z/OS
2.5

*Integrated Security Services Network
Authentication Service Programming*

IBM

Note

Before using this information and the product it supports, read the information in "Notices" on page 267.

Contents

x

Tables

Figures

About this document

This publication describes application programming interfaces (APIs) for z/OS Integrated Security Services Network Authentication Service. It supports z/OS (5650-ZOS).

Who should use this document

This document is for application programmers who want to create interfaces to z/OS Integrated Security Services Network Authentication Service.

How this document is organized

This document is divided into two parts. Part 1 deals with the Kerberos programming interfaces and Part 2 handles GSS-API interfaces. Within Part 1, there is a chapter introducing the use of Kerberos interfaces and two chapters containing the actual interfaces. Part 2 contains an introductory chapter on using GSS-API interfaces, and two chapters of interfaces.

The document also contains a bibliography and an appendix listing the POSIX-based character set. For a glossary of terms for Network Authentication Service, see *z/OS Integrated Security Services Network Authentication Service Administration*.

Where to find more information

Where necessary, this document refers to information in other documents. For complete titles and order numbers for all elements of z/OS, see *z/OS Information Roadmap*.

The companion publication for this document is *z/OS Integrated Security Services Network Authentication Service Administration*, which provides planning, configuration, and administration information for the product.

Internet sources

The softcopy z/OS publications are also available for web browsing, and PDF versions for viewing or printing from the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

You can also provide comments about this document and any other z/OS documentation by visiting that URL. Your feedback is important in helping to provide the most accurate and high-quality information.

Conventions used in this document

This document uses the following typographic conventions:

Table 1. Typographic conventions	
Font style or characters	**Explanation**
Boldface	Indicates the name of: • The item you need to select • A field, option, parameter, or command • A new term
Italic	Indicates document titles or variable information that must be replaced by an actual value.

Table 1. Typographic conventions (continued)	
Font style or characters	**Explanation**
Monofont	Indicates: • Names of directories, files, and user IDs • Information displayed by the system • An example • A portion of a file or sample code • A previously entered value.
Bold Monofont	Indicates information that you type into the system exactly as it appears in this document.
[ ]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list of required items, in format and syntax descriptions, from which you must select one.
|	A vertical bar separates items in a list of choices.
< >	Angle brackets enclose the name of a key on the keyboard.
...	Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.
\	A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character as the last nonblank character on the line to be continued, and continue the command on the next line.

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xix.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
> If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Documentation
> If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation at ibmdoc@us.ibm.com.

Feedback on the z/OS product documentation and content
> If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

> To help us better process your submission, include the following information:

> - Your name, company/university/institution name, and email address
> - The following deliverable title and order number: z/OS Integrated Security Services Network Authentication Service Programing, SC23-6787-50
> - The section title of the specific information to which your comment relates
> - The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

Summary of changes for z/OS Version 2 Release 5 (V2R5)

The following changes are made to z/OS Version 2 Release 5 (V2R5).

New information

The following information is new:

September 2021
There is no new content for this release.

Changed information

The following information has changed.

September 2021
There is no changed content for this release.

Deleted information

The following information has been deleted:

September 2021
No content has been deleted from this release.

Summary of changes for z/OS Version 2 Release 4 (V2R4)

The following changes are made to z/OS Version 2 Release 4 (V2R4).

New

- The following API is new:
  - "krb5_set_fast_armor_ticket (set the armor ticket for use in FAST pre-authentication)" on page 141
- The following API's have been updated to include two new cksumtypes:
  - "krb5_auth_con_set_req_cksumtype (set checksum type)" on page 18
  - "krb5_auth_con_set_safe_cksumtype (set application method checksum type)" on page 19
- The params parameter description has been updated in the following API:
  "krb5_c_string_to_key_with_params (generate encryption key from text string with params)" on page 36
- The following API's have been updated to include two new enctype parameters:
  - "krb5_get_in_tkt_system (get initial KDC ticket)" on page 85
  - "krb5_get_in_tkt_with_keytab (get initial ticket using key table)" on page 87
  - "krb5_get_in_tkt_with_password (get initial ticket with text password)" on page 89
  - "krb5_get_in_tkt_with_skey (get initial ticket using session key)" on page 93
- The following API's have been updated to include two new ktype parameters:
  - "krb5_set_default_in_tkt_ktypes (set default encryption types)" on page 139
  - "krb5_set_default_tgs_ktypes (set default encryption types)" on page 140

Changed

- The following API has been updated:
  -
  -
  -
  -
  -
  -
  -
  -

Deleted information

No information has been deleted from this version.

Summary of changes for z/OS Version 2 Release 3 (V2R3)

The following changes are made to z/OS Version 2 Release 3 (V2R3).

New

- A new function_mask value for KRB5_API_LVL7 has been added to the .

Changed

- APAR OA54527 updates the usage section in the following API's with a note in regards to using the function in a 31 bit application:
  -
  -
- The following API's have parameters that have been updated to include FIPS support:
  -
  -
  -
  -
  -
  -
  -
  -
  -
  -
  -
  -
  -
  -
  -

- "krb5_set_default_tgs_ktypes (set default encryption types)" on page 140
- The Usage section of the following APIs has been updated to include FIPS support:
  - "krb5_get_default_in_tkt_ktypes (return default encryption type)" on page 82
  - "krb5_get_default_tgs_ktypes (return KDC default encryption types)" on page 83
  - "krb5_init_context (create Kerberos context)" on page 97
- The following new APIs are added to the documentation in support of time values beyond year 2038:
  - "krb5_timeofday64 (return current time of day)" on page 144
  - "krb5_us_timeofday64 (return current time of day)" on page 147

Deleted information

No information has been deleted from this version.

Part 1. Kerberos interfaces

This Part introduces the Kerberos application programming interfaces (APIs) and describes each one. These topics are covered:

- Introduction to Kerberos

  - Kerberos basics
  - Using Kerberos files
  - Using Kerberos services
- Kerberos programming interfaces
- Kerberos administration programming interfaces

Chapter 1. Introduction to Kerberos

Integrated Security Services Network Authentication Service for z/OS is based on Kerberos Version 5. This chapter describes the routines that make up the Kerberos Version 5 application programming interface. The description is oriented towards programmers who already have a basic familiarity with Kerberos and are in the process of including Kerberos authentication as part of applications being developed.

Fore more details, refer to the list of Internet Request for Comment (RFC) documents in *z/OS Integrated Security Services Network Authentication Service Administration*.

There is a glossary of terms for Network Authentication Service in *z/OS Integrated Security Services Network Authentication Service Administration*.

Kerberos basics

Kerberos performs authentication as a trusted third-party authentication service by using conventional shared secret key cryptography. Kerberos provides a means of verifying the identities of principals, without relying on authentication by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets traveling along the network can be read, modified, and inserted at will.

The two methods for obtaining credentials, the initial-ticket exchange and the ticket-granting-ticket exchange, use slightly different protocols and require different Application Programming Interface (API) routines.

The basic difference an application programmer sees is that the initial-ticket exchange does not require a ticket-granting-ticket (TGT) but does require the client's secret key. Usually, the initial-ticket exchange is for a TGT, and TGT exchanges are used from then on. In a TGT exchange, the TGT is sent as part of the request for a ticket and the reply is encrypted in the session key obtained from the TGT. Thus, once a user's password is used to obtain the initial TGT, it is not required for subsequent TGT exchanges to obtain additional tickets.

A *ticket-granting ticket* contains the Kerberos server (**krbtgt/realm**) as the server name. A *service ticket* contains the application server as the server name. A ticket-granting ticket is used to obtain service tickets. In order to obtain a service ticket for a server in another realm, the application must first obtain a ticket-granting ticket to the Kerberos server for that realm.

The Kerberos server reply consists of a ticket and a session key, encrypted either in the user's secret key or the TGT session key. The combination of a ticket and a session key is known as a set of *credentials*. An application client can use these credentials to authenticate to the application server by sending the ticket and an *authenticator* to the server. The authenticator is encrypted in the session key of the ticket and contains the name of the client, the name of the server, and the time the authenticator was created.

In order to verify the authentication, the application server decrypts the ticket using its service key, which is known only by the application server and the Kerberos server. Inside the ticket, the Kerberos server has placed the name of the client, the name of the server, a session key associated with the ticket, and some additional information.

The application server then uses the ticket session key to decrypt the authenticator and verifies that the information in the authenticator matches the information in the ticket. The server also verifies that the authenticator timestamp is recent to prevent replay attacks (the default is 5 minutes). Since the session key was generated randomly by the Kerberos server and delivered encrypted in the service key and a key known only by the user, the application server can be confident that users really are who they claim to be, by virtue of the fact that the user was able to encrypt the authenticator in the correct key.

To provide detection of both replay attacks and message stream modification attacks, the integrity of all the messages exchanged between principals can also be guaranteed by generating and transmitting a collision-proof checksum of the client's message, keyed with the session key. Privacy and integrity of the

message exchanged between principals can be secured by encrypting the data to be passed using the session key.

The purpose of realms

The Kerberos protocol is designed to operate across organizational boundaries. Each organization wanting to run a Kerberos server establishes its own *realm*. The name of the realm in which a client is registered is part of the client's name and can be used by the application server to decide whether to honor a request.

By establishing *inter-realm keys*, the administrators of two realms can allow a client authenticated in one realm to use its credentials in the other realm. The exchange of inter-realm keys registers the ticket-granting service of each realm as a principal in the other realm. A client is then able to obtain a ticket-granting ticket for the remote realm's ticket-granting service from its local ticket-granting service. Tickets issued to a service in the remote realm indicate that the client was authenticated from another realm.

This method can be repeated to authenticate throughout an organization across multiple realms. To build a valid authentication path to a distant realm, the local realm must share an inter-realm key with the target realm or with an intermediate realm that communicates with either the target realm or with another intermediate realm.

Realms are typically organized hierarchically. Each realm shares a key with its parent and a different key with each child. If an inter-realm key is not directly shared by two realms, the hierarchical organization allows an authentication path to be easily constructed. If a hierarchical organization is not used, it may be necessary to consult some database in order to construct an authentication path between realms.

Although realms are typically hierarchical, intermediate realms may be bypassed to achieve cross-realm authentication through alternate authentication paths. It is important for the end-service to know which realms were transited when deciding how much faith to place in the authentication process. To facilitate this decision, a field in each ticket contains the names of the realms that were involved in authenticating the client.

Assumptions about the environment

Kerberos has certain limitations that should be kept in mind when designing security measures:

- Kerberos does not address 'denial of service' attacks. There are places in these protocols where an intruder can prevent an application from participating in the proper authentication steps. Detection and solution of such attacks (some of which can appear to be 'usual' failure modes for the system) is usually best left to human administrators and users.
- Principals must keep their secret keys secret. If an intruder steals a principal's key, it can then masquerade as that principal or impersonate any server to the legitimate principal.
- 'Password guessing' attacks are not solved by using Kerberos. If a user chooses a poor password, it is possible for an attacker to successfully mount an offline dictionary attack by repeatedly attempting to decrypt messages that are encrypted under a key derived from the user's password.

Using Kerberos files

The Kerberos runtime uses three types of files during its processing: credentials cache, replay cache, and key table. Each type of file has a set of API routines to manage and manipulate the file.

Credentials cache

The credentials cache holds Kerberos credentials (tickets, session keys, and other identifying information) in a semi-permanent store. The Kerberos runtime reads credentials from the cache as they are needed and stores new credentials in the cache as they are obtained. This way, the application does not have to manage the credentials itself.

Kerberos supports three types of credentials caches: FILE, MEMORY, and XMEM. The default credentials cache type is FILE.

- A FILE credentials cache is maintained in an HFS file and can be shared between applications. The credentials cache files are located in **/var/skrb/creds**. This directory can be shared by multiple systems in the sysplex (Kerberos uses global resource serialization to serialize access to the credentials cache file). A unique filename is generated each time a new credentials cache file is created. These credentials cache files persist until they are deleted (the **kinit** command deletes the current default credentials cache file for a user when it creates a new default credentials cache). The **kdestroy** command with the **-e** option can be used to remove expired credentials cache files.

- A MEMORY credentials cache is maintained in storage and can be accessed only by the application that created it. The credentials cache does not persist when the application terminates.

- An XMEM credentials cache is maintained in a data space by the Kerberos security server. The credentials cache can be read from any system in the sysplex but can be updated only from the system that created the credentials cache. The credentials cache does not persist when the Kerberos security server terminates. The Kerberos security server periodically deletes credentials caches that contain only expired credentials. The MODIFY SKRBKDC,DISPLAY CREDS command can be used to display the current contents of the credentials data space.

Replay cache

The replay cache is used to detect duplicate requests. Each time a request is processed by the Kerberos runtime, an entry is made in the replay cache. If a later request is processed that matches an entry already in the replay cache, an error is returned to the application program. The replay cache is periodically purged to remove stale entries (a stale entry occurs when the lifetime of the associated request has expired).

Kerberos supports two types of replay caches: *dfl* and *mem*. The *dfl* replay cache is maintained in a file and persists across application restarts. The *mem* replay cache is maintained in memory and does not exist after the application has ended. The replay cache should not be shared between applications since this could result in false replay errors caused by different requests with the same timestamp.

The **use_dvipa_override** configuration option, when set to 1, will allow principals to be shared by applications across the sysplex which means that the replay cache needs to be shared across the sysplex as well but the replay caches mentioned cannot be shared. Therefore, when the **use_dvipa_override** configuration option is set to 1, the selected, or defaulted, replay cache will be overridden by one that is controlled by the SKRBKDC started task. This requires the SKRBDC started task to be running on every image where Kerberized applications run. If it is not, the **use_dvipa_override** configuration option will be rest back to 0.

Key table

The key table is used to store encryption keys. This is generally used by server applications to provide the encryption keys for use by the Kerberos runtime when it needs to decrypt a request received from a client application. Each key has an associated version number, and the version is incremented each time the key is changed. When a service ticket is encrypted by the key distribution center (KDC), it uses the latest encryption key stored in the Kerberos database and records the key version number in the ticket. Then, when the ticket is presented to the server, the key version number is used to retrieve the proper key from the key table. This allows the server to change its key without invalidating existing tickets.

Kerberos supports two types of key tables: FILE and WRFILE. Both of these key table types refer to the same file-based key table. The difference is that a key table opened as FILE is read-only while a key table opened as WRFILE can be read and written. The key table can be shared by multiple applications.

Using Kerberos services

The **krb5_context** opaque data type represents the current Kerberos context. Each application must have at least one Kerberos context. The Kerberos context contains configuration data obtained from the Kerberos configuration file, as well as override values that have been set by the application. A single Kerberos context may be shared by multiple threads in the same process but may not be shared between processes. The **krb5_init_context()** API routine is used to create a Kerberos context.

The **krb5_auth_context** opaque data type represents a Kerberos authentication context. The Kerberos authentication context is used by message service routines. Each client-server connection must have its own authentication context because sequence numbers, encryption keys, check sums, and authenticators are stored in the context. If an authentication context is shared between threads, the application must provide concurrency control so that the context is not accessed by more than one thread at a time. The **krb5_auth_con_init()** API routine is used to create a Kerberos authentication context.

In order to properly handle code pages, the **setlocale()** routine must be called before any Kerberos API routines are called. This insures that the proper code page is set. Kerberos does not support double-byte or bi-directional character sets. In addition, it is strongly recommended that principal and realm names consist of characters from the POSIX character set. See Appendix A, "POSIX-based portable character set," on page 259 for a table showing the POSIX character set.

The Kerberos API does not establish its own signal handlers since this could conflict with the application's use of signals (signal handlers have a process-wide scope). Consequently, the application should set up its own signal handler for the SIGPIPE signal. The action routine can be SIG_IGN unless the application needs to perform its own processing for a broken pipe.

In order to compile, link, and run a Kerberos application, you must:

- Define the S390 compiler variable (**-D S390**) when compiling your application
- Specify the DLL option to the compiler (**-Wc,DLL**)
- Specify the DLL option to the binder (**-Wl,DLL**)
- Link with the **libskrb.a** library and include the EUVFKDLL.x side file. **For a 64-bit application**, link with the libskrb64.a library and include the EUVFKD64.x side file.
- Ensure that the EUVFKDLL load module is in the load module search list. **For a 64-bit application**, ensure that the EUVFKD64 load module is in the load module search list
- Ensure that LE run-time at this or higher release level is available.
- Ensure that POSIX(ON) is in effect as an LE runtime option.

If you are compiling with Job Control Language (JCL), and you choose to specify the PDS version of the header files required by Kerberos and/or GSS services, specify SYS1.SIEAHDR.H with no + or * in your SEARCH compiler option. Alternatively, include SYS1.SIEAHDR.H in your compile step SYSLIB DD concatenation. Refer to the section on "Forming PDS with LSEARCH | SEARCH Options with No +" and to the rules on searching include files in the *z/OS C/C++ User's Guide* (SC09-4767) for more details.

If you are linking with JCL, include the relevant EUVF prefixed export files provided in SYS1.SIEASID such as EUVFKDLL (or EUVFKD64 for a 64 bit application) during the link. EUVF.SEUVFLIB is the PDS equivalent to libskrb.a and is required during the link if you want to include functions not provided in the DLLs provided in SYS1.SIEALNKE, for example if you are calling krb5_dll_load in order to do an explicit load of the Kerberos run-time library.

SYS1.SIEALNKE is required in the search chain during the run step in order to access the provided EUVF prefixed DLLs.

Contact your system administrator if you need READ access to SYS1.SIEAHDR.H, SYS1.SIEASID, EUVF.SUEVFLIB, or SYS1.SIEALNKE.

Chapter 2. Kerberos programming interfaces

This chapter presents the Kerberos programming interfaces in alphabetical order. It provides the purpose, format, parameters, and use of each.

krb5_address_compare (compare two Kerberos addresses)

Purpose

Compares two Kerberos addresses.

Format

```
#include <skrb/krb5.h>
krb5_boolean krb5_address_compare (
    krb5_context                     context,
    const krb5_address *             addr1,
    const krb5_address *             addr2)
```

Parameters

Input

context
  Specifies the Kerberos context.

addr1
  Specifies the first address.

addr2
  Specifies the second address.

Usage

The **krb5_address_compare()** routine compares two Kerberos addresses and returns TRUE if they are the same and FALSE otherwise. An IPv6 address that maps an IPv4 address is considered to be equal to the IPv4 address (a mapped IPv6 address consists of 10 bytes of 0, 2 bytes of 255, and the 4-byte IPv4 address).

krb5_address_search (search for address in address book)

Purpose

Determine if an address is present in an address list.

Format

```
#include <skrb/krb5.h>
krb5_boolean krb5_address_search (

    krb5_context                     context,

    const krb5_address *             addr,

    krb5_address * const *           addrlist)
```

Parameters

Input

context
　　Specifies the Kerberos context.

addr
　　Specifies the search address.

addrlist
　　Specifies the address list as an array of addresses. The last entry in the array must be a NULL pointer. Specify NULL for this parameter if no address list is present.

Usage

The krb5_address_search() routine determines if an address is present in an address list.

The function return value is TRUE if the address is found in the address list or if no address list was provided. The function return value is FALSE otherwise.

krb5_auth_con_free (release an authentication context)

Purpose

Releases an authentication context.

Format

```
#include <skrb/krb5.h>

krb5_error_code krb5_auth_con_free (

    krb5_context            context,

    krb5_auth_context       auth_context)
```

Parameters

Input

context
　　Specifies the Kerberos context.

auth_context
　　Specifies the authentication context.

Usage

The **krb5_auth_con_free()** routine releases an authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_genaddrs (generate local and remote network addresses)

Purpose

Generates local and remote network addresses.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_genaddrs (

    krb5_context                    context,

    krb5_auth_context               auth_context,
    int                             fd,
    int                             flags)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

fd
Specifies the socket descriptor to use.

flags
Specifies the address generation flags as follows:

- KRB5_AUTH_CONTEXT_GENERATE_LOCAL_ADDR - Generate the local network address.
- KRB5_AUTH_CONTEXT_GENERATE_LOCAL_FULL_ADDR - Generate the local network address and the local port.
- KRB5_AUTH_CONTEXT_GENERATE_REMOTE_ADDR - Generate the remote network address.
- KRB5_AUTH_CONTEXT_GENERATE_REMOTE_FULL_ADDR - Generate the remote network address and the remote port.

Usage

The **krb5_auth_con_genaddrs()** routine generates the local and remote network addresses represented by a socket connection. These addresses are stored in the authentication context and the can be retrieved by calling the **krb5_auth_con_getaddrs()** routine.

The socket must have been created using the AF_INET or AF_INET6 address family. The socket must be in the connected state if the remote network address is to be generated. An IPv6 address representing a mapped IPv4 address is generated as an IPv4 address.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getaddrs (return local and remote network addresses)

Purpose

Returns the local and remote network addresses stored in the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getaddrs (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_address **                 local_addr,
    krb5_address **                 remote_addr)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

Output

local_addr
Returns the local network address. Specify NULL for this parameter if there is no local network address. The return value is NULL if the local network address has not been set. The **krb5_free_address()** routine should be called to release the address when it is no longer needed.

remote_addr
Returns the remote network address. Specify NULL for this parameter if there is no remote network address. The return value is NULL if the remote network address has not been set. The **krb5_free_address()** routine should be called to release the address when it is no longer needed.

Usage

The **krb5_auth_con_getaddrs()** routine returns the local and remote network addresses stored in the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code

krb5_auth_con_getauthenticator (return authenticator)

Purpose

Returns the authenticator from the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getauthenticator (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_authenticator **           authent)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

Output

authent
Returns the authenticator. The **krb5_free_authenticator()** routine should be called to release the authenticator when it is no longer needed.

Usage

The **krb5_auth_con_getauthenticator()** routine returns the authenticator used during mutual authentication.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getflags (return current authentication flags)

Purpose

Returns the current authentication context flags.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getflags (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_int32 *                    flags)
```

Parameters

Input

context
    Specifies the Kerberos context.

auth_context
    Specifies the authentication context.

Output

flags
    Returns the current flags. The following symbolic definitions are provided for the flag bits:

- KRB5_AUTH_CONTEXT_DO_TIME - Use timestamps in messages.
- KRB5_AUTH_CONTEXT_RET_TIME - Return timestamps to application.
- KRB5_AUTH_CONTEXT_DO_SEQUENCE - Use sequence numbers in messages.
- KRB5_AUTH_CONTEXT_RET_SEQUENCE - Return sequence numbers to application.

Usage

The **krb5_auth_con_getflags()** routine returns the current authentication context flags.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getivector (return address of initial vector)

Purpose

Returns the address of the initial vector in the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getivector (
    krb5_context                        context,
```

```
    krb5_auth_context                          auth_context,
    krb5_pointer *                             ivec)
```

Parameters

Input

context
>   Specifies the Kerberos context.

auth_context
>   Specifies the authentication context.

Output

ivec
>   Returns the address of the initial vector. The authentication context still points to this vector, so any changes made to the vector will affect future data encryption operations performed using the authentication context.

Usage

The **krb5_auth_con_getivector()** routine returns the address of the initial vector used by the specified authentication context. The application can then use this address to change the contents of the initial vector. However, the application must not free the storage represented by the initial vector.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getkey (retrieve encryption key)

Purpose

Retrieves the encryption key stored in the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getkey (
    krb5_context                          context,
    krb5_auth_context                     auth_context,
    krb5_keyblock **                      keyblock)
```

Parameters

Input

context
>   Specifies the Kerberos context.

auth_context
>   Specifies the authentication context.

Output

keyblock
>   Returns a keyblock containing the encryption key. The **krb5_free_keyblock()** routine routine should be called to release the keyblock when it is no longer needed.

Usage

The **krb5_auth_con_getkey()** routine returns the current encryption key stored in the authentication context. This is normally the session key that was obtained from an application request message.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getlocalseqnumber (return local message sequence)

Purpose

Returns the local message sequence number from the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getlocalseqnumber (
    krb5_context                 context,
    krb5_auth_context            auth_context,
    krb5_int32 *                 seqnum)
```

Parameters

Input

context
> Specifies the Kerberos context.

auth_context
> Specifies the authentication context.

Output

seqnum
> Returns the message sequence number.

Usage

The **krb5_auth_con_getlocalseqnumber()** routine returns the local message sequence number. Sequence numbers are used when generating messages if the KRB5_AUTH_CONTEXT_DO_SEQUENCE flag has been set in the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getlocalsubkey (return local subsession key)

Purpose

Returns the local subsession key from the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getlocalsubkey (
    krb5_context                 context,
    krb5_auth_context            auth_context,
    krb5_keyblock **             keyblock)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

Output

keyblock
Returns the subsession key. The **krb5_free_keyblock()** routine should be called to release the keyblock when it is no longer needed.

Usage

The **krb5_auth_con_getlocalsubkey()** routine returns the local subsession key from the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getports (return local and remote network ports)

Purpose

Returns the local and remote network ports stored in the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getports (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_address **                 local_port,
    krb5_address **                 remote_port)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

Output

local_port
Returns the local network port. Specify NULL for this parameter if the local network port is not required. The return value is NULL if the local network port has not been set. The **krb5_free_address()** routine should be called to release the address when it is no longer needed.

remote_port
Returns the remote network port. Specify NULL for this parameter if the remote network port is not required. The return value is NULL if the remote network port has not been set. The **krb5_free_address()** routine should be called to release the address when it is no longer needed.

Usage

The **krb5_auth_con_getports()** routine returns the local and remote network ports stored in the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getrcache (return replay cache)

Purpose

Returns the replay cache for the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getrcache (
    krb5_context                        context,
    krb5_auth_context                   auth_context,
    krb5_rcache *                       rcache)
```

Parameters

Input

context
    Specifies the Kerberos context.

auth_context
    Specifies the authentication context.

Output

rcache
    Returns the replay cache handle.

Usage

The **krb5_auth_con_getrcache()** function returns the replay cache for the authentication context. A replay cache is used when processing a message in order to detect message replay. A replay cache must be set in the authentication context if message timestamps are being used.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getremoteseqnumber (return remote message sequence number)

Purpose

Returns the remote message sequence number from the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getremoteseqnumber (
    krb5_context                        context,
    krb5_auth_context                   auth_context,
    krb5_int32 *                        seqnum)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

Output

seqnum
Returns the message sequence number.

Usage

The **krb5_auth_con_getremoteseqnumber()** routine returns the remote message sequence number. Sequence numbers are used when generating messages if the KRB5_AUTH_CONTEXT_DO_SEQUENCE flag has been set in the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_getremotesubkey (return remote subsession key)

Purpose

Returns the remote subsession key from the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_getremotesubkey (
    krb5_context                     context,
    krb5_auth_context                auth_context,
    krb5_keyblock **                 keyblock)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

Output

keyblock
Returns the subsession key. The **krb5_free_keyblock()** routine should be called to release the keyblock when it is no longer needed.

Usage

The **krb5_auth_con_getremotesubkey()** routine returns the remote subsession key from the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_init (create an authentication context)

Purpose

Creates an authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_init (
    krb5_context                    context,
    krb5_auth_context *             auth_context)
```

Parameters

Input

context
    Specifies the Kerberos context.

Output

auth_context
    Returns the authentication context created by this call. The **krb5_auth_con_free()** routine should be
    called to release the authentication context when it is no longer needed.

Usage

The **krb5_auth_con_init()** routine creates an authentication context. An authentication context contains
information relating to a single connection between two applications. The context is initialized to enable
the use of the replay cache (KRB5_AUTH_CONTEXT_DO_TIME) but to disable the use of message
sequence numbers. The **krb5_auth_con_setflags()** routine can be used to change these defaults. If
using KRB-SAFE or KRB-PRIV messages, you should have one of the above settings on to prevent replay
attacks.

The **krb5_auth_con_free()** routine should be used to release the authentication context when it is no
longer needed.

