randSrc	Random bit string.
- targ-name	SGSS-API v1.0a uses the printable Kerberos name of the target.
- src-name	SGSS-API v1.0a uses the printable Kerberos name of the source.
- req_data:	
-- channelId	Not used: octet string of one octet whose value is 0.
-- seq-number	Missing.
-- options	Not used: all bits are set to 0.
-- conf_alg	Not used: NULL choice.
-- intg_alg	Not used: filled with a “SEQUENCE OF” with zero elements.
- validity	Not used in SGSS-API v1.0a.
- key_est_set	Only one element is present: an object identifier representing SESAME Key Establishment Algorithm (sesame-key-est-alg).
- key_set_req	Contains a “KeyEstablishmentData” structure. In SGSS-API v1.0a only the field “encryptedPlainKey” is present, which contains the basic key encrypted using the public key of the target.
- key_src_bind	Missing.

req_integrity	Not used in SGSS-API v1.0a.
certif_data	Not used in SGSS-API v1.0a.
auth_data	Not used in SGSS-API v1.0a.

## 5.4 Target Result Token

A Target Result Token is returned to the context initiator, if mutual-authentication is requested, to authenticate the target to the initiator. The Target Result Token contents are shown in figure 13.



**Figure 13: Target Result Token.**

The ASN.1 type for a Target Result Token is as follows:

```

TargetResultToken ::= SEQUENCE {
    trtContents    [0] TRTContents,
    trtSeal       [1] Seal
}

TRTContents ::= SEQUENCE {
    tokenId       [0] INTEGER,
    saId          [1] OCTET STRING,
    utcTime       [5] UTCTime    OPTIONAL,
    usec          [6] INTEGER     OPTIONAL,
    seq-number    [7] INTEGER     OPTIONAL
}

```

The values for the TRT Contents fields are:

<b>Token ID</b>	The token ID for a Target Result Token is 0x0200.
<b>SA ID</b>	The security association ID.
<b>utcTime</b>	The acceptor's UTC Time (optional).
<b>usec.</b>	Microsecond part of the acceptor's UTC Time (optional).
<b>Seq-number</b>	Could be optionally used for message sequencing.

## 5.5 Error Token

An Error Token could be returned by either “init” or “accept” methods at the presence of an error while initiating or accepting a security context.

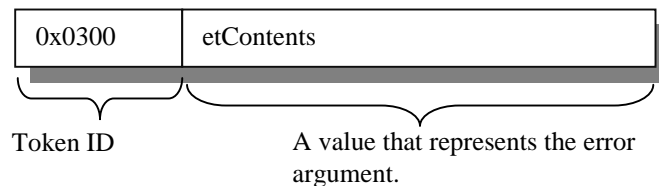


Figure 14: Error Token

The ASN.1 type for the Error Token is:

```

ErrorToken ::= SEQUENCE {
    tokenType      [0] OCTET STRING,
    etContents     [1] ErrorArgument
}

ErrorArgument ::= ENUMERATED {
    gss-ses-s-sg-server-sec-assoc-open           (1),
    gss-ses-s-sg-incomp-cert-syntax             (2),
    gss-ses-s-sg-bad-cert-attributes            (3),
    gss-ses-s-sg-inval-time-for-attrib         (4),
    gss-ses-s-sg-pac-restrictions-prob         (5),
    gss-ses-s-sg-issuer-problem                 (6),
    gss-ses-s-sg-cert-time-too-early           (7),
    gss-ses-s-sg-cert-time-expired             (8),
    gss-ses-s-sg-invalid-cert-prot             (9),
    gss-ses-s-sg-revoked-cert                   (10),
    gss-ses-s-sg-key-constr-not-supp            (11),
    gss-ses-s-sg-init-kd-server-unknown        (12),
    gss-ses-s-sg-init-unknown                   (13),
    gss-ses-s-sg-alg-problem-in-dialogue-key-block (14),
    gss-ses-s-sg-no-basic-key-for-dialogue-key-block (15),
    gss-ses-s-sg-key-distrib-prob              (16),
    gss-ses-s-sg-invalid-user-cert-in-key-block (17),

```

```

gss-ses-s-sg-unspecified          (18),
gss-ses-s-g-unavail-qop          (19),
gss-ses-s-sg-invalid-token-format (20)
}

```

## 5.6 Wrap Token

Per-Message Tokens have the same ASN.1 type and format (figure 15). As the case with other tokens, a seal is calculated over the contents of the token.