The Kerberos runtime provides no concurrency control for the authentication context. If the application
wants to use the same authentication context in multiple threads, it is the responsibility of the
application to serialize access to the authentication context so that just a single thread is accessing
the authentication context at any time. Because message sequence numbers are contained in the
authentication context, this serialization must be extended to encompass the message exchange between
the two applications. Otherwise, message sequence errors are liable to occur if the messages are
delivered out of sequence.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_initivector (allocate initial encryption vector)

Purpose

Allocates the initial encryption vector in the authentication context and sets it to zero.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_initivector (
    krb5_context                    context,
    krb5_auth_context               auth_context)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

Usage

The **krb5_auth_con_initivector()** routine allocates the initial vector in the authentication context and sets it to zero. The authentication context must already contain an encryption key that defines the type of encryption to be used. The initial vector is used to initialize the encryption sequence each time a message is encrypted. This serves to generate different encrypted results for the same message contents and encryption key

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_set_req_cksumtype (set checksum type)

Purpose

Sets the checksum type used to generate an application request message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_set_req_cksumtype (
    krb5_context                  context,
    krb5_auth_context             auth_context,
    krb5_cksumtype                cksumtype)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

cksumtype
Specifies the checksum type as follows:

- CKSUMTYPE_CRC32 - DES CRC checksum (not valid in FIPS mode)
- CKSUMTYPE_DESCBC - DES CBC checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD4 - MD4 checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD4_DES - DES MD4 checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD5 - MD5 checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD5_DES - DES MD5 checksum (not valid in FIPS mode)
- CKSUMTYPE_NIST_SHA - NIST SHA checksum (not valid in FIPS mode)
- CKSUMTYPE_HMAC_SHA1_DES3 - DES3 HMAC checksum
- CKSUMTYPE_HMAC_SHA1_96_AES128 - AES SHA1 checksum
- CKSUMTYPE_HMAC_SHA1_96_AES256 - AES SHA1 checksum
- CKSUMTYPE_HMAC_SHA256_128_AES128 - AES SHA2 checksum

- CKSUMTYPE_HMAC_SHA384_192_AES256 - AES SHA2 checksum

Usage

The **krb5_auth_con_set_req_cksumtype()** routine sets the checksum type to be used by the **krb5_mk_req()** routine. This overrides the default value set by the **ap_req_checksum_type** entry in the Kerberos configuration file. In addition, if an AES128, AES256, or DES3 based checksum type is specified, the checksum type must be compatible with the encryption key in the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_set_safe_cksumtype (set application method checksum type)

Purpose

Sets the checksum type used to generate a signed application message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_set_safe_cksumtype (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_cksumtype                  cksumtype)
```

Parameters

Input

context
    Specifies the Kerberos context.

auth_context
    Specifies the authentication context.

cksumtype
    Specifies the checksum type as follows:

- CKSUMTYPE_NULL - Set the checksum algorithm to NULL to indicate that the checksum being used is based upon the encryption key stored in the authentication context
- CKSUMTYPE_DESCBC - DES CBC checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD4_DES - DES MD4 checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD5_DES - DES MD5 checksum (not valid in FIPS mode)
- CKSUMTYPE_HMAC_SHA1_DES3 - DES3 HMAC checksum
- CKSUMTYPE_HMAC_SHA1_96_AES128 - AES SHA1 checksum
- CKSUMTYPE_HMAC_SHA1_96_AES256 - AES SHA1 checksum
- CKSUMTYPE_HMAC_SHA256_128_AES128 - AES SHA2 checksum
- CKSUMTYPE_HMAC_SHA384_192_AES256 - AES SHA2 checksum

Usage

The **krb5_auth_con_set_req_cksumtype()** routine sets the checksum type to be used by the **krb5_mk_safe()** routine. This overrides the default value set by the *ap_safe_checksum_type* entry in the Kerberos configuration file. The **krb5_mk_safe()** function requires a keyed checksum. In addition, the checksum must be compatible with the encryption key in the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_setaddrs (set local and remote address values)

Purpose

Sets the local and remote address values in the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_setaddrs (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_address *                  local_addr,
    krb5_address *                  remote_addr)
```

Parameters

Input

context
  Specifies the Kerberos context.

auth_context
  Specifies the authentication context.

local_addr
  Specifies the local network address. Specify NULL for this parameter if there is no local network address.

remote_addr
  Specifies the remote network address. Specify NULL for this parameter if there is no remote network address.

Usage

The **krb5_auth_con_setaddrs()** routine sets the local and remote network address values in the authentication context. These values are used when obtaining tickets and constructing authenticators. A mapped IPv6 address is stored in the authentication context as the corresponding IPv4 address.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_setflags (set authentication context flags)

Purpose

Sets the authentication context flags.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_setflags (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_int32                      flags)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

flags
Specifies the context flags. The following symbolic definitions are provided for the flag bits:

- KRB5_AUTH_CONTEXT_DO_TIME - Use timestamps in messages.
- KRB5_AUTH_CONTEXT_RET_TIME - Return timestamps to application.
- KRB5_AUTH_CONTEXT_DO_SEQUENCE - Use sequence numbers in messages.
- KRB5_AUTH_CONTEXT_RET_SEQUENCE - Return sequence numbers to application.

Usage

The **krb5_auth_con_setflags()** routine sets the authentication context flags.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_setivector (set initial encryption vector)

Purpose

Sets the initial encryption vector in the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_setivector (
    krb5_context                        context,
    krb5_auth_context                   auth_context,
    krb5_pointer                        ivec)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

ivec
Specifies the initial vector.

Usage

The **krb5_auth_con_setivector()** routine sets the initial vector in the authentication context. A copy is not made of the initial vector, so the application must not change or free the buffer specified by the **ivec** parameter until either a new initial vector is set or the authentication context is released. The initial vector is used to initialize the encryption sequence each time a message is encrypted. This serves to generate different encrypted results for the same message contents and encryption key.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_setports (set local and remote network ports)

Purpose

Sets the local and remote network ports in the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_setports (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_address *                  local_port,
    krb5_address *                  remote_port)
```

Parameters

Input

context
 Specifies the Kerberos context.

auth_context
 Specifies the authentication context.

local_port
 Specifies the local network port. Specify NULL for this parameter if there is no local network port.

remote_port
 Specifies the remote network port. Specify NULL for this parameter if there is no remote network port.

Usage

The **krb5_auth_con_setports()** routine sets the local and remote network ports in the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_setrcache (set replay cache)

Purpose

Sets the replay cache for the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_setrcache (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_rcache                     rcache)
```

Parameters

Input

context
 Specifies the Kerberos context.

auth_context
 Specifies the authentication context.

rcache
Specifies the replay cache handle.

Usage

The **krb5_auth_con_setrcache()** function sets the replay cache for the authentication context. A replay cache is used when processing a message in order to detect message replay. A replay cache must be set in the authentication context if message timestamps are being used. The **krb5_rc_default()** and **krb5_rc_resolve()** routines can be used to obtain a replay cache handle.

The replay cache must not be closed by the application while it is in use by the authentication context. The **krb5_auth_con_free()** routine closes the replay cache. The application can use the same replay cache with multiple authentication contexts by calling **krb5_auth_con_setrcache()** with a NULL replay cache handle before calling **krb5_auth_con_free()** to free the authentication context. This leaves the replay cache open and available for use by the application.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_con_setuseruserkey (set user-to-user key)

Purpose

Sets the user-to-user key in the authentication context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_con_setuseruserkey (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_keyblock *                 keyblock)
```

Parameters

Input

context
Specifies the Kerberos context.

auth_context
Specifies the authentication context.

keyblock
Specifies the user key.

Usage

The **krb5_auth_con_setuseruserkey()** routine sets the user key in the authentication context. This is useful only prior to calling the **krb5_rd_req()** routine for user-to-user authentication where the server has the key and needs to use it to decrypt the incoming request. Once the request has been decrypted, this key is no longer necessary and is replaced in the authentication context with the session key obtained from the decoded request.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_auth_to_rep (convert Kerberos authenticator to replay entry)

Purpose

Converts Kerberos authenticator to replay entry.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_auth_to_rep (
    krb5_context                    context,
    krb5_tkt_authent *              authent,
    krb5_donot_replay *             replay)
```

Parameters

Input

context
Specifies the Kerberos context.

authent
Specifies the Kerberos authenticator.

Output

replay
Returns the replay entry data. The **krb5_rc_free_entry_contents()** routine should be called to release the entry data when it is no longer needed.

Usage

The **krb5_auth_to_rep()** routine extracts information from ticket authentication data and builds a replay cache entry. This entry can then be used to check for ticket replay by calling the **krb5_rc_store()** routine to save the entry in the replay cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_build_principal (build a kerberos principal)

Purpose

Builds a Kerberos principal from component strings.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_build_principal (
    krb5_context                    context,
    krb5_principal *                ret_princ,
    int                             realm_len,
    krb5_const char *               realm,
    char *                          name1, name2, ...)
```

Parameters

Input

context
Specifies the Kerberos context.

realm_len
Specifies the length of the realm name.

realm
Specifies the realm name.

namen
> One or more name components. The end of the components is indicated by specifying NULL for the parameter.

Output

ret_princ
> Returns the Kerberos principal. The **krb5_free_principal()** routine should be called to release the principal when it is no longer needed.

Usage

The **krb5_build_principal()** routine creates a Kerberos principal from its component strings.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

For example, to create the principal **bambi/admin@forest**, make the following call:

```
retval = krb5_build_principal(context, &princ, 6, "forest",
               "bambi", "admin", NULL);
```

krb5_build_principal_ext (build a Kerberos principal)

Purpose

Builds a Kerberos principal from component strings.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_build_principal_ext (
    krb5_context                    context,
    krb5_principal *                ret_princ,
    int                             realm_len,
    krb5_const char *               realm,
    int                             name1_len,
    char *                          name1,
    int                             name2_len,
    char *                          name2, ...)
```

Parameters

Input

context
> Specifies the Kerberos context.

realm_len
> Specifies the length of the realm name.

realm
> Specifies the realm name.

lenn/namen
> One or more name components. Each component consists of its length followed by its value. The end of the components is indicated by specifying a length of zero.

Output

ret_princ
> Returns the Kerberos principal. The **krb5_free_principal()** routine should be called to release the principal when it is no longer needed.

Usage

The **krb5_build_principal_ext()** routine creates a Kerberos principal from its component strings. This routine is similar to the **krb5_build_principal()** routine except that the name component lengths are explicitly specified on the function call.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

For example, to create the principal **bambi/admin@forest**, make the following call:

```
retval = krb5_build_principal_ext(context, &princ, 6, "forest",
                                  5, "bambi", 5, "admin", 0);
```

krb5_build_principal_ext_va (build a Kerberos principal)

Purpose

Build a Kerberos principal from component strings.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_build_principal_ext_va (
    krb5_context                          context,
    krb5_principal *                      ret_princ,
    int                                   realm_len,
    krb5_const char *                     realm,
    va_list                               ap)
```

Parameters

Input

context
    Specifies the Kerberos context.

realm_len
    Specifies the length of the realm name.

realm
    Specifies the realm name.

ap
    A variable argument list consisting of name lengths and character pointers that specify one or more name components. The end of the components is indicated by specifying a name length of zero.

Output

ret_princ
    Returns the Kerberos principal. The **krb5_free_principal()** routine should be called to release the principal when it is no longer needed.

Usage

The **krb5_build_principal_ext_va()** routine creates a Kerberos principal from its component strings. It is similar to the **krb5_build_principal_ext()** routine except the name components are specified as a variable argument list instead of as discrete parameters on the function call.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

For example, assume we have a function *my_func,* which is called with a list of names. It could generate a Kerberos principal from these names as follows:

```
#include <stdarg.h>
#include <skrb/krb5.h>
krb5_error_code my_func(int realm_len, char *realm, ...) {
    va_list ap;
    krb5_error_code retval;
    va_start(ap, realm);
    retval = krb5_build_principal_ext_va(context, &princ,
                                realm_len, realm, ap);
    va_end(ap);
    return retval;
}
int main(int argc, char *argv[]) {
    my_func(6, "forest", 5, "bambi", 5, "admin", 0);
    return 0;
}
```

krb5_build_principal_va (build a Kerberos principal)

Purpose

Builds a Kerberos principal from component strings.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_build_principal_va (
    krb5_context                        context,
    krb5_principal *                    ret_princ,
    int                                 realm_len,
    krb5_const char *                   realm,
    va_list                             ap)
```

Parameters

Input

context
    Specifies the Kerberos context.

realm_len
    Specifies the length of the realm name.

realm
    Specifies the realm name.

ap
    A variable argument list consisting of name lengths and character pointers that specify one or more name components. The end of the components is indicated by specifying NULL for the parameter.

Output

ret_princ
    Returns the Kerberos principal. The **krb5_free_principal()** routine should be called to release the principal when it is no longer needed.

Usage

The **krb5_build_principal_va()** routine creates a Kerberos principal from its component strings. It is similar to the **krb5_build_principal()** routine except the name components are specified as a variable argument list instead of as discrete parameters on the function call.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

For example, assume we have a function *my_func,* which is called with a list of names. It could generate a Kerberos principal from these names as follows:

```
#include <stdarg.h>
#include <skrb/krb5.h>
krb5_error_code my_func(char *realm, ...) {
    va_list ap;
    krb5_error_code retval;
    va_start(ap, realm);
    retval = krb5_build_principal_va(context, &princ,
                                  strlen(realm), realm, ap);
    va_end(ap);
    return retval;
}
int main(int argc, char *argv[]) {
    my_func("forest", "bambi", "admin", NULL);
    return 0;
}
```

krb5_c_block_size (return cipher block size)

Purpose

Returns the cipher block size.

Format

```
#include <skrb/krb5.h>

krb5_error_code krb5_c_block_size (
    krb5_context                     context,
    krb5_enctype                     enctype,
    krb5_size *                      blocksize)
```

Parameters

Input

context
Specifies the Kerberos context.

enctype
Specifies the encryption algorithm.

Output

blocksize
Returns the cipher blocksize for the specified encryption algorithm.

Usage

The **krb5_c_block_size()** routine returns the cipher block size for the indicated encryption algorithm. The encrypted data generated by the **krb5_c_encrypt()** routine is a multiple of the cipher block size. In addition, the clear text input to **krb5_c_encrypt()** is padded with binary zero to a multiple of the cipher block size.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_checksum_length (return checksum length)

Purpose

Returns the checksum length.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_checksum_length (
    krb5_context                      context,
    krb5_cksumtype                    cksumtype,
    krb5_size *        cksumlen)
```

Parameters

Input

context
    Specifies the Kerberos context.

cksumtype
    Specifies the checksum algorithm.

Output

cksumlen
    Returns the length of the checksum data.

Usage

The **krb5_c_checksum_length()** routine returns the length of the checksum for the specified checksum algorithm.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_decrypt (decrypt a data block)

Purpose

Decrypts a data block.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_decrypt (
    krb5_context                      context,
    const krb5_keyblock *             key,
    krb5_keyusage                     usage,
    const krb5_data *                 ivec,
    const krb5_enc_data *             input,
    krb5_data *                       output)
```

Parameters

Input

context
    Specifies the Kerberos context.

key
    Specifies the encryption key. It is validated to ensure it is a supported encryption type. If FIPS mode is on, it will be checked for FIPS compliance.

usage
    Specifies the key usage. This value is used to derive the actual encryption key from the supplied key and allows different message types to use different keys. This parameter is ignored if the specified encryption algorithm does not use key derivation.

ivec
>   Specifies the initial vector. The initial vector provides the starting value for the encryption process. Changing the initial vector causes the encrypted result to be different even when the key and clear text are the same. The length of the initial vector must be the cipher block size as returned by the **krb5_c_block_size()** routine. Specify NULL for this parameter if you do not want to use an initial vector.

input
>   Specifies the data to be decrypted. The *enctype* field for the encrypted data must either match the *enctype* field of the supplied key or must be set to ENCTYPE_NULL. The data to be decrypted must have been encrypted by the **krb5_c_encrypt()** routine using the same key and key usage.

Output

output
>   Specifies the result buffer. The application is responsible for allocating the result buffer. The buffer must be large enough to hold the decrypted data plus any padding (the safest method is to make the result buffer the same length as the encrypted data). Since the clear text is padded to a multiple of the cipher block size during the encryption process, the application must provide a mechanism to determine the actual data length (for example, by including the data length as part of the clear text).

Usage

The **krb5_c_decrypt()** routine decrypts a data block. Due to government export regulations, some encryption algorithms may not be available on the current system. The function return value is set to KRB5_NO_CONF if the requested encryption algorithm is valid but is not available.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_encrypt (encrypt a data block)

Purpose

Encrypts a data block.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_encrypt (
    krb5_context                        context,
    const krb5_keyblock *               key,
    krb5_keyusage                       usage,
    const krb5_data *                   ivec,
    const krb5_data *                   input,
    krb5_enc_data *                     output)
```

Parameters

Input

context
>   Specifies the Kerberos context.

key
>   Specifies the encryption key. It is validated to ensure it is a supported encryption type. If FIPS mode is on, it will be checked for FIPS compliance.

usage
>   Specifies the key usage. This value is used to derive the actual encryption key from the supplied key and allows different message types to use different keys. This parameter is ignored if the specified encryption algorithm does not use key derivation.

ivec
Specifies the initial vector. The initial vector provides the starting value for the encryption process. Changing the initial vector causes the encrypted result to be different even when the key and clear text are the same. The length of the initial vector must be the cipher block size as returned by the **krb5_c_block_size()** routine. Specify NULL for this parameter if you do not want to use an initial vector.

input
Specifies the data to be encrypted. The data is padded on the end with binary zero if the length is not a multiple of the cipher block size.

Output

output
Specifies the result buffer. The application is responsible for allocating the result buffer and setting the *length* and *data* fields. The buffer must be large enough to hold the encrypted data, including confounder, checksum and padding. The required buffer length can be obtained by calling the **krb5_c_encrypt_length()** routine. Upon completion, the *length* field is set to the actual encrypted data length and the *enctype* field is set to the encryption type of the encryption key.

Usage

The **krb5_c_encrypt()** routine encrypts a data block. Due to government export regulations, some encryption algorithms may not be available. The function return value is set to KRB5_NO_CONF if the requested encryption algorithm is valid but is not available on the current system.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_encrypt_length (return encrypted data length)

Purpose

Returns the encrypted data length.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_encrypt_length (
    krb5_context            context,
    krb5_enctype            enctype,
    krb5_size               datalen,
    krb5_size *             enclen)
```

Parameters

Input

context
Specifies the Kerberos context.

enctype
Specifies the encryption algorithm.

datalen
Specifies the length of the data to be encrypted.

Output

enclen
Returns the length of the encrypted data. This length includes confounder, checksum and padding added by the specified encryption algorithm.

Usage

The **krb5_c_encrypt_length()** routine returns the length of the encrypted data which would be generated by the **krb5_c_encrypt()** routine. This value is then used to allocate the result buffer before calling **krb5_c_encrypt()**.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_enctype_compare (compare two encryption types)

Purpose

Compares two encryption types to determine if they are similar.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_enctype_compare (
    krb5_context                        context,
    krb5_enctype                        e1,
    krb5_enctype                        e2,
    krb5_boolean *                      similar)
```

Parameters

Input

context
    Specifies the Kerberos context.

e1
    Specifies the first encryption type.

e2
    Specifies the second encryption type.

Output

similar
    Returns TRUE if the encryption types are similar and FALSE otherwise.

Usage

The **krb5_c_enctype_compare()** routine compares two encryption types. Encryption types are similar if they use the same encryption provider and have the same key generation algorithm. Similar encryption types use the same encryption key. For example, ENCTYPE_DES_CBC_CRC, ENCTYPE_DES_CBC_MD4 and ENCTYPE_DES_CBC_MD5 are similar encryption types.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_keyed_checksum_types (return list of checksum types)

Purpose

Returns a list of keyed checksum types compatible with an encryption type.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_keyed_checksum_types (
    krb5_context                        context,
```

```
    krb5_enctype                            enctype,
    int *                                   count,
    krb5_cksumtype **                       cksumtypes)
```

Parameters

Input

context
>   Specifies the Kerberos context.

enctype
>   Specifies the first encryption type.

Output

count
>   Returns the number of elements in the returned array.

cksumtypes
>   Returns an array of checksum types that are compatible with the specified encryption type. The array
>   should be released when it is no longer needed by calling the **krb5_free_cksumtypes()** routine.

Usage

The **krb5_c_keyed_checksum_types()** routine returns an array of checksum types that are compatible
with the specified encryption type. A checksum type is compatible if it uses an encryption key that
is supported by the specified encryption type. For example, CKSUMTYPE_DESCBC is a compatible
checksum type for the ENCTYPE_DES_CBC_CRC encryption type. A derived key checksum type is
compatible with any encryption type.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_make_checksum (generate checksum for a data block)

Purpose

Generates the checksum for a data block.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_make_checksum (
    krb5_context                            context,
    krb5_cksumtype                          cksumtype,
    const krb5_keyblock *                   key,
    krb5_keyusage                           usage,
    const krb5_data *                       input,
    krb5_checksum *                         cksum)
```

Parameters

Input

context
>   Specifies the Kerberos context.

cksumtype
>   Specifies the checksum type. (Note that its value will be validated for FIPS compliance)

key
>   Specifies the key for a keyed checksum. This parameter is ignored if the specified checksum algorithm
>   does not use an encryption key. Specifies the checksum type as follows:

- CKSUMTYPE_CRC32 - DES CRC checksum (not valid in FIPS mode)
- CKSUMTYPE_DESCBC - DES CBC checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD4 - MD4 checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD4_DES - DES MD4 checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD5 - MD5 checksum (not valid in FIPS mode)
- CKSUMTYPE_RSA_MD5_DES - DES MD5 checksum (not valid in FIPS mode)
- CKSUMTYPE_NIST_SHA - NIST SHA checksum (not valid in FIPS mode)
- CKSUMTYPE_HMAC_SHA1_DES3 - DES3 HMAC checksum
- CKSUMTYPE_HMAC_SHA1_96_AES128 - AES SHA1 checksum
- CKSUMTYPE_HMAC_SHA1_96_AES256 - AES SHA1 checksum
- CKSUMTYPE_HMAC_SHA256_128_AES128 - AES SHA2 checksum
- CKSUMTYPE_HMAC_SHA384_192_AES256 - AES SHA2 checksum

usage
Specifies the key usage. This value is used to derive the actual encryption key from the supplied key and allows different message types to use different keys. This parameter is ignored if the specified checksum algorithm does not use an encryption key or does not use key derivation. Refer to RFC 4120 for usage values reserved for applications.

input
Specifies the data to be used to generate the checksum.

Output

cksum
Returns the generated checksum. The checksum contents should be released when no longer needed by calling the **krb5_free_checksum_contents()** routine.

Usage

The **krb5_c_make_checksum()** routine generates a checksum for the supplied data. The **krb5_c_verify_checksum()** routine can then be used to verify the data integrity.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_make_random_key (generate random encryption key)

Purpose

Generates a random encryption key.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_make_random_key (
    krb5_context                              context,
    krb5_enctype                              enctype,
    krb5_keyblock *                           random_key)
```

Parameters

Input

context
Specifies the Kerberos context.

enctype
> Specifies the encryption type for the generated key. (Note that its value will be validated for FIPS compliance)The following encryption types may be specified:
>
> - ENCTYPE_DES_CBC_CRC - 32-bit CRC checksum with DES encryption. This encryption type should be used for interoperability with older levels of Kerberos V5. (not valid in FIPS mode)
> - ENCTYPE_DES_CBC_MD4 - MD4 checksum with DES encryption. (not valid in FIPS mode)
> - ENCTYPE_DES_CBC_MD5 - MD5 checksum with DES encryption. (not valid in FIPS mode)
> - ENCTYPE_DES_HMAC_SHA1 - SHA1 checksum with DES encryption and key derivation. (not valid in FIPS mode)
> - ENCTYPE_DES3_CBC_SHA1 - SHA1 checksum with DES3 encryption and key derivation.
> - ENCTYPE_AES128_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
> - ENCTYPE_AES256_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
> - ENCTYPE_AES128_CTS_HMAC_SHA256_128 - SHA2 checksum with AES encryption.
> - ENCTYPE_AES256_CTS_HMAC_SHA384_192 - SHA2 checksum with AES encryption.

Output

random_key
> Returns the generated random key. The keyblock contents should be released when no longer needed by calling the krb5_free_keyblock_contents() routine.

Usage

The **krb5_c_make_random_key()** routine generates a random encryption key. This key can then be used to encrypt data or generate keyed checksums using the requested encryption algorithm.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_random_make_octets (generate random binary string)

Purpose

Generates a random binary string.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_random_make_octets (
    krb5_context                    context,
    krb5_data *                     data)
```

Parameters

Input

context
> Specifies the Kerberos context.

data
> The *length* field in data specifies the number of random bytes to be generated. The application is responsible for setting the length field, and for allocating the data result buffer to fit the number of bytes specified in the length field.

Output

data
> Returns the generated random data.

Usage

The **krb5_c_random_make_octets()** routine generates random bytes.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_string_to_key (generate encryption key from text string)

Purpose

Generates an encryption key from a text string. For a description of the parameters see "krb5_c_string_to_key_with_params (generate encryption key from text string with params)" on page 36 which now supercedes this.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_string_to_key (
    krb5_context                        context,
    krb5_enctype                        enctype,
    const krb5_data *                   string,
    const krb5_data *                   salt,
    krb5_keyblock *                     key)
```

krb5_c_string_to_key_with_params (generate encryption key from text string with params)

Purpose

Generates an encryption key from a text string with params.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_string_to_key_with_params (
    krb5_context                        context,
    krb5_enctype                        enctype,
    const krb5_data *                   string,
    const krb5_data *                   salt,
    const krb5_data *                   params,
    krb5_keyblock *                     key)
```

Parameters

Input

context
    Specifies the Kerberos context.

enctype
    Specifies the encryption type of the generated key. (Note that its value will be validated for FIPS compliance.) The following encryption types may be specified:

- ENCTYPE_DES_CBC_CRC - 32-bit CRC checksum with DES encryption. This encryption type should be used for interoperability with older levels of Kerberos V5. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD4 - MD4 checksum with DES encryption. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD5 - MD5 checksum with DES encryption. (not valid in FIPS mode)

- ENCTYPE_DES_HMAC_SHA1 - SHA1 checksum with DES encryption and key derivation. (not valid in FIPS mode)
- ENCTYPE_DES3_CBC_SHA1 - SHA1 checksum with DES3 encryption and key derivation.
- ENCTYPE_AES128_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES128_CTS_HMAC_SHA256_128 - SHA2 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA384_192 - SHA2 checksum with AES encryption.

string
Specifies the text string used to generate the key. This is normally a text password.

salt
Specifies the salt string used to generate the key. This is normally a string composed of the Kerberos realm and principal names. Specify NULL for this parameter if no salt is to be used when generating the key.

params
Specifies parameters that are specific for the encryption type.

Supported params by encryption type:

- DES: Params not supported. Must be NULL or have a length of zero.
- DESD: Params not supported. Must be NULL or have a length of zero.
- DES3: Params not supported. Must be NULL or have a length of zero.
- AES:

  – Iteration count:

    Must be a 4 byte unsigned integer between 1 and 50,000 inclusive. If not specified, 4096 will be used for AES SHA1 encryption types and 32768 will be used for AES SHA2 encryption types.

Output

key
Returns the generated key. The key contents should be released when no longer needed by calling the **krb5_free_keyblock_contents()** routine.

Usage

The **krb5_c_string_to_key_with_params()** routine generates an encryption key of the specified type. One use for this routine is to generate an encryption key from a user password.

The usual Kerberos password routines generate an encryption key from a password using a salt composed of the realm and the principal with component separators removed. For example, if the realm is KRB390.IBM.COM and the principal is **rwh/admin**, the salt is "KRB390.IBM.COMrwhadmin".

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_c_verify_checksum (verify checksum)

Purpose

Verifies the checksum for a data block.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_c_verify_checksum (
    krb5_context                            context,
    const krb5_keyblock *                   key,
    krb5_keyusage                           usage,
```

```
    const krb5_data *                        data,
    const krb5_checksum *                    cksum,
    krb5_boolean *                           valid)
```

Parameters

Input

context
> Specifies the Kerberos context.

key
> Specifies the key for a keyed checksum. This parameter is ignored if the specified checksum algorithm does not use an encryption key.

usage
> Specifies the key usage. This value is used to derive the actual encryption key from the supplied key and allows different message types to use different keys. This parameter is ignored if the specified checksum algorithm does not use an encryption key or does not use key derivation. Refer to RFC 4120 for usage values reserved for applications.

data
> Specifies the data to be used.

cksum
> Specifies the checksum to be verified. (Note that its value will be validated for FIPS compliance)

Output

valid
> Returns TRUE if the supplied checksum matches the checksum generated for the supplied data, otherwise returns FALSE.

Usage

The **krb5_c_verify_checksum()** routine verifies that a data block has not been modified, by computing the checksum for the supplied data, and then comparing this checksum to the checksum provided by the application.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_close (close credentials cache)

Purpose

Closes a credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_close (
    krb5_context                         context,
    krb5_ccache                          ccache)
```

Parameters

Input

context
> Specifies the Kerberos context.

ccache
    Specifies the credentials cache handle.

Usage

The **krb5_cc_close()** routine closes a credentials cache. The cache handle may not be used once this routine completes.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_default (resolve default credentials cache)

Purpose

Resolves the default credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_default (
    krb5_context                    context,
    krb5_ccache *                   ccache)
```

Parameters

Input

context
    Specifies the Kerberos context.

Output

ccache
    Returns the credentials cache handle.

Usage

The **krb5_cc_default()** routine resolves the default credentials cache and returns a handle that can be used to access the cache. This is equivalent to calling the **krb5_cc_resolve()** routine with the name returned by the **krb5_cc_default_name()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_default_name (return default credentials cache name)

Purpose

Returns the default credentials cache name.

Format

```
#include <skrb/krb5.h>
char * krb5_cc_default_name (
    krb5_context                context)
```

Parameters

Input

context
  Specifies the Kerberos context.

Usage

The **krb5_cc_default_name()** routine returns the name of the default credentials cache for the Kerberos context. The default credentials cache is determined as follows:

1. The name set by the **krb5_cc_set_default_name()** routine.

2. The value of the KRB5CCNAME environment variable.

3. The contents of the file specified by the _EUV_SEC_KRB5CCNAME_FILE environment variable (the file name defaults to **$HOME/krb5ccname** if _EUV_SEC_KRB5CCNAME_FILE is not set).

4. A new credentials cache name is generated if no default name is found.

The function return value is NULL if an error occurred. Otherwise, it is the address of the default credentials cache name. This is a pointer to read-only storage and must not be freed by the application.

The **krb5_cc_default_name()** and **krb5_cc_set_default_name()** routines use storage within the Kerberos context to hold the default credentials cache name. Thus, these routines are not thread-safe unless a separate Kerberos context is used for each thread.

krb5_cc_destroy (delete credentials cache)

Purpose

Deletes a credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_destroy (
    krb5_context                      context,
    krb5_ccache                       ccache)
```

Parameters

Input

context
  Specifies the Kerberos context.

ccache
  Specifies the credentials cache handle.

Usage

The **krb5_cc_destroy()** routine closes and deletes a credentials cache. The cache handle may not be used after this routine completes.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_end_seq_get (end reading of credential cache)

Purpose

Ends the sequential reading of the credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_end_seq_get (
    krb5_context                    context,
    krb5_ccache                     ccache,
    krb5_cc_cursor *                cursor)
```

Parameters

Input

context
　　Specifies the Kerberos context.

ccache
　　Specifies the credentials cache handle.

Input/Output

cursor
　　Specifies the cursor created by the **krb5_cc_start_seq_get()** routine.

Usage

The **krb5_cc_end_seq_get()** routine unlocks the credentials cache and releases the cursor. The cursor may not be used once **krb5_cc_end_seq_get()** has completed. The **krb5_cc_end_seq_get()** must be called on the same thread that called **krb5_cc_start_seq_get()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_generate_new (generate new credentials cache)

Purpose

Generates a new credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_generate_new (
    krb5_context                    context,
    const char *                    type,
    krb5_ccache *                   ccache)
```

Parameters

Input

context
　　Specifies the Kerberos context.

type
　　Specifies the credentials cache type (for example, FILE).

Output

ccache
> Returns the credentials cache handle. Either the **krb5_cc_close()** routine or **krb5_cc_destroy()** routine should be called to release the handle when it is no longer needed.

Usage

The **krb5_cc_generate_new()** routine creates a new credentials cache with a unique name. The **krb5_cc_initialize()** function must be called to set the cache principal before storing any credentials in the cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_get_name (return credentials cache)

Purpose

Returns the credentials cache name.

Format

```
#include <skrb/krb5.h>
char * krb5_cc_get_name (
    krb5_context                    context,
    krb5_ccache                     ccache)
```

Parameters

Input

context
> Specifies the Kerberos context.

ccache
> Specifies the credentials cache handle.

Usage

The **krb5_cc_get_name()** routine returns the name of the credentials cache. The returned name does not include the credentials cache type prefix.

The function return value is the address of the credentials cache name. This is a read-only value and must not be freed by the application.

krb5_cc_get_principal (return credentials cache principal)

Purpose

Returns the principal associated with the credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_get_principal (
    krb5_context                    context,
    krb5_ccache                     ccache,
    krb5_principal *                principal)
```

Parameters

Input

context
Specifies the Kerberos context.

ccache
Specifies the credentials cache handle.

Output

principal
Returns the principal. The **krb5_free_principal()** routine should be called to release the principal when it is no longer needed.

Usage

The **krb5_cc_get_principal()** routine returns the principal associated with the credentials cache. The principal name is set by the **krb5_cc_initialize()** routine. This is the default client principal for tickets stored in the credentials cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_get_type (return credentials cache type)

Purpose

Returns the credentials cache type.

Format

```
#include <skrb/krb5.h>
char * krb5_cc_get_type (
    krb5_contextcontext,
    krb5_ccache                         ccache)
```

Parameters

Input

context
Specifies the Kerberos context.

ccache
Specifies the credentials cache handle.

Usage

The **krb5_cc_get_type()** routine returns the credentials cache type.

The function return value is the address of the credentials cache type. This is a read-only value and must not be freed by the application.

krb5_cc_initialize (initialize credentials cache)

Purpose

Initializes a credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_initialize (
    krb5_context                    context,
    krb5_ccache                     ccache,
    krb5_principal                  principal)
```

Parameters

Input

context
    Specifies the Kerberos context.

ccache
    Specifies the credentials cache handle.

principal
    Specifies the default principal for the cache.

Usage

The **krb5_cc_initialize()** routine initializes a credentials cache. Any existing credentials are discarded and the principal name for the cache is set to the value specified. The principal name is the default client name for tickets, which are placed into the cache. Initialize a new cache before storing tickets in it.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_next_cred (return credentials cache next entry)

Purpose

Returns the next entry from the credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_next_cred (
    krb5_context                        context,
    krb5_ccache                         ccache,
    krb5_cc_cursor *                    cursor,
    krb5_creds *                        creds)
```

Parameters

Input

context
    Specifies the Kerberos context.

ccache
    Specifies the credentials cache handle.

Input/Output

cursor
    Specifies the cursor created by the **krb5_cc_start_seq_get()** routine. The cursor is updated upon successful completion of this routine.

Output

creds

Returns the contents of the cache entry. The **krb5_free_cred_contents()** routine should be called to release the credentials contents when they are no longer needed.

Usage

The **krb5_cc_next_cred()** routine reads the next entry from the credentials cache and returns it to the application. The **krb5_cc_start_seq_get()** routine must be called to begin the sequential read operation. The **krb5_cc_next_cred()** routine is then called repeatedly to read cache entries. Finally, the **krb5_cc_end_seq_get()** routine is called when no more entries are to be read. The **krb5_cc_next_cred()** routine must be called on the same thread that called the **krb5_cc_start_seq_get()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_register (define new credentials cache type)

Purpose

Defines a new credentials cache type.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_register (
    krb5_context                    context,
    krb5_cc_ops *                   ops,
    krb5_boolean                    override)
```

Parameters

Input

context

Specifies the Kerberos context.

ops

Specifies the credentials cache operations vector. This vector defines the routines that are called to perform the credentials cache operations for the new cache type.

override

Specifies whether to override an existing definition for the same type. An error is returned if the type is already registered and FALSE is specified for this parameter.

Usage

The **krb5_cc_register()** routine registers a new credentials cache type. After the new type is registered, it can be used by any thread in the current process. The type is not known outside the current process and is no longer registered when the application ends.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_remove_cred (remove credentials cache entry)

Purpose

Removes an entry from the credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_remove_cred (
    krb5_context                          context,
    krb5_ccache                           ccache,
    krb5_flags                            flags,
    krb5_creds *                          mcreds)
```

Parameters

Input

context
　　Specifies the Kerberos context.

ccache
　　Specifies the credentials cache handle.

flags
　　Specifies the search flags that are used to determine whether a particular cache entry should be removed. The following symbolic definitions are provided for the flags and should be ORed together to set the desired search flags:

- KRB5_TC_MATCH_TIMES - The **renew_till** and **endtime** values in the cache entry must be greater than the values in the match credentials. A time value is ignored if it is zero.
- KRB5_TC_MATCH_IS_SKEY - The **is_skey** flag in the cache entry must be the same as the **is_skey** flag in the match credentials.
- KRB5_TC_MATCH_FLAGS - All of the flags set in the match credentials must also be set in the cache entry.
- KRB5_TC_MATCH_TIMES_EXACT - The time fields in the cache entry must exactly match the time fields in the match credentials.
- KRB5_TC_MATCH_FLAGS_EXACT - The flags in the cache entry must exactly match the flags in the match credentials.
- KRB5_TC_MATCH_AUTHDATA - The authorization data in the cache entry must be identical to the authorization data in the match credentials.
- KRB5_TC_MATCH_SRV_NAMEONLY - Only the name portion of the server principal in the cache entry needs to match the server principal in the match credentials. The realm values may be different. If this flag is not set, the complete principal name must match.
- KRB5_TC_MATCH_2ND_TKT - The second ticket in the cache entry must exactly match the second ticket in the match credentials.
- KRB5_TC_MATCH_KTYPE - The encryption key type in the cache entry must match the encryption key type in the match credentials.

mcreds
　　Specifies the match credentials. Fields from these credentials are matched with fields in the cache entries based upon the search flags. The client and server principals must always be set in the match credentials no matter what search flags are specified.

Usage

The **krb5_cc_remove_cred()** routine removes matching entries from the credentials cache. The client principal must always match. The KRB5_TC_MATCH_SRV_NAMEONLY flag controls how much of the server principal must match.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

Note: The **krb5_cc_remove_cred()** routine is not supported for the FILE, MEMORY, or XMEM cache types and returns an error code of KRB5_OP_NOT_SUPPORTED.

krb5_cc_resolve (resolve credentials cache name)

Purpose

Resolves a credentials cache name.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_resolve (
    krb5_context                    context,
    char *                          cache_name,
    krb5_ccache *                   ccache)
```

Parameters

Input

context
    Specifies the Kerberos context.

cache_name
    Specifies the credentials cache name in the format *type:name*. The type must be a registered credentials cache type and the name must uniquely identify a particular credentials cache of the specified type.

Output

ccache
    Returns the credentials cache handle.

Usage

The **krb5_cc_resolve()** routine resolves a credentials cache name and returns a handle that can be used to access the cache. The Kerberos runtime supports three credentials cache types: FILE, MEMORY, and XMEM. Additional credentials cache types can be registered by the application by calling the **krb5_cc_register()** routine. If no type is specified, the default is FILE.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_close or **krb5_cc_destroy** should be called when cache processing is complete. Refer to the Usage section of **krb5_cc_set_flags** for more details.

krb5_cc_retrieve_cred (retrieve credentials from cache)

Purpose

Retrieves a set of credentials from the cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_retrieve_cred (
    krb5_context                     context,
    krb5_ccache                      ccache,
    krb5_flags                       flags,
    krb5_creds *                     mcreds,
    krb5_creds *                     creds)
```

Parameters

Input

context
Specifies the Kerberos context.

ccache
Specifies the credentials cache handle.

flags
Specifies the search flags that are used to determine whether or not a particular cache entry should be returned to the caller. The following symbolic definitions are provided for the flags and should be ORed together to set the desired search flags:

- KRB5_TC_MATCH_TIMES - The **renew_till** and **endtime** values in the cache entry must be greater than the values in the match credentials. A time value is ignored if it is zero.
- KRB5_TC_MATCH_IS_SKEY - The **is_skey** flag in the cache entry must be the same as the **is_skey** flag in the match credentials.
- KRB5_TC_MATCH_FLAGS - All of the flags set in the match credentials must also be set in the cache entry.
- KRB5_TC_MATCH_TIMES_EXACT - The time fields in the cache entry must exactly match the time fields in the match credentials.
- KRB5_TC_MATCH_FLAGS_EXACT - The flags in the cache entry must exactly match the flags in the match credentials
- KRB5_TC_MATCH_AUTHDATA - The authorization data in the cache entry must be identical to the authorization data in the match credentials.
- KRB5_TC_MATCH_SRV_NAMEONLY - Only the name portion of the server principal in the cache entry needs to match the server principal in the match credentials. The realm values may be different. If this flag is not set, the complete principal name must match.
- KRB5_TC_MATCH_2ND_TKT - The second ticket in the cache entry must exactly match the second ticket in the match credentials.
- KRB5_TC_MATCH_KTYPE - The encryption key type in the cache entry must match the encryption key type in the match credentials.
- KRB5_TC_SUPPORTED_KTYPES - The encryption key type in the cache entry must be one of the encryption types specified by the **default_tgs_enctypes** value in the Kerberos configuration profile. If the **default_tgs_enctypes** value contains multiple encryption types, the list is processed from left to right and the first matching credential is returned.

mcreds
Specifies the match credentials. Fields from these credentials will be matched with fields in the cache entries based upon the search flags. The client and server principals must always be set in the match credentials no matter what search flags are specified.

Output

creds
Returns the contents of the matched cache entry. The **krb5_free_cred_contents()** routine should be called to release the credentials contents when they are no longer needed.

Usage

The **krb5_cc_retrieve_cred()** routine searches the credentials cache and returns an entry that matches the credentials specified. The client principal must always match. The KRB5_TC_MATCH_SRV_NAMEONLY flag controls how much of the server principal must match.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_set_default_name (set default credentials cache name)

Purpose

Sets the default credentials cache name for the Kerberos context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_set_default_name (
    krb5_context                context,
    const char *                name)
```

Parameters

Input

context
> Specifies the Kerberos context.

name
> Specifies the credentials cache name.

Usage

The **krb5_cc_set_default_name()** routine sets the name of the default credentials cache for the Kerberos context. Specifying NULL for the name causes the normal search order to be used to determine the default credentials cache name (refer to **krb5_cc_default_name()** for a description of the search order).

The **krb5_cc_default_name()** and **krb5_cc_set_default_name()** routines are not thread-safe unless a separate Kerberos context is used for each thread.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_set_flags (set processing flags)

Purpose

Sets processing flags for the credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_set_flags (
    krb5_context                context,
    krb5_ccache                 ccache,
    krb5_flags                  flags)
```

Parameters

Input

context
> Specifies the Kerberos context.

ccache
> Specifies the credentials cache handle.

flags
> Specifies the flags. The allowable flags depend upon the cache type.

Usage

The **krb5_cc_set_flags()** routines sets the processing flags for a credentials cache. The interpretation of the flags is dependent upon the cache type.

The**krb5_cc_set_flags()** routine is not supported by the MEMORY or XMEM cache types and returns an error code of KRB5_CC_OP_NOT_SUPPORTED.

The FILE cache type supports just the KRB5_TC_OPENCLOSE flag. If this flag is specified, the credentials cache file is opened each time a credentials cache routine is called and then closed before returning to the caller (this is the default behavior if the **krb5_cc_set_flags()** routine is not called). If this flag is not specified, the credentials cache file is opened and remains open until the credentials cache is closed by the **krb5_cc_close()** or **krb5_cc_destroy()** routine. An exception is for the sequential read routines. Regardless of the KRB5_TC_OPENCLOSE flag setting, the credentials cache file is opened when the **krb5_cc_start_seq_get()** routine is called and remains open until the **krb5_cc_end_seq_get()** routine is called

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_start_seq_get (start retrieving credentials cache)

Purpose

Starts sequentially retrieving entries from the credentials cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_start_seq_get (
    krb5_context                        context,
    krb5_ccache                         ccache,
    krb5_cc_cursor *                    cursor)
```

Parameters

Input

context
Specifies the Kerberos context.
ccache
Specifies the credentials cache handle.

Output

cursor
Returns the cursor. The **krb5_cc_end_seq_get()** routine should be called to release the cursor at the completion of the sequential read operation.

Usage

The **krb5_cc_start_seq_get**() routine prepares for sequentially reading entries in the credentials cache. The **krb5_cc_next_cred()** routine is called repeatedly to retrieve each successive cache entry. The **krb5_cc_end_seq_get()** routine is called at the completion of the read operation.

The credentials cache is locked when the **krb5_cc_start_seq_get()** routine is called and remains locked until the **krb5_cc_end_seq_get()** routine is called. Write access to the cache by other processes and threads isblocked until the cache is unlocked. After the **krb5_cc_start_seq_get()** routine has been called, the current thread may not call any other credentials cache functions except **krb5_cc_next_cred()** and **krb5_cc_end_seq_get()** for the specified cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_cc_store_cred (store new credentials)

Purpose

Stores a new set of credentials in the cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_cc_store_cred (
    krb5_context                        context,
    krb5_ccache                         ccache,
    krb5_creds *                        creds)
```

Parameters

Input

context
  Specifies the Kerberos context.

ccache
  Specifies the credentials cache handle.

creds
  Specifies the Kerberos credentials.

Usage

The **krb5_cc_store_cred()** routine stores a new set of Kerberos credentials in the credentials cache. Existing credentials for the same client/server pair are not removed, even if they are expired. Credentials are stored first-in, first-out which means that newer credentials are retrieved after older credentials.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_change_password (change principal password)

Purpose

Changes the password for a principal.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_change_password (
    krb5_context                        context,
    krb5_creds *                        creds,
    char *                              newpw,
    int *                               result_code,
    krb5_data *                         result_code_string,
    krb5_data *                         result_string);
```

Parameters

Input

context
  Specifies the Kerberos context.

creds
Specifies the credentials for the request. This must be an initial ticket to the **kadmin/changepw** service for the principal whose password is to be changed.

newpw
Specifies the new password for the principal.

Output

result_code
Returns the result code for the change password request:

- 0 = password changed (KRB5_KPASSWD_SUCCESS)
- 1 = request packet incorrect (KRB5_KPASSWD_MALFORMED)
- 2 = password server error (KRB5_KPASSWD_HARDERROR)
- 3 = authentication error (KRB5_KPASSWD_AUTHERROR)
- 4 = password change rejected (KRB5_KPASSWD_SOFTERROR)

result_code_string
Returns the text description associated with the result code. Specify NULL for this parameter if the text description is not needed. The text description should be released when it is no longer needed by calling the **krb5_free_string()** function.

result_string
Returns any additional information provided by the password change server. Specify NULL for this parameter if the additional information is not needed. The result string should be released when it is no longer needed by calling the **krb5_free_string()** function.

Usage

The **krb5_change_password()** function changes the password for the principal identified by the supplied credentials. The password change server applies any applicable password policy checks before changing the password. The password change is rejected if the policy checks are not successful.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. The password is not changed unless both the function return value and the result code are zero.

krb5_copy_address (copy Kerberos address)

Purpose

Copies a Kerberos address to a new structure.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_address (
    krb5_context                    context,
    const krb5_address *            from_addr,
    krb5_address **                 to_addr)
```

Parameters

Input

context
Specifies the Kerberos context.

from_address
Specifies the address to be copied.

Output

to_address
    Returns the new **krb5_address** structure. The **krb5_free_address()** routine should be called to release the address when it is no longer needed.

Usage

The **krb5_copy_address()** routine makes a copy of a Kerberos address structure.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_addresses (copy an array of Kerberos addresses)

Purpose

Copies an array of Kerberos addresses.

Format

```
#include <skrb/krb5.h>

krb5_error_code krb5_copy_addresses (
    krb5_context                    context,
    krb5_address * const *          from_addrs,
    krb5_address ***                to_addrs)
```

Parameters

Input

context
    Specifies the Kerberos context.

from_addrs
    Specifies the array of addresses to be copied. The last array entry must be a NULL pointer.

Output

to_addrs
    Returns the new **krb5_address** array. The **krb5_free_addresses()** routine should be called to release the address array when it is no longer needed.

Usage

The **krb5_copy_addresses()** routine makes a copy of an array of Kerberos address structures.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_authdata (copy an array of authorization data structures)

Purpose

Copies an array of authorization data structures.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_authdata (
```

```
    krb5_context                    context,
    krb5_authdata * const *         from_authdata,
    krb5_authdata ***               to_authdata)
```

Parameters

Input

context
    Specifies the Kerberos context.

from_authdata
    Specifies the array of **krb5_authdata** structures. The last array entry must be a NULL pointer.

Output

to_authdata
    Returns the new array of **krb5_authdata** structures. The **krb5_free_authdata()** routine should be called to release the array when it is no longer needed.

Usage

The **krb5_copy_authdata()** routine copies an array of **krb5_authdata** structures.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_authenticator (copy a Kerberos authenticator)

Purpose

Copies a Kerberos authenticator.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_authenticator (
    krb5_context                    context,
    const krb5_authenticator *      from_authent,
     krb5_authenticator **          to_authent)
```

Parameters

Input

context
    Specifies the Kerberos context.

from_authent
    Specifies the authenticator to be copied.

Output

to_authent
    Returns the copied authenticator. The **krb5_free_authenticator()** routine should be called to release the authenticator when it is no longer needed.

Usage

The **krb5_copy_authenticator()** routine copies a Kerberos authenticator.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_checksum (copy a Kerberos checksum)

Purpose

Copies a Kerberos checksum.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_checksum (
    krb5_context                context,
    const krb5_checksum *       from_cksum,
    krb5_checksum **            to_cksum)
```

Parameters

Input

context
> Specifies the Kerberos context.

from_cksum
> Specifies the checksum to be copied.

Output

to_cksum
> Returns the copied checksum. The **krb5_free_checksum()** routine should be called to release the checksum when it is no longer needed.

Usage

The **krb5_copy_checksum()** copies a Kerberos checksum.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_creds (copy Kerberos credentials)

Purpose

Copies Kerberos credentials.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_creds (
    krb5_context                context,
    const krb5_creds *          from_creds,
    krb5_creds **              to_creds)
```

Parameters

Input

context
> Specifies the Kerberos context.

from_creds
> Specifies the credentials to be copied.

Output

to_creds
> Returns the copied credentials. The **krb5_free_creds()** routine should be called to release the credentials are no longer needed.

Usage

The **krb5_copy_creds()** routine copies Kerberos credentials.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_data (copy Kerberos data object)

Purpose

Copies a Kerberos data object.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_data (
    krb5_context                    context,
    const krb5_data *               from_data,
    krb5_data **                    to_data)
```

Parameters

Input

context
> Specifies the Kerberos context.

from_data
> Specifies the data object to be copied.

Output

to_data
> Returns the copied data object. The **krb5_free_data()** routine should be called to release the data object when it is no longer needed.

Usage

The **krb5_copy_data()** routine copies a Kerberos data object that is represented by a **krb5_data** structure.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_keyblock (copy Kerberos keyblock)

Purpose

Copies a Kerberos keyblock.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_keyblock (
    krb5_context                    context,
```

```
    const krb5_keyblock *          from_keyblock,
    krb5_keyblock **                to_keyblock)
```

Parameters

Input

context
  Specifies the Kerberos context.

from_keyblock
  Specifies the keyblock to be copied.

Output

to_keyblock
  Returns the copied keyblock. The **krb5_free_keyblock()** routine should be called to release the keyblock when it is no longer needed.

Usage

The **krb5_copy_keyblock()** routine copies a Kerberos keyblock.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_keyblock_contents (copy Kerberos keyblock contents)

Purpose

Copies the contents of a Kerberos keyblock.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_keyblock_contents (
    krb5_context                   context,
    const krb5_keyblock *          from_keyblock,
    krb5_keyblock *                to_keyblock)
```

Parameters

Input

context
  Specifies the Kerberos context.

from_keyblock
  Specifies the keyblock to be copied.

Output

to_keyblock
  Returns the contents of the input keyblock. The **krb5_free_keyblock_contents()** routine should be called to release the contents of the keyblock when it is no longer needed.

Usage

The **krb5_copy_keyblock_contents()** routine copies the contents of a Kerberos keyblock into an existing keyblock. The current contents of the output keyblock are not released before performing the copy.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_principal (copy Kerberos principal)

Purpose

Copies a Kerberos principal.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_principal (
    krb5_context                    context,
    krb5_const_principal            from_princ,
    krb5_principal *                to_princ)
```

Parameters

Input

context
Specifies the Kerberos context.

from_princ
Specifies the principal to be copied.

Output

to_princ
Returns the copied principal. The **krb5_free_principal()** routine should be called to release the principal when it is no longer needed.