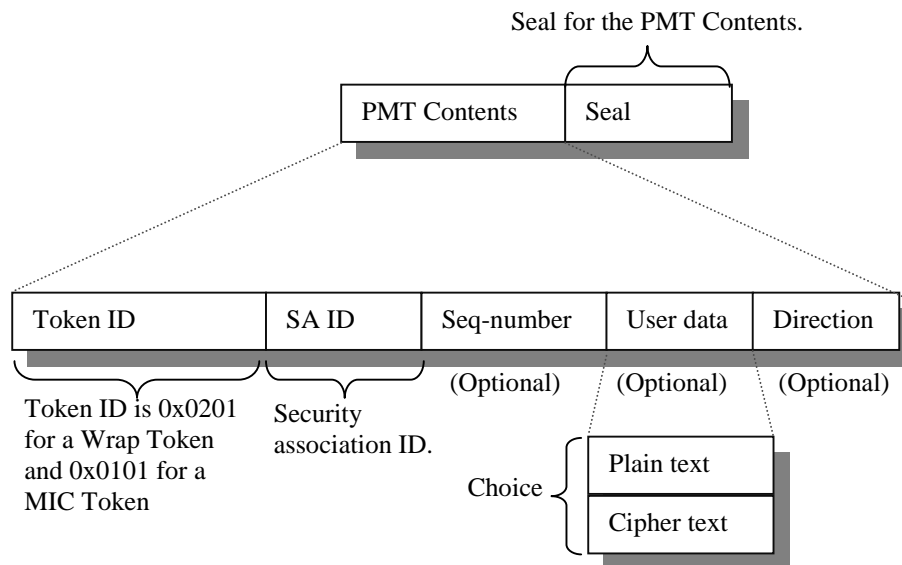


Figure 15: Wrap Token

Both Per-Message Tokens (Wrap Token and MIC Token) have the same ASN.1 type:

```

PMTToken ::= SEQUENCE {
    pmtContents  [0]  PMTContents,
    pmtSeal      [1]  Seal
}

PMTContents ::= SEQUENCE {
    tokenId      [0]  INTEGER,
    saId         [1]  OCTET STRING,
    seq-number   [2]  INTEGER OPTIONAL,
    userData     [3]  CHOICE {
        plaintext  BIT STRING,
        ciphertext OCTET STRING
    }
}

```

```

    } OPTIONAL,
    directionIndicator    [4]    BOOLEAN OPTIONAL
}

```

The values for the PMT Contents fields are as follows:

<b>Token ID</b>	The token ID for a Wrap Token is 0x0201, whereas the token ID for an MIC Token is 0x0101.
<b>SA ID</b>	The security association ID.
<b>Seq-number</b>	Could be optionally used for message sequencing.
<b>User Data</b>	The user's message which is either present as plain text (if no confidentiality service is requested) or encrypted.
<b>Direction indicator</b>	"False" if the sender is the context initiator. "True" otherwise.

The seal over PMT Contents is calculated as discussed in previous sections. However, if the user data is to be encrypted, the seal is calculated *before* encrypting the user data.

## 5.7 MIC Token

The ASN.1 type is the same as in section 5.6 and figure 15 above. The values of the PMT Contents fields are identical here. However, the user data is *not* present in a MIC Token, since such a token does not encapsulate the user data, rather, it merely calculates a MIC (Message integrity check).

To calculate the MIC the user's data is *temporarily* placed in the user data field (as plain text) and the seal is computed over the DER encoding of PMT Contents. The user data is then removed.

## 5.8 Future Work

As in any software package, there is plenty of room for improvements. For instance:

- Adding support for other security mechanisms in addition to UIUC SESAME to increase interoperability among the applications the employ SGSS-API.
- SGSS-API v1.0a and UIUC SESAME supports the asymmetric key distribution scheme. It may be worthwhile to add support for other schemes supported by SESAME including *symmetric intra-domain*, and *hybrid inter-domain*.
- Testing the interoperability between SGSS-API over UIUC SESAME with other GSS-APIs that support SESAME.
- Improved support for delegation.
- Supporting multiple privilege and security attributes and target names per GSS Credential, as well as supporting more types of privilege attributes like primary group, group, and domain defined attributes.