Usage

The **krb5_copy_principal()** routine copies a Kerberos principal.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_copy_ticket (copy Kerberos ticket)

Purpose

Copies a Kerberos ticket.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_copy_ticket (
    krb5_context                    context,
    const krb5_ticket *             from_ticket,
    krb5_ticket **                  to_ticket)
```

Parameters

Input

context
Specifies the Kerberos context.

from_ticket
Specifies the ticket to be copied.

Output

to_ticket
Returns the copied ticket. The **krb5_free_ticket()** routine should be called to release the ticket when it is no longer needed.

Usage

The **krb5_copy_ticket()** routine copies a Kerberos ticket.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_dll_load (load Kerberos runtime library)

Purpose

Loads the Kerberos runtime library.

Format

```
#include <skrb/krbload.h>
int krb5_dll_load (
    krb5_ui_4 *                             function_mask,
    krb5_api_vector **                      function_vector)
```

Parameters

Output

function_mask
Returns a bit mask indicating the functions available in the version of the Kerberos runtime.

function_vector
Returns the address of the DLL address vector.

Usage

The krb5_dll_load() routine dynamically loads the Kerberos runtime. This is an alternative to automatically loading the Kerberos runtime during process initialization. In order to dynamically load the Kerberos DLL, the application must not make direct calls to any function contained in the DLL nor make any direct references to variables defined in the DLL. Instead, functions and variables must be accessed using the addresses in the vector returned by the krb5_dll_load() routine.

The application can unload the DLL when it is no longer needed by calling the krb5_dll_unload() routine. The DLL is automatically unloaded at process termination.

Multiple calls to krb5_dll_load() without an intervening call to krb5_dll_unload() cause the dynamic load count to be incremented. The Kerberos runtime is not unloaded until the the load count is reduced to zero by calling the krb5_dll_unload() routine once for each call to the krb5_dll_load() routine.

The function mask indicates the capabilities of the version of the Kerberos DLL currently loaded. The following values have been defined:

- KRB5_API_LVL1 - Kerberos functions provided as part of z/OS Version 1 Release 2 are available
- KRB5_API_LVL2 - Kerberos functions provided as part of z/OS Version 1 Release 4 are available
- KRB5_API_LVL3 - Kerberos functions provided as part of z/OS Version 1 Release 6 are available
- KRB5_API_LVL4 - Kerberos functions provided as part of z/OS Version 1 Release 9 are available
- KRB5_API_LVL5 - Kerberos functions provided as part of z/OS Version 1 Release 12 are available
- KRB5_API_LVL6 - Kerberos functions provided as part of z/OS Version 2 Release 2 are available.

- KRB5_API_LVL7 - Kerberos functions provided as part of z/OS Version 2 Release 3 are available.
- KRB5_API_LVL8 - Kerberos functions provided as part of z/OS Version 2 Release 4 are available.

The function return code is 0 if no error occurred or the errno value for the failing system function if an error occurred.

krb5_dll_unload (unload Kerberos runtime library)

Purpose

Unloads the Kerberos runtime library.

Format

```
#include <skrb/krbload.h>
int krb5_dll_unload ( void )
```

Parameters

None

Usage

Each call to **krb5_dll_load()** increments the dynamic load count, and each call to **krb5_dll_unload()** decrements the dynamic load count. The Kerberos runtime is terminated and the Kerberos DLL is unloaded when the dynamic load count reaches 0. The DLL is not unloaded if it was loaded automatically during process initialization, but the Kerberos runtime is still terminated when the dynamic load count reaches 0.

Results are unpredictable if the Kerberos runtime is in use by another thread at the time the **krb5_dll_unload()** routine is called. The application is responsible for closing or destroying open credentials caches, replay caches, and key tables before unloading the Kerberos runtime.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_free_address (release Kerberos address storage)

Purpose

Releases the storage assigned to a Kerberos address.

Format

```
#include <skrb/krb5.h>
void krb5_free_address (
    krb5_context                        context,
    krb5_address *                      addr)
```

Parameters

Input

context
    Specifies the Kerberos context.

addr
    Specifies the **krb5_address** to be released.

Usage

The **krb5_free_address()** routine releases the storage assigned to the contents of a **krb5_address** structure and then it releases the **krb5_address** structure itself.

krb5_free_addresses (release Kerberos address storage)

Purpose

Releases the storage assigned to an array of Kerberos addresses.

Format

```
#include <skrb/krb5.h>
void krb5_free_addresses (
    krb5_context                        context,
    krb5_address **                     addrs)
```

Parameters

Input

context
    Specifies the Kerberos context.

addrs
    Specifies the array to be released. The last entry in the array must be a NULL pointer.

Usage

The **krb5_free_addresses()** routine releases the storage assigned to an array of **krb5_address** structures. Each **krb5_address** structure is released and then the pointer array itself is released.

krb5_free_ap_rep_enc_part (release decrypted storage)

Purpose

Releases the storage assigned to the decrypted portion of an AP_REP message.

Format

```
#include <skrb/krb5.h>
void krb5_free_ap_rep_enc_part (
    krb5_context                        context,
    krb5_ap_rep_enc_part *              enc_part)
```

Parameters

Input

context
    Specifies the Kerberos context.

enc_part
    Specifies the reply to be released.

Usage

The **krb5_free_ap_rep_enc_part()** routine releases the storage assigned to the decrypted reply returned by the **krb5_rd_rep()** routine.

krb5_free_authdata (release authentication data storage)

Purpose

Releases the storage assigned to an array of authentication data.

Format

```
#include <skrb/krb5.h>
void krb5_free_authdata (
    krb5_context                         context,
    krb5_authdata **                     authdata)
```

Parameters

Input

context
    Specifies the Kerberos context.

authdata
    Specifies the array to be released. The last entry in the array must be a NULL pointer.

Usage

The **krb5_free_authdata()** routine releases the storage assigned to an array of **krb5_authdata** structures. Each **krb5_authdata** structure is released and then the pointer array itself is released.

krb5_free_authenticator (release authenticator storage)

Purpose

Releases the storage assigned to an authenticator.

Format

```
#include <skrb/krb5.h>
void krb5_free_authenticator (
    krb5_context                         context,
    krb5_authenticator *                 authent)
```

Parameters

Input

context
    Specifies the Kerberos context.

authent
    Specifies the **krb5_authenticator** to be released.

Usage

The **krb5_free_authenticator()** routine releases the storage assigned to the contents of a **krb5_authenticator** structure and then it releases the **krb5_authenticator** structure itself.

krb5_free_authenticator_contents (release authenticator storage)

Purpose

Releases the storage assigned to the contents of an authenticator.

Format

```
#include <skrb/krb5.h>
void krb5_free_authenticator_contents (
    krb5_context                        context,
    krb5_authenticator *                authent)
```

Parameters

Input

context
    Specifies the Kerberos context.

authent
    Specifies the **krb5_authenticator** to be released.

Usage

The **krb5_free_authenticator_contents()** routine releases the storage assigned to the contents of a **krb5_authenticator** structure. Unlike the **krb5_free_authenticator()** routine, the **krb5_free_authenticator_contents()** routine does not free the **krb5_authenticator** structure.

krb5_free_checksum (release checksum storage)

Purpose

Releases the storage assigned to a checksum.

Format

```
#include <skrb/krb5.h>
void krb5_free_checksum (
    krb5_context                        context,
    krb5_checksum *                     cksum)
```

Parameters

Input

context
    Specifies the Kerberos context.

cksum
    Specifies the **krb5_checksum** to be released.

Usage

The **krb5_free_checksum()** routine releases the storage assigned to a **krb5_checksum** structure and then releases the **krb5_checksum** structure itself.

krb5_free_checksum_contents (release checksum storage)

Purpose

Releases the storage assigned to the contents of a checksum.

Format

```
#include <skrb/krb5.h>
void krb5_free_checksum_contents (
    krb5_context          context,
    krb5_checksum *       cksum)
```

Parameters

Input

context
Specifies the Kerberos context.

cksum
Specifies the **krb5_checksum** to be released.

Usage

The **krb5_free_checksum_contents()** routine releases the storage assigned to the contents of a **krb5_checksum** structure. Unlike the **krb5_free_checksum()** routine, the **krb5_checksum** structure itself is not released.

krb5_free_cksumtypes (release checksum storage)

Purpose

Release the storage assigned to an array of checksum types.

Format

```
#include <skrb/krb5.h>
void krb5_free_cksumtypes (
    krb5_context                   context,
    krb5_cksumtype *               cksumtypes)
```

Parameters

Input

context
Specifies the Kerberos context.

cksumtypes
Specifies the array of checksum types to be released.

Usage

The **krb5_free_cksumtypes()** routine releases storage that was created by **krb5_c_keyed_checksum_types**.

krb5_free_context (release Kerberos context)

Purpose

Releases a Kerberos context.

Format

```
#include <skrb/krb5.h>
void krb5_free_context (
    krb5_context                  context)
```

Parameters

Input

context
    Specifies the Kerberos context.

Usage

The **krb5_free_context()** routine is used to release a context that was created by the **krb5_init_context()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_free_cred_contents (release credential storage)

Purpose

Releases the storage assigned to contents of a credential.

Format

```
#include <skrb/krb5.h>
void krb5_free_cred_contents (
    krb5_context                  context,
    krb5_creds *                  creds)
```

Parameters

Input

context
    Specifies the Kerberos context.

creds
    Specifies the credentials.

Usage

The **krb5_free_cred_contents()** routine releases the storage assigned to the contents of a **krb5_creds** structure. Unlike the **krb5_free_creds()** routine, the **krb5_free_cred_contents()** routine does not release the **krb5_creds** structure.

krb5_free_creds (release credential storage)

Purpose

Releases the storage assigned to a credential.

Format

```
#include <skrb/krb5.h>
void krb5_free_creds (
    krb5_context                         context,
    krb5_creds *                         creds)
```

Parameters

Input

context
    Specifies the Kerberos context.

creds
    Specifies the credentials.

Usage

The **krb5_free_creds()** routine releases the storage assigned to the contents of a **krb5_creds** structure and then releases the **krb5_creds** structure itself.

krb5_free_data (release Kerberos data object storage)

Purpose

Releases the storage assigned to a Kerberos data object.

Format

```
#include <skrb/krb5.h>
void krb5_free_data (
    krb5_context                         context,
    krb5_data *                          data)
```

Parameters

Input

context
    Specifies the Kerberos context.

data
    Specifies the data object.

Usage

The **krb5_free_data()** routine releases the storage assigned to a Kerberos data object represented by a **krb5_data** structure.

krb5_free_data_contents (release Kerberos data object storage)

Purpose

Release the storage assigned to the contents of a Kerberos data object.

Format

```
#include <skrb/krb5.h>
void krb5_free_data_contents (
    krb5_context                        context,
    krb5_data *                         data)
```

Parameters

Input

context
    Specifies the Kerberos context.

data
    Specifies the data object.

Usage

The **krb5_free_data_contents()** routine releases the storage assigned to the contents of a Kerberos data object represented by a **krb5_data** structure. Unlike the **krb5_free_data()** routine, the **krb5_free_data_contents()** routine does not release the **krb5_data** structure.

krb5_free_enc_tkt_part (release encrypted ticket storage)

Purpose

Releases the storage assigned to an encrypted ticket part.

Format

```
#include <skrb/krb5.h>
void krb5_free_enc_tkt_part (
    krb5_context                  context,
    krb5_enc_tkt_part *           enc_tkt)
```

Parameters

Input

context
    Specifies the Kerberos context.

enc_tkt
    Specifies the **krb5_enc_tkt_part** structure to be released.

Usage

The **krb5_free_enc_tkt_part()** routine releases the storage assigned to the **krb5_enc_tkt_part** structure and then releases the **krb5_enc_tkt_part** structure itself. The **krb5_enc_tkt_part** structure is created when a ticket is decrypted and decoded.

krb5_free_enctypes (release encryption storage)

Purpose

Releases the storage assigned to an array of encryption types.

Format

```
#include <skrb/krb5.h>
void krb5_free_enctypes (
    krb5_context                    context,
    krb5_enctype *                  enctypes)
```

Parameters

Input

context
Specifies the Kerberos context.

enctypes
Specifies the array of encryption types to be released.

Usage

The **krb5_free_enctypes()** routine releases storage assigned to an array of encryption types.

krb5_free_error (release Kerberos error message storage)

Purpose

Releases the storage assigned to a Kerberos error message.

Format

```
#include <skrb/krb5.h>
void krb5_free_error (
    krb5_context                    context,
    krb5_error *                    error)
```

Parameters

Input

context
Specifies the Kerberos context.

error
Specifies the **krb5_error** structure to be released.

Usage

The **krb5_free_error()** routine releases the storage assigned to the **krb5_error** structure and then releases the **krb5_error** structure itself. The **krb5_error** structure is created when a Kerberos error message is processed by the **krb5_rd_error()** routine.

krb5_free_host_realm (release realm list storage)

Purpose

Releases the storage assigned to a realm list.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_free_host_realm (
    krb5_context                    context,
    char * const *                  realm_list)
```

Parameters

Input

context
    Specifies the Kerberos context.

realm_list
    Specifies the realm list to be released.

Usage

The **krb5_free_host_realm()** routine releases the storage assigned to a realm list.

The function return value is always zero.

krb5_free_kdc_rep (release KDC reply storage)

Purpose

Releases the storage assigned to a KDC reply.

Format

```
#include <skrb/krb5.h>
void krb5_free_kdc_rep (
    krb5_context                        context,
    krb5_kdc_rep *                      reply)
```

Parameters

Input

context
    Specifies the Kerberos context.

reply
    Specifies the KDC reply to be released.

Usage

The **krb5_free_kdc_rep()** routine releases the contents of the **krb5_kdc_rep** structure and then it releases the **krb5_kdc_rep** structure itself.

krb5_free_keyblock (release keyblock storage)

Purpose

Releases the storage assigned to a keyblock.

Format

```
#include <skrb/krb5.h>
void krb5_free_keyblock (
    krb5_context                  context,
    krb5_keyblock *               keyblock)
```

Parameters

Input

context
    Specifies the Kerberos context.

keyblock
    Specifies the keyblock to be released.

Usage

The **krb5_free_keyblock()** routine releases the contents of the **krb5_keyblock** structure and then it releases the **krb5_keyblock** structure itself.

krb5_free_keyblock_contents (release keyblock storage)

Purpose

Releases the storage assigned to the contents of a keyblock.

Format

```
#include <skrb/krb5.h>
void krb5_free_keyblock_contents (
    krb5_context                  context,
    krb5_keyblock *               keyblock)
```

Parameters

Input

context
    Specifies the Kerberos context.

keyblock
    Specifies the keyblock to be released.

Usage

The **krb5_free_keyblock_contents()** routine releases the contents of the **krb5_keyblock** structure. Unlike the **krb5_free_keyblock()** routine, the **krb5_free_keyblock_contents()** routine does not release the **krb5_keyblock** structure.

krb5_free_krbhst (release host list storage)

Purpose

Releases the storage assigned to a host list.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_free_krbhst (
    krb5_context                    context,
    char * const *                  host_list)
```

Parameters

Input

context
Specifies the Kerberos context.

host_list
Specifies the host list to be released.

Usage

The **krb5_free_krbhst()** routine releases the storage assigned to a host list.

The function return value is always zero.

krb5_free_principal (release principal storage)

Purpose

Releases the storage assigned to a principal.

Format

```
#include <skrb/krb5.h>
void krb5_free_principal (
    krb5_context           context,
    krb5_principal         principal)
```

Parameters

Input

context
Specifies the Kerberos context.

principal
Specifies the **krb5_principal** to be released.

Usage

The **krb5_free_principal()** routine releases storage assigned to a **krb5_principal**.

krb5_free_string (release character string storage)

Purpose

Releases the storage assigned to a character string.

Format

```
#include <skrb/krb5.h>
void krb5_free_string (
    krb5_context        context,
    char *              string)
```

Parameters

Input

context
    Specifies the Kerberos context.

string
    Specifies the character string to be released.

Usage

The **krb5_free_string()** routine releases storage assigned to a character string.

krb5_free_tgt_creds (release credential storage)

Purpose

Releases the storage assigned to an array of credentials.

Format

```
#include <skrb/krb5.h>
void krb5_free_tgt_creds (
    krb5_context        context,
    krb5_creds **       creds)
```

Parameters

Input

context
    Specifies the Kerberos context.

creds
    Specifies the credentials array to be released. The last entry in the array must be a NULL pointer.

Usage

The **krb5_free_tgt_creds()** routine releases the storage assigned to an array of **krb5_creds** structures. Each **krb5_creds** structure is released and then the pointer array itself is released.

krb5_free_ticket (release ticket storage)

Purpose

Releases the storage assigned to a ticket.

Format

```
#include <skrb/krb5.h>
void krb5_free_ticket (
    krb5_context          context,
    krb5_ticket *         ticket)
```

Parameters

Input

context
Specifies the Kerberos context.

ticket
Specifies the **krb5_ticket** to be released.

Usage

The **krb5_free_ticket()** routine releases the storage assigned to a **krb5_ticket** structure and then releases the **krb5_ticket** structure itself.

krb5_free_tickets (release ticket storage)

Purpose

Releases the storage assigned to an array of tickets.

Format

```
#include <skrb/krb5.h>
void krb5_free_tickets (
    krb5_context                    context,
    krb5_ticket **                  tickets)
```

Parameters

Input

context
Specifies the Kerberos context.

tickets
Specifies the array to be released. The last entry in the array must be a NULL pointer.

Usage

The **krb5_free_tickets()** routine releases the storage assigned to an array of **krb5_ticket** structures. Each **krb5_ticket** structure is released and then the pointer array itself is released.

krb5_gen_replay_name (generate replay cache name)

Purpose

Generates a replay cache name.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_gen_replay_name (
    krb5_context                      context,
    const krb5_address *              inaddr,
    const char *                      unique,
    char **                           string)
```

Parameters

Input

context
    Specifies the Kerberos context.

inaddr
    Specifies the address to be incorporated into the cache name.

unique
    Specifies the unique portion of the replay cache name.

Output

string
    Returns the generated replay cache name. This string should be freed by the application when it is no longer needed.

Usage

The **krb5_gen_replay_name()** routine generates a unique replay cache name based on the Kerberos address supplied by the caller. The **unique** parameter is used to differentiate this replay cache from others currently in use on the system. The generated cache name consists of the unique portion concatenated with the hexadecimal representation of the Kerberos address.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

When the **use_dvipa_override** configuration option is set to 1, the selected replay cache will be overridden with a shared replay cache controlled by the SKRBKDC started task which does not use a name. This function will still work but the generation of a unique name will be meaningless and ignored.

krb5_generate_seq_number (generate random sequence number)

Purpose

Generates a random sequence number.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_generate_seq_number (
    krb5_context                      context,
    const krb5_keyblock *             key,
    krb5_int32 *                      seqno)
```

Parameters

Input

context
Specifies the Kerberos context.

key
Specifies the key used to generate the random sequence number.

Output

seqno
Returns the random sequence number.

Usage

The **krb5_generate_seq_number()** generates a random sequence number based upon the supplied key.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_generate_subkey (generate subsession key)

Purpose

Generates a subsession key.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_generate_subkey (
    krb5_context                    context,
    const krb5_keyblock *           key,
    krb5_keyblock **                subkey)
```

Parameters

Input

context
Specifies the Kerberos context.

key
Specifies the session key.

Output

subkey
Returns the generated subsession key. The **krb5_free_keyblock()** routine should be called to release the key when it is no longer needed.

Usage

The **krb5_generate_subkey()** generates a random subsession key that is based on the supplied session key.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_cred_from_kdc (obtain KDC server service ticket)

Purpose

Obtains a service ticket from the Kerberos KDC server.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_cred_from_kdc (
    krb5_context                        context,
    krb5_ccache                         ccache,
    krb5_creds *                        in_cred,
    krb5_creds **                       out_cred,
    krb5_creds ***                      tgts)
```

Parameters

Input

context
> Specifies the Kerberos context.

ccache
> Specifies the credentials cache. The initial TGT for the local realm must already be in the cache. The Kerberos runtime obtains additional ticket-granting tickets as needed if the target server is not in the local realm.

in_cred
> Specifies the request credentials. The client and server fields must be set to the desired values for the service ticket. The **second_ticket** field must be set if the service ticket is to be encrypted in a session key. The ticket expiration time can be set to override the default expiration time.

Output

out_cred
> Returns the service ticket. The **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

tgts
> Returns any new ticket-granting tickets that were obtained while getting the service target from the KDC in the target realm. There may be ticket-granting tickets returned for this parameter even if the Kerberos runtime was ultimately unable to obtain a service ticket from the target KDC. The **krb5_free_tgt_creds()** routine should be called to release the TGT array when it is no longer needed.

Usage

The **krb5_get_cred_from_kdc()** routine obtains a service ticket from the Kerberos KDC server. The credentials are not stored in the credentials cache (the application should store them in the cache if appropriate). The application should not call **krb5_get_cred_from_kdc()** if the requested service ticket is already in the credentials cache.

The **krb5_get_cred_from_kdc()** routine obtains any necessary ticket-granting tickets for intermediate realms between the client realm and the server realm. It then calls the **krb5_get_cred_via_tkt()** routine to obtain the actual service ticket. The KDC options are the same as the TGT ticket options. The KDC_OPT_ENC_TKT_IN_SKEY flag is set if the **in_cred** parameter provided a second ticket.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

See "krb5_get_credentials (obtain service ticket)" on page 79 for more details.

krb5_get_cred_from_kdc_renew (renew KDC server service ticket)

Purpose

Renews a service ticket obtained from the Kerberos KDC server.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_cred_from_kdc_renew (
    krb5_context                        context,
    krb5_ccache                         ccache,
    krb5_creds *                        in_cred,
    krb5_creds **                       out_cred,
    krb5_creds ***                      tgts)
```

Parameters

Input

context
Specifies the Kerberos context.

ccache
Specifies the credentials cache. The initial TGT for the local realm must already be in the cache. The Kerberos runtime obtains additional ticket-granting tickets as needed if the target server is not in the local realm.

in_cred
Specifies the request credentials. The client and server fields must be set to the desired values for the service ticket. The **second_ticket** field must be set if the service ticket is to be encrypted in a session key. The ticket expiration time can be set to override the default expiration time.

Output

out_cred
Returns the renewed service ticket. The **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

tgts
Returns any new ticket-granting tickets that were obtained while getting the service target from the KDC in the target realm. There may be ticket-granting tickets returned for this parameter even if the Kerberos runtime was ultimately unable to obtain a service ticket from the target KDC. The **krb5_free_tgt_creds()** routine should be called to release the TGT array when it is no longer needed.

Usage

The **krb5_get_cred_from_kdc_renew()** routine renews a service ticket obtained from the Kerberos KDC server. The credentials are not stored in the credentials cache (the application should store them in the cache if appropriate). The application should call **krb5_get_cred_from_kdc_renew()** to renew a renewable ticket before the ticket end time is reached. Note that a renewable ticket may not be renewed after its end time even if its **renew_till** time has not been reached yet.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_cred_from_kdc_validate (validate KDC server service ticket)

Purpose

Validates a service ticket obtained from the Kerberos KDC server.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_cred_from_kdc_validate (
    krb5_context                         context,
    krb5_ccache                          ccache,
    krb5_creds *                         in_cred,
    krb5_creds **                        out_cred,
    krb5_creds ***                       tgts)
```

Parameters

Input

context
  Specifies the Kerberos context.

ccache
  Specifies the credentials cache. The initial TGT for the local realm must already be in the cache. The Kerberos runtime obtains additional ticket-granting tickets as needed if the target server is not in the local realm.

in_cred
  Specifies the request credentials. The client and server fields must be set to the desired values for the service ticket. The **second_ticket** field must be set if the service ticket is to be encrypted in a session key. The ticket expiration time can be set to override the default expiration time.

Output

out_cred
  Returns the validated service ticket. The **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

tgts
  Returns any new ticket-granting tickets that were obtained while getting the service target from the KDC in the target realm. There may be ticket-granting tickets returned for this parameter even if the Kerberos runtime was ultimately unable to obtain a service ticket from the target KDC. The **krb5_free_tgt_creds()** routine should be called to release the TGT array when it is no longer needed.

Usage

The **krb5_get_cred_from_kdc_validate()** routine validates a service ticket obtained from the Kerberos KDC server. The credentials are not stored in the credentials cache (the application should store them in the cache if appropriate). The application should call **krb5_get_cred_from_kdc_validate()** to validate a postdated ticket after the ticket start time has been reached.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_cred_via_tkt (obtain service ticket)

Purpose

Obtains a service ticket from the Kerberos KDC server.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_cred_via_tkt (
    krb5_context                         context,
    krb5_creds *                         tkt,
    const krb5_flags                     kdc_options,
    krb5_address * const *               address
```

```
        krb5_creds *                        in_cred,
        krb5_creds **                       out_cred)
```

Parameters

Input

context
> Specifies the Kerberos context.

tkt
> Specifies the ticket-granting ticket for the realm containing the target server for the service ticket. The client in the TGT must be the same as the client in the request credentials.

kdc_options
> Specifies KDC options for the service ticket as follows:
>
> - KDC_OPT_FORWARDABLE - Obtain a forwardable ticket.
> - KDC_OPT_PROXIABLE - Obtain a proxiable ticket.
> - KDC_OPT_ALLOW_POSTDATE - Allow postdated tickets.
> - KDC_OPT_RENEWABLE - Obtain a renewable ticket. The **renew_till** time must be set in the request.
> - KDC_OPT_RENEWABLE_OK - A renewable ticket is acceptable if the KDC policy does not allow a ticket to be generated with the requested endtime.
> - KDC_OPT_ENC_TKT_IN_SKEY - Encrypt the service ticket in the session key of the second ticket.
>
> Unrecognized options will no longer be diagnosed by the KDC; applications must ensure their options have been honored by the KDC by checking the returned tickets.

address
> Specifies the addresses to be placed in the ticket. The ticket addresses determine which host systems can generate requests that use the ticket. A mapped IPv6 address is stored in the ticket as the corresponding IPv4 address.

in_cred
> Specifies the request credentials. The client and server fields must be set to the desired values for the service ticket. The encryption type specified must be supported, and if running in FIPS mode, must be a FIPS compliant encryption type. The **second_ticket** field must be set if the service ticket is to be encrypted in a session key. The ticket expiration time can be set to override the default expiration time.

Output

out_cred
> Returns the service ticket. The **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

Usage

The **krb5_get_cred_via_tkt()** routine uses the supplied ticket-granting ticket to obtain a service ticket to the requested server for the requested client.

If the request is for a ticket-granting ticket (TGT) in a foreign realm, the KDC may return a TGT for an intermediate realm if it is unable to return a TGT for the requested realm. The application should check the server name in the returned TGT. If the TGT is not for the desired realm, the application should call **krb5_get_cred_via_tkt()** again to send the request to the KDC for the realm in the returned TGT and should provide the TGT as the credentials for the request.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_credentials (obtain service ticket)

Purpose

Obtains a service ticket.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_credentials (
    krb5_context                      context,
    const krb5_flags                  options,
    krb5_ccache                       ccache,
    krb5_creds *                      in_cred,
    krb5_creds **                     out_cred)
```

Parameters

Input

context
Specifies the Kerberos context.

options
Specifies the option flags as follows:

- KRB5_GC_USER_USER - Obtain a user-to-user ticket.
- KRB5_GC_CACHED - Do not obtain a service ticket if one is not found in the credentials cache.

ccache
Specifies the credentials cache to be used. The initial TGT must already be in the cache.

in_cred
Specifies the request credentials. The client and server fields must be set to the desired values for the service ticket. The **second_ticket** field must be set if the service ticket is to be encrypted in a session key. The ticket expiration time can be set to override the default expiration time. The key encryption type can be set to override the default ticket encryption type.

Output

out_cred
Returns the service ticket. The **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

Usage

The **krb5_get_credentials()** routine obtains a service ticket for the requested server. This routine is the normal way for an application to obtain a service ticket. If the service ticket is already in the credentials cache, the **krb5_get_credentials()** routine returns the cached ticket. Otherwise, the **krb5_get_credentials()** routine calls the **krb5_get_cred_from_kdc()** routine to obtain a service ticket from the KDC.

The **krb5_get_credentials()** routine stores any tickets obtained during its processing in the credentials cache. This includes the requested service ticket as well as any ticket-granting tickets required to obtain the service ticket.

If KRB5_GC_CACHED is specified, the **krb5_get_credentials()** routine searches only the credentials cache for a service ticket.

If KRB5_GC_USER_USER is specified, the **krb5_get_credentials()** routine gets credentials for user-to-user authentication. In user-to-user authentication, the secret key for the server is the session key from the server's ticket-granting ticket (TGT). The TGT is passed from the server to the client over the network (this is safe since the TGT is encrypted in a key known only by the Kerberos server). The client must then pass this TGT to **krb5_get_credentials()** as the second ticket in the request credentials. The Kerberos

server uses this TGT to construct a user-to-user ticket that can be verified by the server using the session key from its TGT.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_credentials_renew (renew a ticket)

Purpose

Renews a ticket.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_credentials_renew (
    krb5_context                    context,
    const krb5_flags                options,
    krb5_ccache                     ccache,
    krb5_creds *                    in_cred,
    krb5_creds **                   out_cred)
```

Parameters

Input

context
    Specifies the Kerberos context.

options
    Specifies the option flags as follows:

- KRB5_GC_USER_USER - Obtain a user-to-user ticket.

ccache
    Specifies the credentials cache to be used.

in_cred
    Specifies the request credentials. The client and server fields must be set to the desired values for the service ticket. The **second_ticket** field must be set if the service ticket is to be encrypted in a session key. The ticket expiration time can be set to override the default expiration time.

Output

out_cred
    Returns the service ticket. The **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

Usage

The **krb5_get_credentials_renew()** routine renews a service ticket for the requested service. Upon successful completion, the credentials cache is re-initialized and the service ticket is stored in the cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_credentials_validate (validate a ticket)

Purpose

Validates a ticket.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_credentials_validate (
    krb5_context                    context,
    const krb5_flags                options,
    krb5_ccache                     ccache,
    krb5_creds *                    in_cred,
    krb5_creds **                   out_cred)
```

Parameters

Input

context
Specifies the Kerberos context.

options
Specifies the option flags as follows:

- KRB5_GC_USER_USER - Obtain a user-to-user ticket.

ccache
Specifies the credentials cache to be used.

in_cred
Specifies the request credentials. The client and server fields must be set to the desired values for the service ticket. The **second_ticket** field must be set if the service ticket is to be encrypted in a session key. The ticket expiration time can be set to override the default expiration time.

Output

out_cred
Returns the service ticket. The **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

Usage

The **krb5_get_credentials_validate()** routine validates a service ticket for the requested service. Upon successful completion, the credentials cache is re-initialized and the service ticket is stored in the cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_default_in_tkt_ktypes (return default encryption type)

Purpose

Returns the default encryption types that are used when requesting an initial ticket from the KDC.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_default_in_tkt_ktypes (
    krb5_context                    context,
    krb5_enctype **                 ktypes)
```

Parameters

Input

context
Specifies the Kerberos context.

Output

ktypes
> Returns an array of encryption types. The last entry in the array is ENCTYPE_NULL. The caller is responsible for freeing the array returned for this parameter, when it is no longer needed, by calling the **krb5_free_enctypes()** routine.

Usage

The **krb5_get_default_in_tkt_ktypes()** routine returns the encryption types that are used when requesting the initial ticket from the KDC. The encryption types are obtained from the list of encryption types defined for **default_tkt_enctypes** in the Kerberos configuration file, or if **default_tkt_enctypes** is not specified, the default encryption types, which are: aes256-cts-hmac-sha1-96, aes128-cts-hmac-sha1-96, des3-cbc-sha1. When running with a FIPS level greater than zero, all encryption types that are not FIPS compliant are removed, and if all of the encryption types specified in **default_tkt_enctypes** are not FIPS compliant, then the default encryption types will be used.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_default_realm (return default realm)

Purpose

Returns the default realm for the local system.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_default_realm (
    krb5_context                          context,
    char **                               realm)
```

Parameters

Input

context
> Specifies the Kerberos context.

Output

realm
> Returns the realm name. The application should free the name when it is no longer needed by calling the **krb5_free_string()** routine.

Usage

The **krb5_get_default_realm()** routine returns the default realm for the local system. The default realm is set by the **krb5_set_default_realm()** routine. If the default realm has not been set, it is obtained from the *default_realm* entry in the [libdefaults] section of the Kerberos configuration file.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_default_tgs_ktypes (return KDC default encryption types)

Purpose

Returns the default encryption types that are used when requesting a service ticket from the KDC.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_default_tgs_ktypes (
    krb5_context                        context,
    krb5_enctype **                     ktypes)
```

Parameters

Input

context
> Specifies the Kerberos context.

Output

ktypes
> Returns an array of encryption types. The last entry in the array is ENCTYPE_NULL. The caller is responsible for freeing the array returned for this parameter, when it is no longer needed, by calling the **krb5_free_enctypes()** routine.

Usage

The **krb5_get_default_tgs_ktypes()** routine returns the encryption types that are used when requesting a service ticket from the KDC. The encryption types are obtained from the list of encryption types defined for **default_tgs_enctypes** in the Kerberos configuration file, or if **default_tgs_enctypes** is not specified, the default encryption types, which are: aes256-cts-hmac-sha1-96, aes128-cts-hmac-sha1-96, des3-cbc-sha1. When running with a FIPS level greater than zero, all encryption types that are not FIPS compliant are removed, and if all of the encryption types specified in **default_tgs_enctypes** are not FIPS compliant, then the default encryption types will be used.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_host_realm (get Kerberos realm name)

Purpose

Gets the Kerberos realm name for a host name.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_host_realm (
    krb5_context                        context,
    const char *                        host,
    char ***                            realm_list)
```

Parameters

Input

context
> Specifies the Kerberos context.

host
> Specifies the host name. The local host name is used if NULL is specified for this parameter.

Output

realm_list
Returns an array of realm names. The last entry in the array is a NULL pointer. The
krb5_free_host_realm() routine should be called to release the realm list when it is no longer
needed.

Usage

The **krb5_get_host_realm()** routine returns a list of Kerberos realm names for the specified host
name. The entries in the [domain_realm] section of the Kerberos configuration file are used, unless
dns_lookup or ldap_lookup are specified. A direct match takes precedence over a suffix match. The
current implementation of this routine returns a single realm name. If no realm name is found, the
uppercased host domain is returned as the realm name.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_in_tkt_system (get initial KDC ticket)

Purpose

Gets an initial ticket from the local KDC using the current system identity.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_in_tkt_system (
    krb5_context                context,
    const krb5_flags            options,
    krb5_address * const *      addrs,
    krb5_enctype *              enctypes,
    krb5_ccache                 ccache,
    krb5_creds *                creds,
    krb5_kdc_rep **             ret_as_reply)
```

Parameters

Input

context
Specifies the Kerberos context.

options
Specifies KDC options as follows:

- KDC_OPT_FORWARDABLE - Obtain a forwardable ticket.

- KDC_OPT_PROXIABLE - Obtain a proxiable ticket.

- KDC_OPT_ALLOW_POSTDATE - Allow postdated tickets.

- KDC_OPT_RENEWABLE - Obtain a renewable ticket. The **renew_till** time must be set in the request.

- KDC_OPT_RENEWABLE_OK - A renewable ticket is acceptable if the KDC policy does not allow a
  ticket to be generated with the requested endtime.

Unrecognized options will no longer be diagnosed by the KDC; applications must ensure their options
have been honored by the KDC by checking the returned tickets.

addrs
Specifies the addresses to be placed in the ticket. If NULL is specified for this parameter, the local
system addresses are used. The address list is an array of **krb5_address** pointers. The end of the
array is indicated by a NULL pointer. No addresses are included in the initial ticket if the address array
consists of a single NULL entry. The ticket addresses determine which host systems can generate

requests that use the ticket. A mapped IPv6 address is stored in the ticket as the corresponding IPv4 address.

enctypes
Specifies an array of encryption types to be used. The last entry in the array must be ENCTYPE_NULL. If NULL is specified for this parameter, the default encryption types are used. The following encryption types may be specified:

- ENCTYPE_DES_CBC_CRC - 32-bit CRC checksum with DES encryption. This encryption type should be used for interoperability with older levels of Kerberos V5. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD4 - MD4 checksum with DES encryption (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD5 - MD5 checksum with DES encryption. (not valid in FIPS mode)
- ENCTYPE_DES_HMAC_SHA1 - SHA1 checksum with DES encryption and key derivation. (not valid in FIPS mode)
- ENCTYPE_DES3_CBC_SHA1 - SHA1 checksum with DES3 encryption and key derivation.
- ENCTYPE_AES128_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES128_CTS_HMAC_SHA256_128 SHA2 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA384_192 - SHA2 checksum with AES encryption.

Input/Output

ccache
Specifies the credentials cache handle. The credentials cache is initialized with the client name and the initial ticket is stored in the credentials cache for later use by the application. The initial ticket is not stored if NULL is specified for this parameter.

creds
Specifies attributes for the initial ticket. The server field must be set to the desired TGS service principal. The endtime field may be set to explicitly specify the ticket lifetime or it may be set to zero to use the default ticket lifetime. The **renew_till** field must be set if a renewable ticket is being requested. The **starttime** field must be set if a postdated ticket is being requested.

Upon completion of the request, **creds** is updated with the client name, the initial ticket, the session key, and the client address list. The **krb5_free_cred_contents()** or **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

Output

ret_as_reply
Returns the KDC reply. Specify NULL for this parameter if the KDC reply is not needed. The **krb5_free_kdc_rep()** routine should be called to release the reply when it is no longer needed.

Usage

The **krb5_get_in_tkt_system()** routine is called to obtain an initial ticket for the Kerberos principal associated with the current system identity. This initial ticket can then be used to obtain service tickets. The client must be in the same realm as the KDC in order to be able to obtain an initial ticket from the KDC. The initial ticket can be used to obtain tickets in the same realm or in different realms as long as the proper inter-realm trust relationships have been established.

As a general rule, the application should not specify the encryption types. This allows the encryption type to be determined by the Kerberos configuration profile.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

The Kerberos security server must be running on the local system in order to use this function. Otherwise, the function return value is set to KRB5_KDC_UNREACH.

Unrecognized options will no longer be diagnosed by the KDC. Applications must check that their options have been honored by the KDC by checking the returned tickets.

krb5_get_in_tkt_with_keytab (get initial ticket using key table)

Purpose

Gets an initial ticket using a key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_in_tkt_with_keytab (
    krb5_context                context,
    const krb5_flags            options,
    krb5_address * const *      addrs,
    krb5_enctype *              enctypes,
    krb5_preauthtype *          pre_auth_types,
    const krb5_keytab           keytab,
    krb5_ccache                 ccache,
    krb5_creds *                creds,
    krb5_kdc_rep **             ret_as_reply)
```

Parameters

Input

context
Specifies the Kerberos context.

options
Specifies KDC options as follows:

- KDC_OPT_FORWARDABLE - Obtain a forwardable ticket.

- KDC_OPT_PROXIABLE - Obtain a proxiable ticket.

- KDC_OPT_ALLOW_POSTDATE - Allow postdated tickets.

- KDC_OPT_RENEWABLE - Obtain a renewable ticket. The **renew_till** time must be set in the request.

- KDC_OPT_RENEWABLE_OK - A renewable ticket is acceptable if the KDC policy does not allow a ticket to be generated with the requested endtime.

Unrecognized options will no longer be diagnosed by the KDC; applications must ensure their options have been honored by the KDC by checking the returned tickets.

addrs
Specifies the addresses to be placed in the ticket. If NULL is specified for this parameter, the local system addresses are used. The address list is an array of **krb5_address** pointers. The end of the array is indicated by a NULL pointer. No addresses are included in the initial ticket if the address array consists of a single NULL entry. The ticket addresses determine which host systems can generate requests that use the ticket. A mapped IPv6 address is stored in the ticket as the corresponding IPv4 address.

enctypes
Specifies an array of encryption types to be used. The last entry in the array must be ENCTYPE_NULL. If NULL is specified for this parameter, the default encryption types are used. The following encryption types may be specified:

- ENCTYPE_DES_CBC_CRC - 32-bit CRC checksum with DES encryption. This encryption type should be used for interoperability with older levels of Kerberos V5. (not valid in FIPS mode)

- ENCTYPE_DES_CBC_MD4 - MD4 checksum with DES encryption. (not valid in FIPS mode)

- ENCTYPE_DES_CBC_MD5 - MD5 checksum with DES encryption. (not valid in FIPS mode)

- ENCTYPE_DES_HMAC_SHA1 - SHA1 checksum with DES encryption and key derivation. (not valid in FIPS mode and not valid when request FAST preauthentication)
- ENCTYPE_DES3_CBC_SHA1 - SHA1 checksum with DES3 encryption and key derivation.
- ENCTYPE_AES128_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES128_CTS_HMAC_SHA256_128 - SHA2 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA384_192 - SHA2 checksum with AES encryption.

pre_auth_types
Specifies an array of preauthentication types to be used. The last entry in the array must be KRB5_PADATA_NONE. If NULL is specified for this parameter, no preauthentication is done unless required by KDC policy (in which case the KDC provides the preauthentication types). If multiple preauthentication types are specified, the KDC is supposed to accept the request as long as it recognizes at least one of the preauthentication types. Unfortunately, early implementations of the KDC did not follow this rule and fail the request if the first preauthentication type is not recognized. The following preauthentication types may be specified:

- KRB5_PADATA_ENC_TIMESTAMP - Encrypted timestamp preauthentication.
- KRB5_PADATA_FX_FAST - FAST preauthentication is to be used to armor the encapsulated preauthentication data, known as FAST Factors. The supported FAST Factors are:

  - KRB5_PADATA_ENCRYPTED_CHALLENGE - Encrypted challenge preauthentication. (Note: To use FAST preauthentication, an anonymous PKINIT ticket (TGT) is required, see usage section)
  - KRB5_ENC_PADATA_REQ_ENC_PA_REP - FAST Negotiation preauthentication.

keytab
Specifies the key table containing the key for the client principal. The entry with the highest key version number is used. The default key table is used if NULL is specified for this parameter.

Input/Output

ccache
Specifies the credentials cache handle. The initial ticket is stored in the credentials cache for later use by the application. The credentials is not stored if NULL is specified for this parameter.

creds
Specifies the credentials that are used to obtain the initial ticket. The client and server fields must be set. The endtime field may be set to explicitly specify the ticket lifetime or it may be set to zero to use the default ticket lifetime. The **renew_till** field must be set if a renewable ticket is being requested. The **starttime** field must be set if a postdated ticket is being requested.

Upon completion of the request, **creds** is updated with the initial ticket, the session key, and the client address list. The **krb5_free_cred_contents()** or **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

Output

ret_as_reply
Returns the KDC reply. Specify NULL for this parameter if the KDC reply is not needed. The **krb5_free_kdc_rep()** routine should be called to release the reply when it is no longer needed

Usage

The **krb5_get_in_tkt_with_keytab()** routine is called to obtain an initial ticket using a key table. This initial ticket can then be used to obtain service tickets. The client must be in the same realm as the KDC in order to obtain an initial ticket from the KDC. The initial ticket can be used to obtain tickets in the same realm or in different realms as long as the proper inter-realm trust relationships have been established.

As a general rule, the application should not specify encryption or preauthentication types. This allows the encryption type to be determined by the Kerberos configuration profile and the preauthentication type

to be determined by the KDC policy. If FAST preauthentication is requested, an anonymous PKINIT ticket must be anchored in the input **krb5_context parameter via a call to krb5_set_fast_armor_ticket()**. To obtain an anonymous PKINIT ticket, see **krb5_get_in_tkt_with_pkinit()** for details.

The first encryption type specified (either explicitly or through the Kerberos configuration profile) is used for preauthentication types that require an encryption key. If the KDC returns a list of encryption types, the first supported encryption type is used for preauthentication data.

Unrecognized options will no longer be diagnosed by the KDC. Applications must check that their options have been honored by the KDC by checking the returned tickets.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_in_tkt_with_password (get initial ticket with text password)

Purpose

Gets an initial ticket using a text password.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_in_tkt_with_password (
    krb5_context                context,
    const krb5_flags            options,
    krb5_address * const *      addrs,
    krb5_enctype *              enctypes,
    krb5_preauthtype *          pre_auth_types,
    const char *                password,
    krb5_ccache                 ccache,
    krb5_creds *                creds,
    krb5_kdc_rep **             ret_as_reply)
```

Parameters

Input

context
> Specifies the Kerberos context.

options
> Specifies KDC options as follows:
>
> - KDC_OPT_FORWARDABLE - Obtain a forwardable ticket.
>
> - KDC_OPT_PROXIABLE - Obtain a proxiable ticket
>
> - KDC_OPT_ALLOW_POSTDATE - Allow postdated tickets.
>
> - KDC_OPT_RENEWABLE - Obtain a renewable ticket. The **renew_till** time must be set in the request.
>
> - KDC_OPT_RENEWABLE_OK - A renewable ticket is acceptable if the KDC policy does not allow a ticket to be generated with the requested endtime.
>
> Unrecognized options will no longer be diagnosed by the KDC; applications must ensure their options have been honored by the KDC by checking the returned tickets.

addrs
> Specifies the addresses to be placed in the ticket. If NULL is specified for this parameter, the local system addresses are used. The address list is an array of **krb5_address** pointers. The end of the array is indicated by a NULL pointer. No addresses are included in the initial ticket if the address array consists of a single NULL entry. The ticket addresses determine which host systems can generate requests that use the ticket. A mapped IPv6 address is stored in the ticket as the corresponding IPv4 address.

enctypes

Specifies an array of encryption types to be used. The last entry in the array must be ENCTYPE_NULL. If NULL is specified for this parameter, the default encryption types are used. The following encryption types may be specified:

- ENCTYPE_DES_CBC_CRC - 32-bit CRC checksum with DES encryption. This encryption type should be used for interoperability with older levels of Kerberos V5. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD4 - MD4 checksum with DES encryption. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD5 - MD5 checksum with DES encryption. (not valid in FIPS mode)
- ENCTYPE_DES_HMAC_SHA1 - SHA1 checksum with DES encryption and key derivation. (not valid in FIPS mode and not valid when request FAST preauthentication)
- ENCTYPE_DES3_CBC_SHA1 - SHA1 checksum with DES3 encryption and key derivation.
- ENCTYPE_AES128_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES128_CTS_HMAC_SHA256_128 - SHA2 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA384_192 - SHA2 checksum with AES encryption.

pre_auth_types

Specifies an array of preauthentication types to be used. The last entry in the array must be KRB5_PADATA_NONE. If NULL is specified for this parameter, no preauthentication is done unless required by KDC policy (in which case the KDC provides the preauthentication types). If multiple preauthentication types are specified, the KDC is supposed to accept the request as long as it recognizes at least one of the preauthentication types. Unfortunately, early implementations of the KDC did not follow this rule and fail the request if the first preauthentication type is not recognized. The following preauthentication types may be specified:

- KRB5_PADATA_ENC_TIMESTAMP - Encrypted timestamp preauthentication.
- KRB5_PADATA_FX_FAST - FAST preauthentication is to be used to armor the encapsulated preauthentication data, known as FAST Factors. The supported FAST Factors are:

  - KRB5_PADATA_ENCRYPTED_CHALLENGE - Encrypted challenge preauthentication. (Note: To use FAST preauthentication, an anonymous PKINIT ticket (TGT) is required, see usage section.
  - KRB5_ENC_PADATA_REQ_ENC_PA_REP - FAST Negotiation preauthentication.

password

Specifies the password string. This string is converted to a Kerberos key value using the rules for the first encryption type specified by the enctypes parameter. The user is prompted to enter the password if NULL is specified for this parameter.

Input/Output

ccache

Specifies the credentials cache handle. The initial ticket is stored in the credentials cache for later use by the application. The credentials are not stored if NULL is specified for this parameter.

creds

Specifies the credentials that are used to obtain the initial ticket. The client and server fields must be set. The endtime field may be set to explicitly specify the ticket lifetime or it may be set to zero to use the default ticket lifetime. The **renew_till** field must be set if a renewable ticket is being requested. The **starttime** field must be set if a postdated ticket is being requested.

Upon completion of the request, **creds** is updated with the initial ticket, the session key, and the client address list. The **krb5_free_cred_contents()** or **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

Output

ret_as_reply
> Returns the KDC reply. Specify NULL for this parameter if the KDC reply is not needed. The **krb5_free_kdc_rep()** routine should be called to release the reply when it is no longer needed.

Usage

The **krb5_get_in_tkt_with_password()** routine is called to obtain an initial ticket using a text password. This initial ticket can then be used to obtain service tickets. The client must be in the same realm as the KDC in order to obtain an initial ticket from the KDC. The initial ticket can be used to obtain tickets in the same realm or in different realms as long as the proper inter-realm trust relationships have been established.

As a general rule, the application should not specify encryption or preauthentication types. This allows the encryption type to be determined by the Kerberos configuration profile and the preauthentication type to be determined by the KDC policy.

If FAST preauthentication is requested, an anonymous PKINIT ticket must be anchored in the input **krb5_context parameter via a call to krb5_set_fast_armor_ticket()**. To obtain an anonymous PKINIT ticket, see **krb5_get_in_tkt_with_pkinit()** for details.

The first encryption type specified (either explicitly or through the Kerberos configuration profile) is used for preauthentication types that require an encryption key. If the KDC returns a list of encryption types, the first supported encryption type is used for preauthentication data.

Unrecognized options will no longer be diagnosed by the KDC. Applications must check that their options have been honored by the KDC by checking the returned tickets.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_in_tkt_with_pkinit (get initial ticket using public private key pair)

Purpose

Gets an initial ticket using a public private key pair.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_in_tkt_with_pkinit (
    krb5_context                    context
    const krb5_flags                options,
    krb5_address * const *          addrs,
    krb5_enctype *                  enctypes,
    krb5_ccache                     ccache,
    krb5_creds *                    creds,
    krb5_kdc_rep **                 ret_as_reply))
```

Parameters

Input

context
> Specifies the Kerberos context. This parameter cannot be NULL.

options
> Specifies KDC options as follows:

> - KDC_OPT_FORWARDABLE - Obtain a forwardable ticket.

> - KDC_OPT_PROXIABLE - Obtain a proxiable ticket.

- KDC_OPT_ALLOW_POSTDATE - Allow postdated tickets.
- KDC_OPT_RENEWABLE - Obtain a renewable ticket. The renew_till time must be set in the request.
- KDC_OPT_RENEWABLE_OK - A renewable ticket is acceptable if the KDC policy does not allow a ticket to be generated with the requested endtime. Unrecognized options will no longer be diagnosed by the KDC; applications must ensure their options have been honored by the KDC by checking the returned tickets.
- KDC_OPT_ANONYMOUS - Include this option when requesting an anonymous ticket. The client principal name in the **creds** parameter must be a KRB_NT_WELLKNOWN name type with two parts; first name element is "WELLKNOWN", the second name element is "ANONYMOUS."

addrs
    Specifies the addresses to be placed in the ticket. If NULL is specified for this parameter, the local system addresses are used. The address list is an array of krb5_address pointers. The end of the array is indicated by a NULL pointer. No addresses are included in the initial ticket if the address array consists of a single NULL entry. The ticket addresses determine which host systems can generate requests that use the ticket. A mapped IPv6 address is stored in the ticket as the corresponding IPv4 address.

enctypes
    Specifies an array of encryption types to be used. The last entry in the array must be ENCTYPE_NULL. If NULL is specified for this parameter, the default encryption types are used. The following encryption types may be specified:

- ENCTYPE_DES_CBC_CRC - 32-bit CRC checksum with DES encryption. This encryption type should be used for interoperability with older levels of Kerberos V5. (not valid in FIPS mode).
- ENCTYPE_DES_CBC_MD4 - MD4 checksum with DES encryption. (not valid in FIPS mode).
- ENCTYPE_DES_CBC_MD5 - MD5 checksum with DES encryption. (not valid in FIPS mode).
- ENCTYPE_DES_HMAC_SHA1 - SHA1 checksum with DES encryption and key derivation. (not valid in FIPS mode, and not a supported encryption type for an anonymous PKINIT ticket request).
- ENCTYPE_DES3_CBC_SHA1 - SHA1 checksum with DES3 encryption and key derivation.
- ENCTYPE_AES128_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES128_CTS_HMAC_SHA256_128 - SHA2 checksum with AES encryption
- ENCTYPE_AES256_CTS_HMAC_SHA384_192 - SHA2 checksum with AES encryption

Input/Output

ccache
    Specifies the credentials cache handle. The initial ticket is stored in the credentials cache for later use by the application. The credentials is not stored if NULL is specified for this parameter.

creds
    Specifies the credentials that are used to obtain the initial ticket. The client and server fields must be set. The endtime field may be set to explicitly specify the ticket lifetime or it may be set to zero to use the default ticket lifetime. The renew_till field must be set if a renewable ticket is being requested. The starttime field must be set if a postdated ticket is being requested. To request an anonymous PKINIT ticket, the client field must be set to the WELLKNOWN/ANONYMOUS principal name using the KRB5_NT_WELLKNOWN name type. The client realm field must identify the realm from which the anonymous ticket is being requested.

    When this function is used for anonymous PKINIT, the pkinit_rsa_protocol value, if specified in krb5.conf must not be 1. Otherwise this function call will fail to generate anonymous armor ticket.

    When an anonymous PKINIT ticket is obtained, the client realm name is changed to the anonymous realm name, WELLKNOWN:ANONYMOUS.

Upon completion of the request, creds is updated with the initial ticket, the session key, and the client address list. The krb5_free_cred_contents() or krb5_free_creds() routine should be called to release the credentials when they are no longer needed.

Output

ret_as_reply
Returns the KDC reply. Specify NULL for this parameter if the KDC reply is not needed. The krb5_free_kdc_rep() routine should be called to release the reply when it is no longer needed.

Usage

The krb5_get_in_tkt_with_pkinit() routine is called to obtain an initial ticket using the pkinit context which contains public private key information supplied by the client. This initial ticket can then be used to obtain service tickets. The client must be in the same realm as the KDC in order to obtain an initial ticket from the KDC. The initial ticket can be used to obtain tickets in the same realm or in different realms as long as the proper inter-realm trust relationships have been established.

The krb5_get_in_tkt_with_pkinit() routine may also be called to obtain an anonymous PKINIT ticket. In this case, the pkinit context only needs to contain certificates sufficient enough to validate the KDC certificate in the reply. The anonymous PKINIT ticket can then be used as an armor ticket to obtain an initial ticket in a more secure manner. Although the client realm name to which the krb5_creds structure is the realm name to which the initial ticket request is sent on entry to this routine, the realm name in both the returned ticket and the client realm name in the krb5_creds structure will be updated when this routine is successful to be the anonymous realm name, "WELLKNOWN:ANONYMOUS".

Unrecognized options will no longer be diagnosed by the KDC. Applications must check that their options have been honored by the KDC by checking the returned tickets

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_in_tkt_with_skey (get initial ticket using session key)

Purpose

Gets an initial ticket using a session key.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_in_tkt_with_skey (
    krb5_context              context,
    const krb5_flags          options,
    krb5_address * const *    addrs,
    krb5_enctype *            enctypes,
    krb5_preauthtype *        pre_auth_types,
    const krb5_keyblock *     key,
    krb5_ccache               ccache,
    krb5_creds *              creds,
    krb5_kdc_rep **           ret_as_reply)
```

Parameters

Input

context
Specifies the Kerberos context.

options
Specifies KDC options as follows:

- KDC_OPT_FORWARDABLE - Obtain a forwardable ticket.

- KDC_OPT_PROXIABLE - Obtain a proxiable ticket.
- KDC_OPT_ALLOW_POSTDATE - Allow postdated tickets.
- KDC_OPT_RENEWABLE - Obtain a renewable ticket. The renew_till time must be set in the request.
- KDC_OPT_RENEWABLE_OK - A renewable ticket is acceptable if the KDC policy does not allow a ticket to be generated with the requested endtime.

Unrecognized options will no longer be diagnosed by the KDC; applications must ensure their options have been honored by the KDC by checking the returned tickets.

addrs
Specifies the addresses to be placed in the ticket. If NULL is specified for this parameter, the local system addresses are used. The address list is an array of **krb5_address** pointers. The end of the array is indicated by a NULL pointer. No addresses are included in the initial ticket if the address array consists of a single NULL entry. The ticket addresses determine which host systems can generate requests that use the ticket. A mapped IPv6 address is stored in the ticket as the corresponding IPv4 address.

enctypes
Specifies an array of encryption types to be used. The last entry in the array must be ENCTYPE_NULL. If NULL is specified for this parameter, the default encryption types are used. The following encryption types may be specified:

- ENCTYPE_DES_CBC_CRC - 32-bit CRC checksum with DES encryption. This encryption type should be used for interoperability with older levels of Kerberos V5. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD4 - MD4 checksum with DES encryption. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD5 - MD5 checksum with DES encryption. (not valid in FIPS mode)
- ENCTYPE_DES_HMAC_SHA1 - SHA1 checksum with DES encryption and key derivation. (not valid in FIPS mode and not valid when request FAST preauthentication)
- ENCTYPE_DES3_CBC_SHA1 - SHA1 checksum with DES3 encryption and key derivation.
- ENCTYPE_AES128_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES256_CTS_HMAC_SHA1_96 - SHA1 checksum with AES encryption.
- ENCTYPE_AES128_CTS_HMAC_SHA256_128 - SHA2 checksum with AES encryption .
- ENCTYPE_AES256_CTS_HMAC_SHA384_192 - SHA2 checksum with AES encryption.

pre_auth_types
Specifies an array of preauthentication types to be used. The last entry in the array must be KRB5_PADATA_NONE. If NULL is specified for this parameter, no preauthentication is done unless required by KDC policy (in which case the KDC provides the preauthentication types). If multiple preauthentication types are specified, the KDC is supposed to accept the request as long as it recognizes at least one of the preauthentication types. Unfortunately, early implementations of the KDC did not follow this rule and fail the request if the first preauthentication type is not recognized. The following preauthentication types may be specified:

- KRB5_PADATA_ENC_TIMESTAMP - Encrypted timestamp preauthentication.
- KRB5_PADATA_FX_FAST - FAST preauthentication is to be used to armor the encapsulated preauthentication data, known as FAST Factors. The supported FAST Factors are:

  - KRB5_PADATA_ENCRYPTED_CHALLENGE - Encrypted challenge preauthentication. (Note: To use FAST preauthentication, an anonymous PKINIT ticket (TGT) is required, see usage section)
  - KRB5_ENC_PADATA_REQ_ENC_PA_REP - FAST Negotiation preauthentication

key
Specifies the key to be used. The default key table is used if NULL is specified for this parameter. The key must be the current encryption key for the client principal.

Input/Output

ccache

Specifies the credentials cache handle. The initial ticket is stored in the credentials cache for later use by the application. The credentials are not stored if NULL is specified for this parameter.

creds

Specifies the credentials that are used to obtain the initial ticket. The client and server fields must be set. The endtime field may be set to explicitly specify the ticket lifetime or it may be set to zero to use the default ticket lifetime. The **renew_till** field must be set if a renewable ticket is being requested. The starttime field must be set if a postdated ticket is being requested.

Upon completion of the request, **creds** is updated with the initial ticket, the session key, and the client address list. The **krb5_free_cred_contents()** or **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed.

Output

ret_as_reply

Returns the KDC reply. Specify NULL for this parameter if the KDC reply is not needed. The **krb5_free_kdc_rep()** routine should be called to release the reply when it is no longer needed.

Usage

The **krb5_get_in_tkt_with_skey()** routine is called to obtain an initial ticket using a session key. This initial ticket can then be used to obtain service tickets. The client must be in the same realm as the KDC in order to obtain an initial ticket from the KDC. The initial ticket can be used to obtain tickets in the same realm or in different realms as long as the proper inter-realm trust relationships have been established.

As a general rule, the application should not specify encryption or preauthentication types. This allows the encryption type to be determined by the Kerberos configuration profile and the preauthentication type to be determined by the KDC policy.

If FAST preauthentication is requested, an anonymous PKINIT ticket must be anchored in the input krb5_context parameter via a call to krb5_set_fast_armor_ticket(). To obtain an anonymous PKINIT ticket, see krb5_get_in_tkt_with_pkinit() for details.

The first encryption type specified (either explicitly or through the Kerberos configuration profile) is used for preauthentication types that require an encryption key. If the KDC returns a list of encryption types, the first supported encryption type is used for preauthentication data.