\*

## References

- [1] Internet RFC 1508: “*Generic Security Service Application Program Interface*”, J. Linn, Sept. ‘93.
- [2] Internet RFC 1509: “*Generic Security Service API : C-bindings*”, J. Wray, Sept. ‘93.
- [3] Internet RFC 2078: “*Generic Security Service Application Program Interface, Version 2*”, J. Linn, January ‘97.
- [4] Internet draft: <draft-ietf-cat-sesamemech-02> “*The SESAME V5 GSS-API Mechanism*”, E. Braize, Nov. ‘96.
- [5] Internet draft: <draft-ietf-cat-gssv2-javabind-01> “*Generic Security Services API Version 2: Java Bindings*”, J. Kabat, March ‘99.
- [6] “*Practical Intranet Security*”, P. Ashley and M. Vandenwauver, Kluwer Academic Publishers, Boston.
- [7] “*Distributed Systems Concepts and Designs*”, G. Coulouris, J. Dollimore, T. Kindberg, Addison Wesley.
- [8] “*Using SESAME’s GSS-API to add security to UNIX Applications*”, P. Ashley, M. Rutherford, M. Vandenwauver, S. Bouing.
- [9] Standard ECMA-219 “*Authentication and Privilege Attribute Security Application with related key distribution functions*”, 2<sup>nd</sup> edition, March ‘96.
- [10] “*UIUC SESAME: Achieving a Portable Authentication, Access Control, and Delegation Protocol*”, Monika Chandak, Masters Thesis at CS, UIUC, ‘99.

# Index

## A

ASN.1, 6, 21, 53, 54, 56, 57, 60, 61, 62, 63

## B

basic key, 57

## C

ChannelBinding, 35, 36, 43  
Class ChannelBinding, 34  
Class GSSAttribute, 50  
Class GSSContext, 36  
Class GSSCredential, 27  
Class GSSException, 46  
Class GSSManager, 21  
Class GSSName, 22  
Class MessageProp, 25  
Class Oid, 20  
confidentiality, 3, 8, 9, 11, 25, 26, 41, 43, 45  
credential, 10, 27, 28, 29, 30, 31, 32, 33, 34, 36, 42, 46, 47, 48

## D

DER, 20, 21, 53, 54, 56, 63  
dialogue keys, 57

## G

GSS-API, 3, 4, 6, 9, 10, 11, 22, 27, 34, 46, 48, 65  
GSSAttribute, 34, 46, 50  
GSSContext, 22, 36, 37, 38  
GSSCredential, 22, 28, 29, 30, 31, 32, 33, 36, 46  
GSSException, 20, 22, 23, 24, 25, 28, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 50, 51  
GSSName, 23, 24, 25, 28, 29, 30, 31, 33, 36, 45

## I

integrity, 3, 8, 9, 11, 43, 45, 47

## M

MessageProp, 25, 26, 38, 39, 40, 41

MIC, 2, 13, 14, 39, 40, 41, 47, 53, 62, 63  
mutual authentication, 8, 10, 11, 42, 43

## P

PAC, 4, 53  
Per-Message Services, 13  
Privilege Attribute Certificate, 4, 53  
privilege attributes, 15, 52, 64

## Q

QOP, 13, 17, 25, 26, 39, 40, 41, 48

## S

seal, 56, 62, 63  
security attribute, 34, 46, 50  
security context, 4, 8, 10, 11, 12, 13, 26, 27, 34, 36, 37, 46, 48, 53, 54, 55, 57, 61  
security mechanism, 5, 9  
SESAME, 2, 4, 5, 6, 7, 12, 13, 15, 17, 19, 21, 22, 24, 25, 28, 29, 30, 31, 32, 33, 36, 45, 50, 52, 53, 57, 59, 64, 65  
SGSS-API, 1, 2, 4, 5, 6, 7, 9, 12, 13, 15, 16, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30, 31, 32, 33, 34, 36, 42, 45, 46, 50, 51, 52, 53, 56, 57, 58, 59, 60, 64  
SPKM\_REQ, 58, 59

## T

targetKeyBlock, 57, 58  
token, 10, 11, 13, 14, 17, 18, 19, 25, 27, 36, 37, 38, 39, 40, 41, 48, 49, 52, 53, 54, 55, 61, 62, 63

## U

unwrap, 13, 19, 39, 40

## W

Wrap, 2, 62, 63