Unrecognized options will no longer be diagnosed by the KDC. Applications must check that their options have been honored by the KDC by checking the returned tickets.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_krbhst (return list of KDC hosts)

Purpose

Returns a list of KDC hosts for a Kerberos realm.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_krbhst (
    krb5_context              context,
    const krb5_data *         realm,
    char ***                  hostlist)
```

Parameters

Input

context
  Specifies the Kerberos context.

realm
  Specifies the Kerberos realm.

Output

hostlist
  Returns the KDC host list. The last entry in the list is a NULL pointer. The **krb5_free_krbhst()** routine should be called to release the host list when it is no longer needed.

Usage

The **krb5_get_krbhst()** routine returns a list of hosts in the specified realm that are running Kerberos KDC servers. The list is obtained from the Lightweight Directory Access Protocol (LDAP) directory, the domain name service (DNS) name server, or the [realms] section of the Kerberos configuration file.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_get_server_rcache (generate replay cache)

Purpose

Generates a replay cache for server use.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_get_server_rcache (
    krb5_context                    context,
    const krb5_data *               piece,
    krb5_rcache *                   ret_rcache)
```

Parameters

Input

context
  Specifies the Kerberos context.

piece
  Specifies the unique portion of the replay cache name.

Output

ret_rcache
  Returns the replay cache handle. The **krb5_rc_close()** routine should be called to close the replay cache when it is no longer needed.

Usage

The **krb5_get_server_rcache()** routine generates a unique replay cache name and then opens the replay cache. The *piece* parameter is used to differentiate this replay cache from others currently in use on the system by the same user. The generated cache name is in the form *rc_piece_uid* and uses the default replay cache type.

The replay cache is initialized if it can not be recovered. The clock skew value is obtained from the Kerberos context if it is necessary to initialize the cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

When the **use_dvipa_override** configuration option is set to 1, the selected replay cache will be overridden with a shared replay cache controlled by the SKRBKDC started task which does not use a name. This function will still work but the generation of a unique name will be meaningless and ignored.

krb5_init_context (create Kerberos context)

Purpose

Creates a Kerberos context.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_init_context (
    krb5_context *              context)
```

Parameters

Output

context
    Specifies the Kerberos context.

Usage

The **krb5_init_context()** routine creates a new Kerberos context and initializes it with default values obtained from the Kerberos configuration file. Each applications needs at least one Kerberos context. A context may be shared by multiple threads within the same process. Use the **krb5_free_context()** routine to release the context when it is no longer needed.

During the processing of this function, an attempt will be made to establish the FIPS level for the process if the **fipslevel** value defined in the Kerberos configuration profile is set to a valid value greater than -1. When **fipslevel** is omitted or set to a value of -1, the FIPS level for the process will not be changed. When a **fipslevel** value of 0 is specified, the FIPS level for the process will be set to the OFF state (FIPS mode is disabled). When a **fipslevel** value of 1, 2, or 3 is specified, your programs will be restricted to using FIPS compliant encryption and checksum types. If the FIPS level for the process has already been established prior to calling this function, the only change that may be accomplish during the processing of this function is to set the FIPS level to OFF. The FIPS level for a process is establish by the first call to a System SSL function, so it is recommended that either this function be called prior to the first System SSL function in your program, or set the **fipslevel** value to -1 in the Kerberos configuration profile to prevent krb5_init_context from attempting to set the FIPS level.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

Note: the result from setting the FIPS mode does not affect the function return value.

krb5_init_context_pkinit (update Kerberos context with pkinit values)

Purpose

Add to a Kerberos context with values specified in the configuration file for public private key authentication.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_init_context_pkinit (
    krb5_context                    context,
    char*                           realm,
    krb5_error_code *               warning_code))
```

Parameters

Input/Output

context
> Input is the context obtained from krb5_init_context.

> Output is an updated context with pkinit values obtained from the Kerberos configuration file.

Input

realm
> Specifies the Kerberos realm.

Output

warning_code
> Reports the warning code if there are less severe configuration errors and defaults values are used.

Usage

The krb5_init_context_pkinit() routine adds to a Kerberos context with pkinit values obtained from the Kerberos configuration file. The context must be created using krb5_init_context before this call. Use the krb5_free_context() routine to release the context when it is no longer needed.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_add_entry (add new key table entry)

Purpose

Adds a new entry to a key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_add_entry (
    krb5_context                    context,
    krb5_keytab                     ktid,
    krb5_keytab_entry *             entry)
```

Parameters

Input

context
> Specifies the Kerberos context.

ktid
> Specifies the key table handle.

entry
Specifies the entry to be added to the key table. The application is responsible for setting the *principal*, *vno*, and *key* fields in the entry. The **krb5_kt_add_entry()** routine sets the *timestamp* field to the current time.

Usage

The **krb5_kt_add_entry()** routine adds a new entry to a key table. No checking is done for duplicate entries. The key table type must support write operations.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

It is not necessary to add multiple entries to the key table for keys that use the same key generation algorithm. For example, encryption types ENCTYPE_DES_CBC_CRC and ENCTYPE_DES_CBC_MD5 both generate a 56-bit DES key using the same algorithm. So it is necessary to store just a single entry in the key table specifying one of these encryption types. The **krb5_kt_get_entry()** routine then returns this key table entry when either of these encryption types is specified.

krb5_kt_close (close key table)

Purpose

Closes a key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_close (
    krb5_context                    context,
    krb5_keytab                     ktid)
```

Parameters

Input

context
Specifies the Kerberos context.

ktid
Specifies the key table handle.

Usage

The **krb5_kt_close()** routine closes a key table. The key table handle may not be used once this routine completes.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_default (resolve default key table)

Purpose

Resolves the default key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_default (
```

```
    krb5_context                    context,
    krb5_keytab *                   ktid)
```

Parameters

Input

context
    Specifies the Kerberos context.

Output

ktid
    Returns the key table handle.

Usage

The **krb5_kt_default()** routine resolves the default key table and returns a handle that can be used to access the table. This is equivalent to calling the **krb5_kt_resolve()** routine with the name returned by the **krb5_kt_default_name()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

Krb5_ktclose must be called to free the returned key table handle, once key table processing is complete.

krb5_kt_default_name (return default key table name)

Purpose

Returns the default key table name.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_default_name (
    krb5_context                    context,
    char *                          name,
    int                             name_size)
```

Parameters

Input

context
    Specifies the Kerberos context.

name_size
    Specifies the size of the buffer pointed to by the name parameter. The size must be large enough to contain the key table name and the trailing delimiter. One way to do this is to allocate the buffer to be MAX_KEYTAB_NAME_LENGTH+1 bytes.

Output

name
    Returns the key table name.

Usage

The **krb5_kt_default_name()** routine returns the name of the default key table for the current user. If the KRB5_KTNAME environment variable is set, this is the name of the default key table. Otherwise, the key

table name is obtained from the *default_keytab_name* entry in the [libdefaults] section of the Kerberos configuration file. If this entry is not defined, the default key table name is **/etc/skrb/krb5.keytab**.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_end_seq_get (end sequential key table reading)

Purpose

Ends the sequential reading of the key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_end_seq_get (
    krb5_context                      context,
    krb5_keytab                       ktid,
    krb5_kt_cursor *                  cursor)
```

Parameters

Input

context
    Specifies the Kerberos context.

ktid
    Specifies the key table handle.

Input/Output

cursor
    Specifies the cursor created by the **krb5_kt_start_seq_get()** routine.

Usage

The **krb5_kt_end_seq_get()** routine unlocks the key table and releases the cursor. The cursor may not be used once **krb5_kt_end_seq_get()** has completed.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_free_entry (release key table storage)

Purpose

Releases the storage assigned to a key table entry.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_free_entry (
    krb5_context                      context,
    krb5_keytab_entry *               entry)
```

Parameters

Input

context
    Specifies the Kerberos context.

entry
Specifies the key table entry.

Usage

The **krb5_kt_free_entry()** routine releases the contents of a key table entry. It does not free the **krb5_keytab_entry** structure itself.

The function return value is always zero.

krb5_kt_get_entry (return key table entry)

Purpose

Returns an entry from the key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_get_entry (
    krb5_context                    context,
    krb5_keytab                     ktid,
    krb5_principal                  principal,
    krb5_kvno                       vno,
    krb5_enctype                    enctype,
    krb5_keytab_entry *             entry)
```

Parameters

Input

context
Specifies the Kerberos context.

ktid
Specifies the key table handle.

principal
Specifies the principal.

vno
Specifies the key version number for the key to be retrieved. Specify a version number of zero to retrieve the key with the highest version number.

enctype
Specifies the key encryption type. Specify an encryption type of zero if the encryption type does not matter.

Output

entry
Returns the contents of the key table entry. The **krb5_kt_free_entry()** routine should be called to release the entry contents when they are no longer needed.

Usage

The **krb5_kt_get_entry()** routine returns an entry from the key table for the specified principal. The entry returned is the first one found in the key table that matches the requested principal and version and uses a compatible encryption type. For example, an entry that uses ENCTYPE_DES_CBC_MD5 is compatible with a requested encryption type of ENCTYPE_DES_CBC_CRC.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_get_name (return key table name)

Purpose

Returns the key table name.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_get_name (
    krb5_context                    context,
    krb5_keytab                     ktid,
    char *                          name,
    int                             name_size)
```

Parameters

Input

context
> Specifies the Kerberos context.

ktid
> Specifies the key table handle.

name_size
> Specifies the size of the buffer pointed to by the **name** parameter. The size must be large enough to contain the key table name and the trailing delimiter. One way to do this is to allocate the buffer to be MAX_KEYTAB_NAME_LENGTH+1 bytes.

Output

name
> Returns the key table name.

Usage

The **krb5_kt_get_name()** routine returns the name of the key table. The returned name includes the key table type prefix.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_get_type (return key table type)

Purpose

Returns the key table type.

Format

```
#include <skrb/krb5.h>
char * krb5_kt_get_type (
    krb5_context                        context,
    krb5_keytab                         ktid)
```

Parameters

Input

context
   Specifies the Kerberos context.

ktid
   Specifies the key table handle.

Usage

The **krb5_kt_get_type()** routine returns the key table type.

The function return value is the address of the key table type. This is a read-only value and must not be freed by the application.

krb5_kt_next_entry (return key table next entry)

Purpose

Returns the next entry from the key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_next_entry (
    krb5_context                    context,
    krb5_keytab                     ktid,
    krb5_keytab_entry *             entry,
    krb5_kt_cursor *                cursor)
```

Parameters

Input

context
   Specifies the Kerberos context.

ktid
   Specifies the key table handle.

Input/Output

cursor
   Specifies the cursor created by the **krb5_kt_start_seq_get()** routine. The cursor is updated upon successful completion of this routine.

Output

entry
   Returns the contents of the table entry. The **krb5_kt_free_entry()** routine should be called to release the entry contents when they are no longer needed.

Usage

The **krb5_kt_next_entry()** reads the next entry from the key table and returns it to the application. The **krb5_kt_start_seq_get()** routine must be called to begin the sequential read operation. The **krb5_kt_next_entry()** routine is then called repeatedly to read table entries. Finally, the **krb5_kt_end_seq_get()** routine is called when no more entries are to be read.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_read_service_key (retrieve key table service key)

Purpose

Retrieves the service key from the key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_read_service_key (
    krb5_context                        context,
    krb5_pointer                        keytab_name
    krb5_principal                      principal,
    krb5_kvno                           vno,
    krb5_enctype                        enctype,
    krb5_keyblock **                    key)
```

Parameters

Input

context
　　Specifies the Kerberos context.

keytab_name
　　Specifies the key table name. If a NULL address is specified, the default key table is used.

principal
　　Specifies the service principal.

vno
　　Specifies the key version number for the key to be retrieved. Specify a version number of zero to retrieve the key with the highest version number.

enctype
　　Specifies the key encryption type. Specify an encryption type of zero if the encryption type does not matter.

Output

key
　　Returns the retrieved key. The **krb5_free_keyblock()** routine should be called to release the key when it is no longer needed.

Usage

The **krb5_kt_read_service_key()** routine retrieves the key for a service principal from a key table.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_register (define new key table type)

Purpose

Defines a new key table type.

Format

```
#include <skrb/krb5.h>
tkrb5_error_code krb5_kt_register (
    krb5_context                        context,
    krb5_kt_ops *                       ops)
```

Parameters

Input

context
Specifies the Kerberos context.

ops
Specifies the key table operations vector. This vector defines the routines that are called to perform the various key table operations for the new type.

Usage

The **krb5_kt_register()** routine registers a new key table type. An error is returned if the key table type has already been registered. Once the new type is registered, it can be used by any thread in the current process. The type is not known outside the current process and is no longer registered when the application ends.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_remove_entry (remove key table entry)

Purpose

Removes an entry from a key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_remove_entry (
    krb5_context                    context,
    krb5_keytab                     ktid,
    krb5_keytab_entry *             entry)
```

Parameters

Input

context
Specifies the Kerberos context.

ktid
Specifies the key table handle.

entry
Specifies the entry to be removed from the key table.

Usage

The **krb5_kt_remove_entry()** routine removes an entry from a key table. The key table type must support write operations.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_kt_resolve (resolve key table name)

Purpose

Resolves a key table name.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_resolve (
    krb5_context                   context,
    krb5_const char *              keytab_name,
    krb5_keytab *                  ktid)
```

Parameters

Input

context
> Specifies the Kerberos context.

keytab_name
> Specifies the key table name in the format *type:name*. The type must be a registered key table type and the name must uniquely identify a particular key table of the specified type.

Output

ktid
> Returns the key table handle.

Usage

The **krb5_kt_resolve()** routine resolves a key table name and returns a handle that can be used to access the table. The Kerberos runtime supports two key table types: FILE and WRFILE. Additional key table types can be registered by the application by calling the **krb5_kt_register()** routine. If no type is specified, the default is FILE.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

Krb5_ktclose must be called to free the returned key table handle, once key table processing is complete.

krb5_kt_start_seq_get (sequentially retrieve entries from key table)

Purpose

Starts sequentially retrieving entries from the key table.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_kt_start_seq_get (
    krb5_context                   context,
    krb5_keytab                    ktid,
    krb5_kt_cursor *               cursor)
```

Parameters

Input

context
> Specifies the Kerberos context.

ktid
> Specifies the key table handle.

Output

cursor
> Returns the cursor. The **krb5_kt_end_seq_get()** routine should be called to release the cursor at the completion of the sequential read operation.

Usage

The **krb5_kt_start_seq_get()** routine prepares for sequentially reading entries in the key table. The **krb5_kt_next_entry()** routine is called repeatedly to retrieve each successive table entry. The **krb5_kt_end_seq_get()** routine is called at the completion of the read operation.

The key table is locked when the **krb5_kt_start_seq_get()** routine is called and remains locked until the **krb5_kt_end_seq_get() routine** is called. Write access to the key table by other processes and threads is blocked until the table is unlocked. After the **krb5_kt_start_seq_get()** routine has been called, the current thread may not call any other key table functions except **krb5_kt_next_entry()** and **krb5_kt_end_seq_get()** for the specified table.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_md4_crypto_compat_ctl (set compatibility mode for MD4 checksum generation)

Purpose

Sets the compatibility mode for MD4 checksum generation.

Format

```
#include <skrb/krb5.h>
void krb5_md4_crypto_compat_ctl (
    krb5_boolean                    compat_mode)
```

Parameters

Input

compat_mode
> Specifies the compatibility mode as TRUE or FALSE.

Usage

The **krb5_md4_crypto_compat_ctl()** routine sets the compatibility mode for MD4 DES checksum generation. Early beta levels of Kerberos Version 5 computed the MD4 DES checksum incorrectly. Enabling compatibility mode causes the Kerberos runtime to generate the MD4 DES checksum in the same way while disabling compatibility mode causes the Kerberos runtime to generate the checksum correctly.

MD4 compatibility mode is set for the entire process by this routine and overrides the compatibility mode set by the *rsa_md4_des_compat* entry in the Kerberos configuration file.

krb5_md5_crypto_compat_ctl (set compatibility mode for MD5 checksum generation)

Purpose

Sets the compatibility mode for MD5 checksum generation.

Format

```
#include <skrb/krb5.h>

void krb5_md5_crypto_compat_ctl (
    krb5_boolean                    compat_mode)
```

Parameters

Input

compat_mode
Specifies the compatibility mode as TRUE or FALSE.

Usage

The **krb5_md5_crypto_compat_ctl()** routine sets the compatibility mode for MD5 DES checksum generation. Early beta levels of Kerberos Version 5 computed the MD5 DES checksum incorrectly. Enabling compatibility mode causes the Kerberos runtime to generate the MD5 DES checksum in the same way while disabling compatibility mode causes the Kerberos runtime to generate the checksum correctly.

MD5 compatibility mode is set for the entire process by this routine and overrides the compatibility mode set by the *rsa_md5_des_compat* entry in the Kerberos configuration file.

krb5_mk_error (create Kerberos KRB_ERROR message)

Purpose

Creates a Kerberos KRB_ERROR message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_mk_error (
    krb5_context                    context,
    const krb5_error *              dec_err,
    krb5_data *                     enc_err)
```

Parameters

Input

context
Specifies the Kerberos context.

dec_err
Specifies the **krb5_error** structure that is to be encoded.

Output

enc_err
Returns the encoded **krb5_error** structure as a byte stream. The storage pointed to by the data field of the **krb5_data** structure should be freed by the application when it is no longer needed.

Usage

The **krb5_mk_error()** routine creates a Kerberos KRB_ERROR message. This message is then sent to the remote partner instead of sending a reply message. For example, if an error is detected while processing an AP_REQ message, the application returns a KRB_ERROR message instead of an AP_REP message.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_mk_priv (create Kerberos KRB_PRIV message)

Purpose

Creates a Kerberos KRB_PRIV message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_mk_priv (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    const krb5_data *               userdata,
    krb5_data *                     out_data,
    krb5_replay_data *              replay_data)
```

Parameters

Input

context
Specifies the Kerberos context.

userdata
Specifies the application data for the KRB_PRIV message.

Input/Output

auth_context
Specifies the authentication context.

Output

out_data
Returns the KRB_PRIV message. The storage pointed to by the data field of the returned parameter should be freed by the application when it is no longer needed, by calling **krb5_free_data_contents()**.

replay_data
Returns replay information to the caller. This parameter is required if the KRB5_AUTH_CONTEXT_RET_TIME or KRB5_AUTH_CONTEXT_RET_SEQUENCE flag is set in the authentication context. Otherwise, NULL may be specified for this parameter.

Usage

The **krb5_mk_priv()** routine creates a KRB_PRIV message using data supplied by the application. This is similar to the **krb5_mk_safe()** routine, but the message is encrypted and integrity-protected rather than just integrity-protected. The **krb5_rd_priv()** routine decrypts and validates the message integrity. The authentication context specifies the checksum type, the data encryption type, the keyblock used to seed the checksum, the addresses of the sender and receiver, and the replay cache. The local address in the authentication context is used to create the KRB_PRIV message and must be present. The remote address is optional. The authentication context flags determine whether sequence numbers or timestamps should be used to identify the message. One of these methods must be used for a successful KRB_PRIV message.

The encryption type is taken from the keyblock in the authentication context. If the initial vector has been set in the authentication context, it is used as the initialization vector for the encryption (if the encryption type supports initialization) and its contents are replaced with the last block of encrypted data upon return.

If timestamps are used (KRB5_AUTH_CONTEXT_DO_TIME is set), an entry describing the message is entered in the replay cache so that callers may detect if this message is sent back to them by an attacker. An error is returned if the authentication context does not specify a replay cache.

If sequence numbers are used (KRB5_AUTH_CONTEXT_DO_SEQUENCE or KRB5_AUTH_CONTEXT_RET_SEQUENCE is set), then the local sequence number in the authentication context is placed in the protected message as its sequence number.

The encryption key is obtained from the local subkey, the remote subkey, or the session key, in that order. The application is responsible for setting a checksum type in the authentication context that is compatible with the encryption key. For example, an error is returned if a DES3 encryption key is used with a DES checksum type.

Due to government export regulations, some encryption algorithms may not be available on the current system. If the requested encryption algorithm is valid but not available, the function return value is set to KRB5_NO_CONF.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_mk_rep (create Kerberos AP_REP message)

Purpose

Creates a Kerberos AP_REP message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_mk_rep (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    krb5_data *                     out_data)
```

Parameters

Input

context
    Specifies the Kerberos context.

auth_context
    Specifies the authentication context.

Output

out_data
    Returns the AP_REP message. The storage pointed to by the data field of the **krb5_data** structure should be freed by the application when it is no longer needed.

Usage

The **krb5_mk_rep()** routine creates an AP_REP message using information in the authentication context. An AP_REP message is returned to the partner application after processing an AP_REQ message received from the partner application. The information in the authentication context is set by the **krb5_rd_req()** routine when it processes the AP_REQ message.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_mk_req (create Kerberos AP_REQ message)

Purpose

Creates a Kerberos AP_REQ message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_mk_req (
    krb5_context                        context,
    krb5_auth_context *                 auth_context,
    const krb5_flags                    ap_req_options,
    char *                              service,
    char *                              hostname,
    krb5_data *                         in_data,
    krb5_ccache                         ccache,
    krb5_data *                         out_data)
```

Parameters

Input

context
    Specifies the Kerberos context.

ap_req_options
    Specifies request options as follows:

- AP_OPTS_USE_SESSION_KEY - Use session key instead of server key for the service ticket. The credentials must include a ticket that is encrypted in the session key.

- AP_OPTS_MUTUAL_REQUIRED - Mutual authentication required.

  When both the application client and the application server support the Kerberos Cryptosystem Negotiation Extension and both are capable of using an encryption type that is stronger than the session key selected by the KDC, a new session key will be selected during the mutual authentication using the stronger encryption type.

- AP_OPTS_USE_SUBKEY - Generate a subsession key from the current session key obtained from the credentials.

service
    Specifies the name of the service.

hostname
    Specifies the host name that identifies the desired service instance.

in_data
    Specifies the application data whose checksum is to be included in the authenticator. Specify NULL for this parameter if no checksum is to be included in the authenticator.

ccache
    Specifies the credentials cache that is to be used to obtain credentials to the desired service.

Input/Output

auth_context
    Specifies the authentication context. A new authentication context is created and returned in this parameter if the value is NULL.

Output

out_data
    Returns the generated AP_REQ message. The storage pointed to by the data field in the returned **krb5_data** structure should be freed by the application when it is no longer needed.

Usage

The **krb5_mk_req()** routine generates an AP_REQ message. The checksum of the application data is included in the authenticator that is part of the AP_REQ message. This message is then sent to the partner application, which calls the **krb5_rd_req()** routine to validate the authenticity of the message. The checksum method set in the authentication context is used to generate the checksum.

The **krb5_sname_to_principal()** routine is called to convert the *service* and *hostname* parameters to a Kerberos principal. The **krb5_get_host_realm()** routine is called to convert the *hostname* parameter to a Kerberos realm. If the credentials cache does not already contain a service ticket for the target server, the Kerberos runtime issues a default TGS request to obtain the credentials and stores them in the cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_mk_req_extended (create Kerberos AP_REQ message)

Purpose

Creates a Kerberos AP_REQ message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_mk_req_extended (
    krb5_context                    context,
    krb5_auth_context *             auth_context,
    const krb5_flags                ap_req_options,
    krb5_data *                     appl_data,
    krb5_creds *                    in_creds,
    krb5_data *                     out_data)
```

Parameters

Input

context
　　Specifies the Kerberos context.

ap_req_options
　　Specifies request options as follows:

- AP_OPTS_USE_SESSION_KEY - Use session key instead of server key for the service ticket. The credentials must include a ticket that is encrypted in the session key.

- AP_OPTS_MUTUAL_REQUIRED - Mutual authentication required.

  When both the application client and the application server support the Kerberos Cryptosystem Negotiation Extension and both are capable of using an encryption type that is stronger than the session key selected by the KDC, a new session key will be selected during the mutual authentication using the stronger encryption type.

- AP_OPTS_USE_SUBKEY - Generate a subsession key from the current session key obtained from the credentials.

appl_data
　　Specifies the application data whose checksum is to be included in the authenticator. Specify NULL for this parameter if no checksum is to be included in the authenticator.

in_creds
　　Specifies the credentials for the specified service.

Input/Output

auth_context
> Specifies the authentication context. A new authentication context is created and returned in this parameter if the value is NULL.

Output

out_data
> Returns the generated AP_REQ message. The storage pointed to by the data field in the returned **krb5_data** structure should be freed by the application when it is no longer needed.

Usage

The **krb5_mk_req_extended()** routine is similar to the **krb5_mk_req()** routine but the caller passes the actual credentials as a parameter instead of letting the Kerberos runtime construct the credentials.

The **krb5_mk_req_extended()** routine generates an AP_REQ message. The checksum of the application data is included in the authenticator that is part of the AP_REQ message. This message is then sent to the partner application, which calls the **krb5_rd_req()** routine to validate the authenticity of the message. The checksum method set in the authentication context is used to generate the checksum.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_mk_safe (create Kerberos KRB_SAFE message)

Purpose

Creates a Kerberos KRB_SAFE message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_mk_safe (
    krb5_context                context,
    krb5_auth_context           auth_context,
    const krb5_data *           userdata,
    krb5_data *                 out_data,
    krb5_replay_data *          replay_data)
```

Parameters

Input

context
> Specifies the Kerberos context.

userdata
> Specifies the application data for the KRB_SAFE message.

Input/Output

auth_context
> Specifies the authentication context.

Output

out_data
> Returns the KRB_SAFE message. The storage pointed to by the data field of the returned parameter should be freed by the application when it is no longer needed, by calling **krb5_free_data_contents**.

replay_data

Returns replay information to the caller. This parameter is required if the KRB5_AUTH_CONTEXT_RET_TIME or KRB5_AUTH_CONTEXT_RET_SEQUENCE flag is set in the authentication context. Otherwise, NULL may be specified for this parameter.

Usage

The **krb5_mk_safe()** routine creates a KRB_SAFE message using data supplied by the application. Messages created by the **krb5_mk_safe()** routine are integrity-protected. The **krb5_rd_safe()** routine returns an error if the message has been modified. The authentication context specifies the checksum type, the keyblock used to seed the checksum, the addresses of the sender and receiver, and the replay cache. The local address in the authentication context is used to create the KRB_SAFE message and must be present. The remote address is optional. The authentication context flags determine whether sequence numbers or timestamps should be used to identify the message. One of these methods must be used for a successful KRB_SAFE message.

If timestamps are used (KRB5_AUTH_CONTEXT_DO_TIME is set), an entry describing the message is entered in the replay cache so that callers can detect if this message is sent back to them by an attacker. An error is returned if the authentication context does not specify a replay cache.

If sequence numbers are used (KRB5_AUTH_CONTEXT_DO_SEQUENCE or KRB5_AUTH_CONTEXT_RET_SEQUENCE is set), then the local sequence number in the authentication context is placed in the protected message as its sequence number.

The encryption key is obtained from the local subkey, the remote subkey, or the session key, in that order. The application is responsible for setting a checksum type in the authentication context that is compatible with the encryption key. For example, an error is returned if a DES3 encryption key is used with a DES checksum type.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_os_hostaddr (return network addresses)

Purpose

Returns the network addresses used by a specific host system.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_os_hostaddr (
    krb5_context                        context,
    const char *                        host,
    krb5_address ***                    addrs)
```

Parameters

Input

context

Specifies the Kerberos context.

host

Specifies the name of the host system. The name must be acceptable for use with the **getaddrinfo()** system function.

Output

addrs
> Returns an array of **krb5_address** pointers. The last entry in the array is a NULL pointer. The **krb5_free_addresses()** routine should be called to release the address array when it is no longer needed.

Usage

The **krb5_os_hostaddr()** routine returns the network addresses that are available on the specified host system. Only the AF_INET and AF_INET6 address families are supported. The **getaddrinfo()** system function is used to look up the addresses assigned to the specified host. A mapped IPv6 address is returned as the corresponding IPv4 address.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_os_localaddr (return network addresses)

Purpose

Returns the network addresses used by the local system.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_os_localaddr (
    krb5_context                    context,
    krb5_address ***                addrs)
```

Parameters

Input

context
> Specifies the Kerberos context.

Output

addrs
> Returns an array of **krb5_address** pointers. The last entry in the array is a NULL pointer. The **krb5_free_addresses()** routine should be called to release the address array when it is no longer needed.

Usage

The **krb5_os_localaddr()** routine returns the network addresses that are available on the local system. Only the AF_INET and AF_INET6 address families are supported. A mapped IPv6 address is returned as the corresponding IPv4 address.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_parse_name (create Kerberos principal from text string)

Purpose

Creates a Kerberos principal from a text string.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_parse_name (
    krb5_context                context,
    const char *                name,
    krb5_principal *            principal)
```

Parameters

Input

context
Specifies the Kerberos context.

name
Specifies the string to be parsed. The string must be in the format *name@realm*.

Output

principal
Returns the Kerberos principal. The **krb5_free_principal()** routine should be called to release the principal when it is no longer needed.

Usage

The **krb5_parse_name()** routine converts a text string into a Kerberos principal. The string must be in the format *name@realm*. If the realm is not specified, the default realm is used. Each forward slash in the name starts a new name component unless it is escaped by preceding the forward slash with a backward slash. Forward slashes in the realm are not treated as component separators and are copied unchanged.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_principal_compare (compare two Kerberos principals)

Purpose

Compares two Kerberos principals.

Format

```
#include <skrb/krb5.h>
krb5_boolean krb5_principal_compare (
    krb5_context                    context,
    krb5_const_principal            princ1,
    krb5_const_principal            princ2)
```

Parameters

Input

context
Specifies the Kerberos context.

princ1
Specifies the first principal to be compared.

princ2
Specifies the second principal to be compared.

Usage

The **krb5_principal_compare()** routine compares two Kerberos principals. The function return value is TRUE if the principals are the same and FALSE if they are not the same.

krb5_random_confounder (create random confounder)

Purpose

Creates a random confounder.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_random_confounder (
    krb5_context                    context,
    int                             buffer_size,
    krb5_pointer                    output_buffer)
```

Parameters

Input

context
    Specifies the Kerberos context.

buffer_size
    Specifies the size of the output buffer.

Output

output_buffer
    Specifies the buffer to receive the confounder.

Usage

The **krb5_random_confounder()** routine creates a random value that can be used as a confounder when encrypting data. A confounder is used to initialize the encryption-block chaining value so that the encrypted result is different each time a data value is encrypted even when the data value and encryption key are not changed.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rc_close (close a replay cache)

Purpose

Closes a replay cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_close (
    krb5_context                      context,
    krb5_rcache                       rcache)
```

Parameters

Input

context
Specifies the Kerberos context.

rcache
Specifies the replay cache handle.

Usage

The **krb5_rc_close()** routine closes a replay cache. The cache handle may not be used once this routine completes.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rc_default (resolve default replay cache)

Purpose

Resolves the default replay cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_default (
    krb5_context                      context,
    krb5_rcache *                     rcache)
```

Parameters

Input

context
Specifies the Kerberos context.

Output

rcache
Returns the replay cache handle.

Usage

The **krb5_rc_default()** routine resolves the default replay cache and returns a handle that can be used to access the table. This is equivalent to calling the **krb5_rc_resolve()** routine with the name returned by the **krb5_rc_default_name()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

When the **use_dvipa_override** configuration option is set to 1, the selected replay cache will be overridden with a shared replay cache controlled by the SKRBKDC started task which does not use a name. This function will still work but the generation of a unique name will be meaningless and ignored.

krb5_rc_default_name (return default replay cache name)

Purpose

Returns the default replay cache name.

Format

```
#include <skrb/krb5.h>
char * krb5_rc_default_name (
    krb5_context              context)
```

Parameters

Input

context
  Specifies the Kerberos context.

Usage

The **krb5_rc_default_name()** routine returns the name of the default replay cache for the current user. The KRB5RCACHENAME environment variable defines the default replay cache name.

The function return value is the default replay cache name or NULL if the default name has not been set. The return value is the address of a read-only string and must not be freed by the application.

When the **use_dvipa_override** configuration option is set to 1, the selected replay cache will be overridden with a shared replay cache controlled by the SKRBKDC started task which does not use a name. This function will still work but the generation of a unique name will be meaningless and ignored.

krb5_rc_destroy (delete replay cache)

Purpose

Deletes a replay cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_destroy (
    krb5_context              context,
    krb5_rcache               rcache)
```

Parameters

Input

context
  Specifies the Kerberos context.
rcache
  Specifies the replay cache handle.

Usage

The **krb5_rc_destroy()** routine closes and deletes a replay cache. The cache handle may not be used after this routine completes.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rc_expunge (delete replay cache expired entries)

Purpose

Deletes expired entries from the replay cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_expunge (
    krb5_context                     context,
    krb5_rcache                      rcache)
```

Parameters

Input

context
    Specifies the Kerberos context.

rcache
    Specifies the replay cache handle.

Usage

The **krb5_rc_expunge()** routine deletes expired entries from the replay cache. The entry lifespan is set by the **krb5_rc_initialize()** routine. This routine should be called periodically to clean up the replay cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rc_free_entry_contents (release storage)

Purpose

Releases the storage associated with a replay cache entry.

Format

```
#include <skrb/krb5.h>
void krb5_rc_free_entry_contents (
    krb5_context                     context,
    krb5_donot_replay *              entry)
```

Parameters

Input

context
    Specifies the Kerberos context.

entry
    Specifies the entry to be released.

Usage

The **krb5_rc_free_entry_contents()** releases the contents of a replay entry. The **krb5_donot_replay** structure itself is not released.

krb5_rc_get_lifespan (return authenticator lifespan)

Purpose

Returns the authenticator lifespan for entries in the replay cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_get_lifespan (
    krb5_context                      context,
    krb5_rcache                       rcache,
    krb5_deltat *                     span)
```

Parameters

Input

context
  Specifies the Kerberos context.

rcache
  Specifies the replay cache handle.

Output

span
  Returns the authenticator lifespan in seconds.

Usage

The **krb5_rc_get_lifespan()** routine returns the authenticator lifespan that was set by the **krb5_rc_initialize()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rc_get_name (return replay cache name)

Purpose

Returns the replay cache name.

Format

```
#include <skrb/krb5.h>
char * krb5_rc_get_name (
    krb5_context            context,
    krb5_rcache             rcache)
```

Parameters

Input

context
  Specifies the Kerberos context.

rcache
  Specifies the replay cache handle.

Usage

The **krb5_rc_get_name()** routine returns the name of the replay cache. The returned name does not include the replay cache type prefix.

The function return value is the address of the replay cache name. This is a read-only value and must not be freed by the application.

When the **use_dvipa_override** configuration option is set to 1, the selected replay cache will be overridden with a shared replay cache controlled by the SKRBKDC started task which does not use a name. This function will still work and it will return a name but the name will be meaningless.

krb5_rc_get_type (return replay cache type)

Purpose

Returns the replay cache type.

Format

```
#include <skrb/krb5.h>
char * krb5_rc_get_type (
    krb5_context           context,
    krb5_rcache            rcache)
```

Parameters

Input

context
    Specifies the Kerberos context.

rcache
    Specifies the replay cache handle.

Usage

The **krb5_rc_get_type()** routine returns the replay cache type.

The function return value is the address of the replay cache type. This is a read-only value and must not be freed by the application.

krb5_rc_initialize (initialize replay cache)

Purpose

Initializes the replay cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_initialize (
    krb5_context                     context,
    krb5_rcache                      rcache,
    krb5_deltat                      span)
```

Parameters

Input

context
    Specifies the Kerberos context.

rcache
    Specifies the replay cache handle.

span
>   Specifies the authenticator lifespan in seconds.

Usage

The **krb5_rc_initialize()** routine initializes a replay cache. Any existing cache entries are deleted. The authenticator lifespan indicates how long an authenticator remains valid. Once an authenticator has expired, its replay cache entry can be deleted by calling the **krb5_rc_expunge()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rc_recover (recover replay cache)

Purpose

Recovers the replay cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_recover (
    krb5_context                    context,
    krb5_rcache                     rcache)
```

Parameters

Input

context
>   Specifies the Kerberos context.

rcache
>   Specifies the replay cache handle.

Usage

The **krb5_rc_recover()** routine reads a replay cache into storage after the application has been restarted. Either **krb5_rc_recover()** or **krb5_rc_initialize()** must be called before any replay entries can be added to the replay cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rc_register_type (define new replay cache type)

Purpose

Defines a new replay cache type.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_register_type (
    krb5_context                    context,
    krb5_rc_ops *                   ops)
```

Parameters

Input

context
Specifies the Kerberos context.

ops
Specifies the replay cache operations vector. This vector defines the routines that is called to perform the various replay cache operations for the new type.

Usage

The **krb5_rc_register_type()** routine registers a new replay cache type. An error is returned if the replay cache type has already been registered. Once the new type is registered, it can be used by any thread in the current process. The type is not known outside the current process and is no longer registered when the application ends.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rc_resolve (resolve replay cache name)

Purpose

Resolves a replay cache name.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_resolve (
    krb5_context                    context,
    krb5_rcache *                   rcache,
    char *                          name)
```

Parameters

Input

context
Specifies the Kerberos context.

name
Specifies the replay cache name in the format *type:name*. The type must be a registered replay cache type and the name must uniquely identify a particular replay cache of the specified type.

Output

rcache
Returns the replay cache handle.

Usage

The krb5_rc_resolve() routine resolves a replay cache name and returns a handle that can be used to access the cache. After successfully calling krb5_rc_resolve(), the application should call either the krb5_rc_recover() or the krb5_rc_initialize() routine. This initializes the in-storage replay cache structures. The use of in-storage structures significantly improves performance but means that multiple replay cache handles should not be opened for the same replay cache.

When the **use_dvipa_override** configuration option is set to 1, the selected replay cache will be overridden with a shared replay cache controlled by the SKRBKDC started task which does not use a name. This function will still work but the generation of a unique name will be meaningless and ignored.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rc_store (store new replay cache entry)

Purpose

Stores a new entry in the replay cache.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rc_store (
    krb5_context                context,
    krb5_rcache                 rcache,
    krb5_donot_replay *         replay)
```

Parameters

Input

context
    Specifies the Kerberos context.

rcache
    Specifies the replay cache handle.

replay
    Specifies the replay entry.

Usage

The **krb5_rc_store()** routine stores a new entry in the replay cache after verifying that the entry is not already in the cache. The **krb5_auth_to_rep()** routine can be used to create a replay entry from a Kerberos authenticator. The **krb5_rc_expunge()** routine should be called periodically to purge expired entries from the replay cache.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rd_error (process Kerberos KRB_ERROR message)

Purpose

Processes a Kerberos KRB_ERROR message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rd_error (
    krb5_context                context,
    const krb5_data *           enc_err,
    krb5_error **               dec_err)
```

Parameters

Input

context
    Specifies the Kerberos context.

enc_err
    Specifies the error message created by the krb5_mk_error() routine.

Output

dec_err
    Returns the decoded error message. The **krb5_free_error()** routine should be called to release the **krb5_error** structure when it is no longer needed.

Usage

The **krb5_rd_error()** routine processes a KRB_ERROR message created by the **krb5_mk_error()** routine and returns a **krb5_error** structure.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rd_priv (process Kerberos KRB_PRIV message)

Purpose

Processes a Kerberos KRB_PRIV message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rd_priv (
    krb5_context                context,
    krb5_auth_context           auth_context,
    const krb5_data *           in_data,
    krb5_data *                 out_data,
    krb5_replay_data *          replay_data)
```

Parameters

Input

context
    Specifies the Kerberos context.

in_data
    Specifies the buffer containing the KRB_PRIV message.

Input/Output

auth_context
    Specifies the authentication context.

Output

out_data
    Returns the application data supplied to the **krb5_mk_priv()** routine. The application should release the data when it is no longer needed by calling the **krb5_free_data_contents()** routine.

replay_data
    Returns replay information to the caller. This parameter is required if the KRB5_AUTH_CONTEXT_RET_TIME or KRB5_AUTH_CONTEXT_RET_SEQUENCE flag is set in the authentication context. Otherwise, NULL may be specified for this parameter.

Usage

The **krb5_rd_priv()** routine processes a KRB_PRIV message and extracts the application data after verifying its integrity. If timestamps are being used, the message is stored in the replay cache associated with the authentication context.

The keyblock used for decrypting the data and for verifying message integrity is obtained from the authentication context. The first non-NULL keyblock is used by checking the **local_subkey**, **remote_subkey**, or **keyblock**, in that order. If the initialization vector in the authentication context has been set, it is used to initialize the decryption (if the encryption type supports initialization) and its contents are replaced with the last block of encrypted data in the message upon return.

The remote address in the authentication context must be present. It specifies the address of the sender. The address type used for the **krb5_rd_priv()** routine must be the same as the address type used for the **krb5_mk_priv()** routine. An error is returned if the address in the message does not match the remote address in the authentication context.

The local address in the authentication context is optional. If it is present, then it must match the receiver address in the message. Otherwise, the receiver message in the message must match one of the local addresses returned by the **krb5_os_localaddr()** routine.

If message sequence numbers are being used (KRB5_AUTH_CONTEXT_DO_SEQUENCE is set in the authentication context), the remote sequence number in the authentication context must match the sequence number in the message.

If timestamps are being used (KRB5_AUTH_CONTEXT_DO_TIME is set in the authentication context), the timestamp in the message must be within the Kerberos clock skew for the current time. In addition, the message must not be found in the replay cache obtained from the authentication context.

Due to government export regulations, some encryption algorithms may not be available on the current system. If the requested encryption algorithm is valid but not available, the function return value is set to KRB5_NO_CONF.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rd_rep (process a Kerberos AP_REP message)

Purpose

Processes a Kerberos AP_REP message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rd_rep (
    krb5_context                    context,
    krb5_auth_context               auth_context,
    const krb5_data *               in_data,
    krb5_ap_rep_enc_part **         reply)
```

Parameters

Input

context
  Specifies the Kerberos context.

in_data
  Specifies the buffer containing the AP_REP message.

Input/Output

auth_context
Specifies the authentication context.

Output

reply
Returns the decrypted reply data. The **krb5_free_ap_rep_enc_part()** routine should be called to release the reply when it is no longer needed.

Usage

The **krb5_rd_rep()** routine processes an AP_REP message created by the **krb5_mk_rep()** routine. The authentication context is updated with sequencing information obtained from the reply message.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_rd_req (process a Kerberos AP_REQ message)

Purpose

Processes a Kerberos AP_REQ message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rd_req (
    krb5_context                   context,
    krb5_auth_context *            auth_context,
    const krb5_data *              in_data,
    krb5_const_principal           server,
    krb5_keytab                    keytab,
    krb5_flags *                   ap_req_options,
    krb5_ticket **                 ticket)
```

Parameters

Input

context
Specifies the Kerberos context.

in_data
Specifies the buffer containing the AP_REQ message.

server
Specifies the server name. The server principal in the AP_REQ must be the same as the principal specified by this parameter. Specify NULL if any server principal is acceptable.

keytab
Specifies the key table that contains the server key. The default key table is used if NULL is specified for this parameter.

Input/Output

auth_context
Specifies the authentication context. A new authentication context is created and returned in this parameter if the value is NULL.

Output

ap_req_options
> Returns the options from the AP_REQ message. Specify NULL for this parameter if the options are not needed.

ticket
> Returns the ticket from the AP_REQ message. Specify NULL for this parameter if the ticket is not needed. The **krb5_free_ticket()** routine should be called to release the ticket when it is no longer needed.

Usage

The **krb5_rd_req()** routine processes an AP_REQ message generated by the partner application. The authenticator is extracted, validated, and stored in the authentication context. If the *server* parameter is not NULL and no replay cache is associated with the authentication context, the Kerberos runtime creates a replay cache and stores the cache handle in the authentication context.

If the authentication context contains a keyblock, it is used to decrypt the ticket in the AP_REQ message. This is useful for user-to-user authentication. If the authentication context does not contain a keyblock, the key table specified on the function call is used to obtain the decryption key.

The client in the authenticator must match the client in the ticket. If the remote address is set in the authentication context, the address list in the ticket must either include that address or must be a null list. If a replay cache handle is stored in the authentication context, the new authenticator is stored in the cache after checking for replay.

If no errors are detected, the authenticator, subsession key, and remote sequence number are stored in the authentication context. If AP_OPTS_MUTUAL_REQUIRED is specified in the AP_REQ message, the local sequence number is XORed with the remote sequence number.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

If the Kerberos security server is running on the same system as the application, it is not necessary to provide a key table. Instead, the **krb5_rd_req()** routine uses the local instance of the Kerberos security server to decrypt the ticket. In order to activate this support, the KRB5_SERVER_KEYTAB environment variable needs to be set to one of the following values and, depending on the value set, the following requirements must also be met:

1. If the KRB5_SERVER_KEYTAB environment variable is set to 1:

   a. NULL must be specified for the key table parameter on the call to the **krb5_rd_req()** routine.

   b. The application must be running with a user or group that has at least READ access to the IRR.RUSERMAP resource in the FACILITY class.

   c. The Kerberos principal associated with the current system identity must match the server principal in the ticket.

2. If the KRB5_SERVER_KEYTAB environment variable is set to 2:

   a. NULL must be specified for the key table parameter on the call to the **krb5_rd_req()** routine.

   b. The current system identity must have an associated Kerberos principal that matches the server principal in the ticket or have at least READ access in the KERBLINK class to the server principal in the ticket.

Note: If requirement 2a is satisfied but 2b is not, the **krb5_rd_req()** routine will not fall back to using a keytab file but will fail.

krb5_rd_req_verify (process a Kerberos AP_REQ message and verify checksum data)

Purpose

Processes a Kerberos AP_REQ message and verifies the application data checksum

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rd_req_verify (
    krb5_context                                    context,
    krb5_auth_context *                             auth_context,
    const krb5_data *                               in_data,
    const krb5_data *                               appl_data,
    krb5_const_principal                            server,
    krb5_keytab                                     keytab,
    krb5_flags *                                    ap_req_options,
    krb5_ticket **                                  ticket)
```

Parameters

Input

context
> Specifies the Kerberos context.

in_data
> Specifies the buffer containing the AP_REQ message.

appl_data
> Specifies the application data to be verified. The checksum is computed for the supplied data and compared to the checksum obtained from the authenticator. Specify NULL if the checksum is not to be verified.

server
> Specifies the server name. The server principal in the AP_REQ must be the same as the principal specified by this parameter. Specify NULL if any server principal is acceptable.

keytab
> Specifies the key table that contains the server key. The default key table is used if NULL is specified for this parameter.

Input/Output

auth_context
> Specifies the authentication context. A new authentication context is created and returned in this parameter if the value is NULL.

Output

ap_req_options
> Returns the options from the AP_REQ message. Specify NULL for this parameter if the options are not needed.

ticket
> Returns the ticket from the AP_REQ message. Specify NULL for this parameter if the ticket is not needed. The **krb5_free_ticket()** routine should be called to release the ticket when it is no longer needed.

Usage

The **krb5_rd_req_verify()** routine processes an AP_REQ message generated by the partner application and verifies the application data checksum contained in the authenticator. The authenticator is extracted, validated, and stored in the authentication context. If the server parameter is not NULL and no replay cache is associated with the authentication context, the Kerberos runtime creates a replay cache and stores the cache handle in the authentication context..

If the authentication context contains a keyblock, it is used to decrypt the ticket in the AP_REQ message. This is useful for user-to-user authentication. If the authentication context does not contain a keyblock, the key table specified on the function call is used to obtain the decryption key.

The client in the authenticator must match the client in the ticket. If the remote address has been set in the authentication context, the request must have come from that address. If a replay cache handle is stored in the authentication context, the new authenticator is stored in the cache after checking for replay.

If no errors are detected, the authenticator, subsession key, and remote sequence number are stored in the authentication context. If AP_OPTS_MUTUAL_REQUIRED is specified in the AP_REQ message, the local sequence number is XORed with the remote sequence number.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

If the Kerberos security server is running on the same system as the application, it is not necessary to provide a key table. Instead, the **krb5_rd_req_verify()** routine uses the local instance of the Kerberos security server to decrypt the ticket. In order to activate this support, the KRB5_SERVER_KEYTAB environment variable needs to be set to one of the following values and, depending on the value set, the following requirements must also be met: krb5_recvauth

1. If the KRB5_SERVER_KEYTAB environment variable is set to 1:

    a. NULL must be specified for the key table parameter on the call to the **krb5_rd_req_verify()** routine.

    b. The application must be running with a user or group that has at least READ access to the IRR.RUSERMAP resource in the FACILITY class.

    c. The Kerberos principal associated with the current system identity must match the server principal in the ticket.

2. If the KRB5_SERVER_KEYTAB environment variable is set to 2:

    a. NULL must be specified for the key table parameter on the call to the **krb5_rd_req_verify()** routine.

    b. The current system identity must have an associated Kerberos principal that matches the server principal in the ticket or have at least READ access in the KERBLINK class to the server principal in the ticket.

Note: If requirement 2a is satisfied but 2b is not, the **krb5_rd_req_verify()** routine will not fall back to using a keytab file but will fail.

krb5_rd_safe (process Kerberos KRB_SAFE message)

Purpose

Processes a Kerberos KRB_SAFE message.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_rd_safe (
    krb5_context                  context,
    krb5_auth_context             auth_context,
    const krb5_data *             in_data,
    krb5_data *                   out_data,
    krb5_replay_data *            replay_data)
```

Parameters

Input

context
    Specifies the Kerberos context.

in_data
>   Specifies the buffer containing the KRB_SAFE message

Input/Output

auth_context
>   Specifies the authentication context.

Output

out_data
>   Returns the application data supplied to the **krb5_mk_safe()** routine. The application should release
>   the data when it is no longer needed by calling the **krb5_free_data_contents()** routine.

replay_data
>   Returns replay information to the caller. This parameter is required if the
>   KRB5_AUTH_CONTEXT_RET_TIME or KRB5_AUTH_CONTEXT_RET_SEQUENCE flag is set in the
>   authentication context. Otherwise, NULL may be specified for this parameter.

Usage

The **krb5_rd_safe()** routine processes a KRB_SAFE message and extracts the application data after
verifying its integrity. If timestamps are being used, the message is stored in the replay cache associated
with the authentication context.

The keyblock used for verifying message integrity is obtained from the authentication context. The first
non-NULL keyblock is used by checking the local_subkey, remote_subkey, or keyblock, in that order.

The remote address in the authentication context must be present. It specifies the address of the sender.
The address type used for the **krb5_rd_safe()** routine must be the same as the address type used for the
krb5_mk_safe() routine. An error is returned if the address in the message does not match the remote
address in the authentication context.

The local address in the authentication context is optional. If it is present, then it must match the receiver
address in the message. Otherwise, the receiver address in the message must match one of the local
addresses returned by the **krb5_os_localaddr()** routine.

If message sequence numbers are being used (KRB5_AUTH_CONTEXT_DO_SEQUENCE is set in the
authentication context), the remote sequence number in the authentication context must match the
sequence number in the message.

If timestamps are being used (KRB5_AUTH_CONTEXT_DO_TIME is set in the authentication context), the
timestamp in the message must be within the Kerberos clock skew for the current time. In addition, the
message must not be found in the replay cache obtained from the authentication context.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_read_password (read a password)

Purpose

Reads a password from the terminal in non-display mode.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_read_password (
    krb5_context                     context,
    const char *                     prompt,
    const char *                     prompt2,
    char *                           return_pwd,
    int *                            size_return)
```

Parameters

Input

context
Specifies the Kerberos context.

prompt
Specifies the password prompt string. This string is displayed before reading the password from the terminal.

prompt2
Specifies the password verification string. This string is displayed before re-reading the password from the terminal. Specify NULL for this parameter if you do not want the password to be entered a second time for verification.

Input/Output

size_return
Specifies the size of the password buffer, including the string delimiter. The actual password length, excluding the string delimiter, is returned upon completion.

Output

return_pwd
Returns the password as a null-terminated string.

Usage

The **krb5_read_password()** routine reads a password from the terminal in non-display mode. The supplied buffer must be large enough to hold the password (any characters entered after the buffer size is reached are discarded). The *size_return* parameter must be set to the size of the password buffer before calling the **krb5_read_password()** routine. The actual password length is returned in the **size_return** parameter upon completion.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_realm_compare (compare two principal realms)

Purpose

Compares the realms of two principals.

Format

```
#include <skrb/krb5.h>
krb5_boolean krb5_realm_compare (
    krb5_context                        context,
    krb5_const_principal                princ1,
    krb5_const_principal                princ2)
```

Parameters

Input

context
Specifies the Kerberos context.

princ1
Specifies the first principal to be compared.

princ2
>Specifies the second principal to be compared.

Usage

The **krb5_realm_compare()** routine compares the realms for two principals. The function return value will be TRUE if the realms are the same and FALSE if they are not the same.

krb5_recvauth (receive authentication message)

Purpose

Receives an authentication message sent by the **krb5_sendauth()** routine

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_recvauth (
    krb5_context                    context,
    krb5_auth_context *             auth_context,
    krb5_pointer                    socket,
    char *                          appl_version,
    krb5_principal                  server,
    krb5_int32                      flags,
    krb5_keytab                     keytab,
    krb5_ticket **                  ticket)
```

Parameters

Input

context
>Specifies the Kerberos context.

socket
>Specifies the address of a socket descriptor. This descriptor must represent a TCP stream connection and not a UDP datagram connection.

appl_version
>Specifies the application version message. An error is returned if this application version message does not match the application version message supplied by the sender. Specify NULL for this parameter if the application version message does not need to be verified. The supplied application version message is converted to the network code page (ISO 8859-1) before comparing it with the sender's application version message.

server
>Specifies the server name. The server principal in the AP_REQ must be the same as the principal specified by this parameter. Specify NULL if any server principal is acceptable.

flags
>Specifies flags for the **krb5_recvauth()** routine. There are currently no defined flags.

keytab
>Specifies the key table that contains the server key. The default key table is used if NULL is specified for this parameter.

Input/Output

auth_context
>Specifies the authentication context. A new authentication context is created and returned in this parameter if the value is NULL.

Output

ticket
> Returns the service ticket in the AP_REQ message. Specify NULL for this parameter if the ticket is not needed. The **krb5_free_ticket()** routine should be called to release the ticket when it is no longer needed.

Usage

The **krb5_recvauth()** routine processes an authentication message stream generated by the **krb5_sendauth()** routine. It receives the authentication message and sends the authentication response using the socket descriptor supplied by the application. The application is responsible for establishing the connection before calling the **krb5_recvauth()** routine.

The **krb5_recvauth()** routine processes an AP_REQ message generated by the partner application. The authenticator is extracted, validated, and stored in the authentication context. If the *server* parameter is not NULL and no replay cache is associated with the authentication context, the Kerberos runtime creates a replay cache and stores the cache handle in the authentication context.

If the authentication context contains a keyblock, it is used to decrypt the ticket in the AP_REQ message. This is useful for user-to-user authentication. If the authentication context does not contain a keyblock, the key table specified on the function call is used to obtain the decryption key.

The client in the authenticator must match the client in the ticket. If the remote address is set in the authentication context, the address list in the ticket must either include that address or must be a null list. If a replay cache handle is stored in the authentication context, the new authenticator is stored in the cache after checking for replay.

If no errors are detected, the authenticator, subsession key, and remote sequence number are stored in the authentication context. If AP_OPTS_MUTUAL_REQUIRED is specified in the AP_REQ message, the local sequence number is XORed with the remote sequence number.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

If the Kerberos security server is running on the same system as the application, it is not necessary to provide a key table. Instead, the **krb5_recvauth()** routine uses the local instance of the Kerberos security server to decrypt the ticket. In order to activate this support, the KRB5_SERVER_KEYTAB environment variable needs to be set to one of the following values and, depending on the value set, the following requirements must also be met:

1. If the KRB5_SERVER_KEYTAB environment variable is set to 1:

   a. NULL must be specified for the key table parameter on the call to the **krb5_recvauth()** routine.

   b. The application must be running with a user or group that has at least READ access to the IRR.RUSERMAP resource in the FACILITY class.

   c. The Kerberos principal associated with the current system identity must match the server principal in the ticket.

2. If the KRB5_SERVER_KEYTAB environment variable is set to 2:

   a. NULL must be specified for the key table parameter on the call to the **krb5_recvauth()** routine.

   b. The current system identity must have an associated Kerberos principal that matches the server principal in the ticket or have at least READ access in the KERBLINK class to the server principal in the ticket.

Note: If requirement 2a is satisfied but 2b is not, the **krb5_recvauth()** routine will not fall back to using a keytab file but will fail.

krb5_sendauth (send authentication message)

Purpose

Sends an authentication message for processing by the **krb5_recvauth()** routine.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_sendauth (
    krb5_context                            context,
    krb5_auth_context *                     auth_context,
    krb5_pointer                            socket,
    char *                                  appl_version,
    krb5_principal                          client,
    krb5_principal                          server,
    krb5_int32                              ap_req_options,
    krb5_data *                             appl_data,
    krb5_creds *                            in_creds,
    krb5_ccache                             ccache,
    krb5_error **                           error,
    krb5_ap_rep_enc_part **                 rep_result,
    krb5_creds **                           out_creds)
```

Parameters

Input

context
Specifies the Kerberos context.

socket
Specifies the address of a socket descriptor. This descriptor must represent a TCP stream connection and not a UDP datagram connection.

appl_version
Specifies the application version message. An error is returned if this application version message does not match the application version message supplied by the receiver. The supplied application version message is converted to the network code page (ISO 8859-1) before being sent to the partner application.

client
Specifies the client name. This parameter is ignored if a non-NULL value is supplied for the *in_creds* parameter. The client name is obtained from the credentials cache if this parameter is NULL.

server
Specifies the server name. This parameter is ignored if a non-NULL value is provided for the *in_creds* parameter.

ap_req_options
Specifies request options as follows:

- AP_OPTS_USE_SESSION_KEY - Use session key instead of server key for the service ticket. The credentials must include a ticket that is encrypted in the session key.

- AP_OPTS_MUTUAL_REQUIRED - Mutual authentication required.

- AP_OPTS_USE_SUBKEY - Generate a subsession key from the current session key obtained from the credentials.

appl_data
Specifies the application data whose checksum is to be included in the authenticator. Specify NULL for this parameter if no checksum is to be included in the authenticator.

in_creds
Specifies the credentials for the specified service. The *client* and *server* parameters are ignored if a non-NULL value is provided for the *in_creds* parameter. In this case, the client and server names must be set in the input credentials. The service ticket may be supplied as part of the input credentials by setting a non-zero ticket length value. If the service ticket is not supplied as part of the input

credentials, the Kerberos runtime obtains a service ticket using the ticket-granting ticket retrieved from the credentials cache.

When the Kerberos runtime obtains the service ticket, additional fields are checked in the input credentials. The *second_ticket* field must be set if the service ticket is to be encrypted in a session key. The ticket expiration time can be set to override the default expiration time. The key encryption type can be set to override the default ticket encryption type.

ccache
> Specifies the credentials cache used to obtain credentials to the desired service. The credentials cache is not used when the service ticket is supplied as part of the input credentials. The default credentials cache is used if this parameter is NULL.

Input/Output

auth_context
> Specifies the authentication context. A new authentication context is created and returned in this parameter if the value is NULL.

Output

error
> Returns the KRB_ERROR message if an authentication error is reported by the partner application. The **krb5_free_error()** routine should be called to release the error message when it is no longer needed. Specify NULL for this parameter if the error message is not needed.

rep_result
> Returns the decrypted reply data from the AP_REP message. The **krb5_free_ap_rep_enc_part()** routine should be called to release the reply data when it is no longer needed. Specify NULL for this parameter if the reply data is not needed. A reply is available only if AP_OPTS_MUTUAL_REQUIRED is specified in the request options.

out_creds
> Returns the service ticket. The **krb5_free_creds()** routine should be called to release the credentials when they are no longer needed. Specify NULL for this parameter if the service ticket is not needed.

Usage

The **krb5_sendauth()** routine generates an authentication message stream for processing by the **krb5_recvauth()** routine. It sends the authentication message and receives the authentication response using the socket descriptor supplied by the application. The application is responsible for establishing the connection before calling the **krb5_sendauth()** routine.

The **krb5_sendauth()** routine generates an AP_REQ message. The checksum of the application data is included in the authenticator that is part of the AP_REQ message. This message is then sent to the partner application, which calls the **krb5_recvauth()** routine to validate the authenticity of the message. The checksum method set in the authentication context is used to generate the checksum.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_set_config_files (set Kerberos configuration files for processing)

Purpose

Sets the files to be processed for Kerberos configuration requests.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_set_config_files (
```

```
    krb5_context                        context,
    const char **                       names)
```

Parameters

Input

context
Specifies the Kerberos context.

names
Specifies an array of file names. The last entry in the array must be a NULL pointer.

Usage

The **krb5_set_config_files()** specifies the names of the files to be processed to obtain the Kerberos configuration. This replaces the configuration files that were used to create the Kerberos context. Changing the configuration files does not affect context values that have already been set from the old configuration files.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_set_default_in_tkt_ktypes (set default encryption types)

Purpose

Sets the default encryption types used when requesting an initial ticket from the KDC.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_set_default_in_tkt_ktypes (
    krb5_context                        context,
    const krb5_enctype *                ktypes)
```

Parameters

Input

context
Specifies the Kerberos context.

ktypes
Specifies an array of krb5_enctype values which will be used when requesting a service ticket. The last element in the array must be set to ENCTYPE_NULL. The following symbolic definitions are provided for specifying the encryption types:

- ENCTYPE_DES_CBC_CRC - DES encryption with a CRC checksum. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD4 - DES encryption with an MD4 checksum. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD5 - DES encryption with an MD5 checksum. (not valid in FIPS mode)
- ENCTYPE_DES_HMAC_SHA1 - DES encryption with SHA1 checksum. (not valid in FIPS mode)
- ENCTYPE_DES3_CBC_SHA1 - DES3 encryption with SHA1 checksum.
- ENCTYPE_AES128_CTS_HMAC_SHA1_96 - AES128 encryption with SHA1 checksum.
- ENCTYPE_AES256_CTS_HMAC_SHA1_96 - AES256 encryption with SHA1 checksum.
- ENCTYPE_AES128_CTS_HMAC_SHA256_128 – AES 128 encryption with SHA2 checksum
- ENCTYPE_AES256_CTS_HMAC_SHA384_192 – AES 256 encryption with SHA2 checksum

Usage

The **krb5_set_default_in_tkt_ktypes()** routine sets the default encryption types used when requesting the initial ticket from the KDC. In order to interoperate with older Kerberos V5 servers, you should include ENCTYPE_DES_CBC_CRC as one of the encryption types.

The encryption types specified override any values specified by the *default_tkt_enctypes* entry in the Kerberos configuration file.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_set_default_realm (set default realm)

Purpose

Sets the default realm for the local system.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_set_default_realm (
    krb5_context                        context,
    const char *                        realm)
```

Parameters

Input

context
Specifies the Kerberos context.

realm
Specifies the name for the default realm.

Usage

The **krb5_set_default_realm()** routine sets the default realm for the specified Kerberos context. This overrides the default realm set by the Kerberos configuration file. The realm set by **krb5_set_default_realm()** applies only to the Kerberos context specified by the context parameter.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_set_default_tgs_ktypes (set default encryption types)

Purpose

Sets the default encryption types used when requesting a service ticket from the KDC.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_set_default_tgs_ktypes (
    krb5_context                        context,
    const krb5_enctype *                ktypes)
```

Parameters

Input

context
Specifies the Kerberos context.

ktypes
Specifies an array of krb5_enctype values which will be used when requesting a service ticket. The last element in the array must be set to ENCTYPE_NULL. The following symbolic definitions are provided for specifying the encryption types:

- ENCTYPE_DES_CBC_CRC - DES encryption with a CRC checksum. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD4 - DES encryption with an MD4 checksum. (not valid in FIPS mode)
- ENCTYPE_DES_CBC_MD5 - DES encryption with an MD5 checksum. (not valid in FIPS mode)
- ENCTYPE_DES_HMAC_SHA1 - DES encryption with SHA1 checksum. (not valid in FIPS mode)
- ENCTYPE_DES3_CBC_SHA1 - DES3 encryption with SHA1 checksum.
- ENCTYPE_AES128_CTS_HMAC_SHA1_96 - AES128 encryption with SHA1 checksum.
- ENCTYPE_AES256_CTS_HMAC_SHA1_96 - AES256 encryption with SHA1 checksum.
- ENCTYPE_AES128_CTS_HMAC_SHA256_128 – AES 128 encryption with SHA2 checksum
- ENCTYPE_AES256_CTS_HMAC_SHA384_192 – AES 256 encryption with SHA2 checksum

Usage

The **krb5_set_default_tgs_ktypes()** routine sets the default encryption types used when requesting a service ticket from the KDC. In order to interoperate with older Kerberos V5 servers, you should include ENCTYPE_DES_CBC_CRC as one of the encryption types.

The encryption types specified overrides any values specified by the *default_tgs_enctypes* entry in the Kerberos configuration file.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_set_fast_armor_ticket (set the armor ticket for use in FAST pre-authentication)

Purpose

Adds the input armor ticket to the Kerberos context in preparation for requesting an initial ticket (TGT) using FAST pre-authentication.

Format

```
#include <skrb/krb5.conf>
krb5_error_code krb5_set_fast_armor_ticket (
    krb5_context      context,
    krb5_creds *      armor_tkt)
```

Parameters

Input

context
Specifies the Kerberos context.

armor_tkt

Specifies the ticket-granting ticket to be used as a FAST armor ticket. The armor ticket must be an anonymous PKINIT ticket, have a valid session key encryption type, and have at least 10 minutes remaining before it expires. The armor_tkt will be freed by a call to **krb5_free_context()**.

Usage

The **krb5_set_fast_armor_ticket()** establishes the armor ticket to be used in subsequent ticket-granting ticket (TGT) requests that use FAST pre-authentication. The resources associated with the armor_tkt will be freed when **krb5_set_fast_armor_ticket()** is called.

krb5_set_value_pkinit (set pkinit value)

Purpose

Add to a Kerberos context for public private key authentication from values specified in the input attribute / value pair parameter.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_set_value_pkinit (
    krb5_context                    context,
    char*                           attr,
    char*                           value)
```

Parameters

Input/Output

context

Input is the context obtained from krb5_init_context or krb5_init_context_pkinit. Output is an updated context with pkinit values obtained from the input attribute / value pair parameter.

Input

attr

Specifies the pkinit attribute name.

value

Specifies the value of the pkinit attribute.

Usage

The krb5_set_value_pkinit() routine adds to the Kerberos context with pkinit value specified in the attribute / value pair parameter. It can be called multiple times to add multiple attribute / value pairs. The context must be obtained from krb5_init_context or krb5_init_context_pkinit before this call. If the input context is from krb5_init_context_pkinit, the values specified in this call will override them.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_sname_to_principal (convert service name to Kerberos principal)

Purpose

Converts a service name to a Kerberos principal.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_sname_to_principal (
    krb5_context                    context,
    const char *                    hostname,
    const char *                    sname,
    krb5_int32                      type,
    krb5_principal *                ret_princ)
```

Parameters

Input

context
    Specifies the Kerberos context.

hostname
    Specifies the host containing the desired service instance. The local host is used if NULL is specified for this parameter.

sname
    Specifies the service name. The service name is set to *host* if NULL is specified for this parameter.

type
    Specifies the type of host name provided as follows:

- KRB5_NT_SRV_HST - A DNS host name has been provided. The Kerberos runtime calls the **getaddrinfo()** system function to obtain the canonical name for the host. The resulting host name is then converted to lowercase.

- KRB5_NT_UNKNOWN - The host name type is unknown. No translation is performed on the specified host name and is used as-is.

Output

ret_princ
    Returns the generated principal. The **krb5_free_principal()** routine should be called to release the principal when it is no longer needed.

Usage

The **krb5_sname_to_principal()** routine generates a Kerberos principal from a service name and a host name. The principal name is in the format *sname/hostname@realm*. The realm name that corresponds to the host name is obtained by calling the **krb5_get_host_realm()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_svc_get_msg (return text message from Kerberos error code)

Purpose

Returns a printable text message corresponding to a Kerberos error code.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_svc_get_msg (
    krb5_ui_4                       error_code,
    char **                         msg_text)
```

Parameters

Input

error_code
Specifies the Kerberos error code

Output

msg_text
Returns the character string describing the error code. The caller should free the character string returned by this parameter when it is no longer needed by calling the **krb5_free_string()** routine.

Usage

The **krb5_svc_get_msg()** routine returns a printable character string that describes the error represented by the supplied error code. This allows the application to log the error or display it to the user.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_timeofday (return current time of day)

Purpose

Returns the current time of day in seconds since the epoch.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_timeofday (
    krb5_context                     context,
    krb5_timestamp *                 seconds)
```

Parameters

Input

context
Specifies the Kerberos context.

Output

seconds
Returns the number of seconds since the epoch.

Usage

The **krb5_timeofday()** routine returns the number of seconds since the epoch (January 1, 1970). The returned time is not adjusted for local time differences.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_timeofday64 (return current time of day)

Purpose

Returns the current time of day in seconds since the epoch.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_timeofday64 (
    krb5_context                         context,
    krb5_timestamp64 *                   seconds)
```

Parameters

Input

context
    Specifies the Kerberos context.

Output

seconds
    Returns the number of seconds since the epoch.

Usage

The **krb5_timeofday64()** routine returns the number of seconds since the epoch (January 1, 1970) as a 64 bit value. The returned time is not adjusted for local time differences.

Note: In order to use this function in a 31 bit application, the application must use the LANGLVL(LONGLONG) or LANGLVL(EXTENDED) compiler option and define the _LARGE_TIME_API feature test macro prior to including krb5.h or krbload.h.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_unparse_name (convert Keberos principal to text string)

Purpose

Converts a Kerberos principal to a text string.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_unparse_name (
    krb5_context                         context,
    krb5_const_principal                 principal,
    char **                              name)
```

Parameters

Input

context
    Specifies the Kerberos context.
principal
    Specifies the principal to be converted.

Output

name
    Returns the text string for the principal in the format *name@realm*. The application should free the text string when it is no longer needed.

Usage

The **krb5_unparse_name()** routine creates a text string from a Kerberos principal. The string is in the format *name@realm* with the name components separated by forward slashes. If a forward slash occurs within a name component, it is escaped in the generated string by preceding the forward slash with a backward slash.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_unparse_name_ext (convert Kerberos principal to text string)

Purpose

Converts a Kerberos principal to a text string.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_unparse_name_ext (
    krb5_context                            context,
    krb5_const_principal                    principal,
    char **                                 name,
    int *                                   size)
```

Parameters

Input

context
    Specifies the Kerberos context.

principal
    Specifies the principal to be converted.

Input/Output

name
    Returns the text string for the principal in the format *name@realm*. The application should free the text string when it is no longer needed. If the *name* parameter contains a NULL address upon entry, **krb5_unparse_name_ext()** allocates a new buffer and returns the address in the *name* parameter and the size in the *size* parameter. Otherwise, the *name* parameter must contain the address of an existing buffer and the *size* parameter must contain the size of this buffer. The **krb5_unparse_name_ext()** reallocates the buffer if necessary and returns the updated values in the *name* and *size* parameters.

size
    The size of the buffer specified by the *name* parameter.

Usage

The **krb5_unparse_name_ext()** routine creates a text string from a Kerberos principal. The string is in the format *name@realm* with the name components separated by forward slashes. If a forward slash occurs within a name component, it is escaped in the generated string by preceding the forward slash with a backward slash.

The k**rb5_unparse_name_ext()** routine is similar to the **krb5_unparse_name()** routine, but it allows the application to avoid the overhead of repeatedly allocating the output string when a large number of conversions need to be performed.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_us_timeofday (return current time of day)

Purpose

Returns the current time of day in seconds and microseconds since the epoch.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_us_timeofday (
    krb5_context                    context,
    krb5_timestamp *                seconds,
    krb5_int32 *                    useconds)
```

Parameters

Input

context
    Specifies the Kerberos context.

Output

seconds
    Returns the seconds portion of the result.

useconds
    Returns the microseconds portion of the result.

Usage

The **krb5_us_timeofday()** routine returns the number of seconds and microseconds since the epoch (January 1, 1970). The returned time is not adjusted for local time differences.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

krb5_us_timeofday64 (return current time of day)

Purpose

Returns the current time of day in seconds and microseconds since the epoch.

Format

```
#include <skrb/krb5.h>
krb5_error_code krb5_us_timeofday64 (
    krb5_context                    context,
    krb5_timestamp64 *              seconds,
    krb5_int32 *                    useconds)
```

Parameters

Input

context
    Specifies the Kerberos context.

Output

seconds
>   Returns the seconds portion of the result.

useconds
>   Returns the microseconds portion of the result.

Usage

The **krb5_us_timeofday64()** routine returns the number of seconds and microseconds since the epoch (January 1, 1970). The seconds value is a 64 bit value.

Note: In order to use this function in a 31 bit application, the application must use the LANGLVL(LONGLONG) or LANGLVL(EXTENDED) compiler option and define the _LARGE_TIME_API feature test macro prior to including krb5.h or krbload.h.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code.

Chapter 3. Kerberos administration programming interfaces

kadm5_chpass_principal (change the password for a principal entry)

Purpose

Changes the password for a principal entry in the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_chpass_principal (
    void *                              server_handle,
    krb5_principal                      principal,
    char *                              passwd)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

principal
Specifies the principal whose password is to be changed.

passwd
Specifies the new password for the principal.

Usage

The **kadm5_chpass_principal()** routine changes the password for a principal entry in the Kerberos database. You must have CHANGEPW authority, the requested principal entry must be your own entry, or the administration session must be with the **kadmin/changepw** service

The **kadm5_chpass_principal()** routine generates an encryption key for each encryption type supported by the Kerberos administration server. Use the **kadm5_chpass_principal_3()** routine if you want to generate encryption keys for a subset of the available encryption types or if you want to retain the existing encryption keys.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_chpass_principal()** routine:

Table 2. Common errors returned by the **kadm5_chpass_principal()** routine	
Function	**Error**
KADM5_AUTH_CHANGEPW	Not authorized to change the password for the entry
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified
KADM5_BAD_LENGTH	Password length is not valid
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_GSS_ERROR	GSS-API error

*Table 2. Common errors returned by the **kadm5_chpass_principal()** routine (continued)*

Function	Error
KADM5_PASS_Q_CLASS	Specified password does not contain the minimum number of character classes
KADM5_PASS_Q_DICT	Specified password does not pass the dictionary test
KADM5_PASS_Q_TOOSHORT	Specified password is too short
KADM5_PASS_REJECTED	Password rejected by system policy
KADM5_PASS_REUSE	Password has already been used
KADM5_PROTECT_PRINCIPAL	Protected principal cannot be modified
KADM5_RPC_ERROR	Communication error
KADM5_UNK_PRINC	Unknown principal

kadm5_chpass_principal_3 (change the password for a principal entry)

Purpose

Changes the password for a principal entry in the Kerberos database.

Format

```
#include <skrb/admin.h>

kadm5_ret_t kadm5_chpass_principal_3 (
    void *                              server_handle,
    krb5_principal                      principal,
    krb5_boolean                        keepold,
    int                                 n_ks_entries,
    krb5_key_salt_tuple *               ks_entries,
    char *                              passwd)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

principal
Specifies the principal whose password is to be changed.

keepold
Specifies whether to keep the old key entries. The number of retained keys is dependent upon the Kerberos database implementation.

n_ks_entries
Specifies the number of key-salt entries.

ks_entries
Specifies an array of key-salt entries.

passwd
Specifies the new password for the principal.

Usage

The **kadm5_chpass_principal_3()** routine changes the password for a principal entry in the Kerberos database. You must have CHANGEPW authority, the requested principal entry must be your own entry, or the administration session must be with the **kadmin/changepw** service

The **kadm5_chpass_principal_3()** routine allows the specification of the encryption types used to generate encryption keys from the supplied password. It is the same as the **kadm5_chpass_principal()** routine if no key-salt entries are provided. An error is returned if an unsupported encryption type or salt type is specified.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_chpass_principal_3()** routine:

*Table 3. Common errors returned by the **kadm5_chpass_principal_3()** routine*

Function	Error
KADM5_AUTH_CHANGEPW	Not authorized to change the password for the entry.
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified.
KADM5_BAD_ENCTYPE	Encryption type is not valid.
KADM5_BAD_LENGTH	Password length is not valid.
KADM5_BAD_SALTTYPE	Salt type is not valid.
KADM5_BAD_SERVER_HANDLE	Server handle is not valid.
KADM5_GSS_ERROR	GSS-API error.
KADM5_PASS_Q_CLASS	Specified password does not contain the minimum number of character classes.
KADM5_PASS_Q_DICT	Specified password does not pass the dictionary test.
KADM5_PASS_Q_TOOSHORT	Specified password is too short.
KADM5_PASS_REJECTED	Password rejected by system policy.
KADM5_PASS_REUSE	Password has already been used.
KADM5_PROTECT_PRINCIPAL	Protected principal cannot be modified.
KADM5_RPC_ERROR	Communication error.
KADM5_UNK_PRINC	Unknown principal.

kadm5_create_policy (create a policy entry)

Purpose

Creates a policy entry in the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_create_policy (
    void *                          server_handle,
    kadm5_policy_ent_t              entry,
    krb5_flags                      mask)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

entry
Specifies the information for the policy entry.

mask
Specifies the fields in the **krb5_policy_ent_t** that are to be used to create the policy entry. The following flags can be ORed together to define the mask:

- KADM5_POLICY - the policy name is set (this flag must be set when creating a policy entry)
- KADM5_PW_HISTORY_NUM - the password history count is set
- KADM5_PW_MIN_CLASSES - the minimum number of password character classes is set.
- KADM5_PW_MIN_LENGTH - the minimum password length is set.
- KADM5_PW_MIN_LIFE - the minimum password lifetime is set.
- KADM5_PW_MAX_LIFE - the maximum password lifetime is set.

Usage

The **kadm5_create_policy()** routine creates a policy entry in the Kerberos database. You must have ADD authority.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_create_policy()** routine:

*Table 4. Common errors returned by the **kadm5_create_policy()** routine*

Function	Error
KADM5_AUTH_ADD	Not authorized to add an entry
KADM5_BAD_CLASS	Character class count is not valid
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified
KADM5_BAD_HISTORY	Password history count is not valid
KADM5_BAD_LENGTH	Minimum password length is not valid
KADM5_BAD_MASK	Incorrect policy creation mask specified
KADM5_BAD_MIN_PASS_LIFE	Minimum password lifetime is not valid
KADM5_BAD_POLICY	Policy name is not valid
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_DUP	Policy already exists
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error

kadm5_create_principal (create a principal entry)

Purpose

Creates a principal entry in the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_create_principal (
    void *                                  server_handle,
    kadm5_principal_ent_t                   entry,
    krb5_flags                              mask,
    char *                                  passwd)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

entry
Specifies the information for the principal entry.

mask
Specifies the fields in the **krb5_principal_ent_t** that are used to create the principal entry. The following flags can be ORed together to define the mask:

- KADM5_ATTRIBUTES - the principal attributes are set.
- KADM5_KVNO - the key version number is set.
- KADM5_MAX_LIFE - the maximum ticket lifetime is set.
- KADM5_MAX_RLIFE - the maximum renewable lifetime is set.
- KADM5_POLICY - the policy name is set.
- KADM5_PRINCIPAL - the principal name is set (this flag must be set when creating a principal entry)
- KADM5_PRINC_EXPIRE_TIME - the account expiration time is set.
- KADM5_PW_EXPIRATION - the password expiration time is set.
- KADM5_TL_DATA - the tagged data is set.

passwd
Specifies the password for the principal.

Usage

The **kadm5_create_principal()** routine creates a principal entry in the Kerberos database. For KADM5_TL_DATA, the ability to store tagged data is dependent upon the database implementation. You must have ADD authority.

The **kadm5_create_principal()** routine generates an encryption key for each encryption type supported by the Kerberos administration server. Use the **kadm5_create_principal_3()** routine if you want to generate encryption keys for a subset of the available encryption types.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_create_principal()** routine:

*Table 5. Common errors returned by the **kadm5_create_principal()** routine*

Function	Error
KADM5_AUTH_ADD	Not authorized to add an entry
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_BAD_MASK	Incorrect principal creation mask specified
KADM5_DUP	Principal already exists

*Table 5. Common errors returned by the **kadm5_create_principal()** routine (continued)*

Function	Error
KADM5_GSS_ERROR	GSS-API error
KADM5_PASS_Q_CLASS	Password does not contain the minimum number of character classes
KADM5_PASS_Q_DICT	Password does not pass the dictionary test
KADM5_PASS_Q_TOOSHORT	Password is too short
KADM5_PASS_REJECTED	Password rejected by system policy
KADM5_RPC_ERROR	Communication error
KADM5_UNK_POLICY	Policy does not exist

kadm5_create_principal_3 (create a principal entry)

Purpose

Creates a principal entry in the Kerberos database.

Format

```
#include <skrb/admin.h>

kadm5_ret_t kadm5_create_principal_3 (
    void *                    server_handle,
    kadm5_principal_ent_t     entry,
    krb5_flags                mask,
    int                       n_ks_entries,
    krb5_key_salt_tuple *     ks_entries,
    char *                    passwd)
```

Parameters

Input

server_handle
    Specifies the server handle for the session with the administration server.

entry
    Specifies the information for the principal entry.

mask
    Specifies the fields in the **krb5_principal_ent_t** that are used to create the principal entry. The following flags can be ORed together to define the mask:

- KADM5_ATTRIBUTES - the principal attributes are set.
- KADM5_KVNO - the key version number is set.
- KADM5_MAX_LIFE - the maximum ticket lifetime is set.
- KADM5_MAX_RLIFE - the maximum renewable lifetime is set.
- KADM5_POLICY - the policy name is set.
- KADM5_PRINCIPAL - the principal name is set (this flag must be set when creating a principal entry)
- KADM5_PRINC_EXPIRE_TIME - the account expiration time is set.
- KADM5_PW_EXPIRATION - the password expiration time is set.
- KADM5_TL_DATA - the tagged data is set.

n_ks_entries
Specifies the number of key-salt entries.

ks_entries
Specifies an array of key-salt entries.

passwd
Specifies the password for the principal.

Usage

The **kadm5_create_principal_3()** routine creates a principal entry in the Kerberos database. For KADM5_TL_DATA, the ability to store tagged data is dependent upon the database implementation. You must have ADD authority.

The **kadm5_create_principal_3()** routine allows the specification of the encryption types used to generate encryption keys from the supplied password. It is the same as the **kadm5_create_principal()** routine if no key-salt entries are provided. An error is returned if an unsupported encryption type or salt type is specified.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_create_principal()** routine:

Table 6. Common errors returned by the **kadm5_create_principal()** *routine*

Function	Error
KADM5_AUTH_ADD	Not authorized to add an entry.
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified.
KADM5_BAD_ENCTYPE	Encryption type is not supported.
KADM5_BAD_SALTTYPE	Salt type is not supported.
KADM5_BAD_SERVER_HANDLE	Server handle is not valid.
KADM5_BAD_MASK	Incorrect principal creation mask specified.
KADM5_DUP	Principal already exists.
KADM5_GSS_ERROR	GSS-API error.
KADM5_PASS_Q_CLASS	Password does not contain the minimum number of character classes.
KADM5_PASS_Q_DICT	Password does not pass the dictionary test.
KADM5_PASS_Q_TOOSHORT	Password is too short.
KADM5_PASS_REJECTED	Password rejected by system policy.
KADM5_RPC_ERROR	Communication error.
KADM5_UNK_POLICY	Policy does not exist.

kadm5_delete_policy (delete a principal entry)

Purpose

Deletes a policy entry from the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_delete_policy (
```

```
    void *                          server_handle,
    char *                          policy)
```

Parameters

Input

server_handle
    Specifies the server handle for the session with the administration server.

policy
    Specifies the policy entry to be deleted.

Usage

The **kadm5_delete_policy()** routine deletes a policy entry from the Kerberos database. You must have DELETE authority.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_delete_policy()** routine:

Function	Error
*Table 7. Common errors returned by the **kadm5_delete_policy()** routine*	
KADM5_AUTH_DELETE	Not authorized to delete an entry
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified
KADM5_BAD_POLICY	Policy name is not valid
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_GSS_ERROR	GSS-API error
KADM5_POLICY_REF	Policy still refered to by one or more principal entries
KADM5_RPC_ERROR	Communication error
KADM5_UNK_PRINC	Unknown principal

kadm5_delete_principal (delete a principal entry)

Purpose

Deletes a principal entry from the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_delete_principal (
    void *                          server_handle,
    krb5_principal                  principal)
```

Parameters

Input

server_handle
    Specifies the server handle for the session with the administration server.

principal
    Specifies the principal entry to be deleted.

Usage

The **kadm5_delete_principal()** routine deletes a principal entry from the Kerberos database. You must have DELETE authority.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_delete_principal()** routine:

*Table 8. Common errors returned by the **kadm5_delete_principal()** routine*

Function	Error
KADM5_AUTH_DELETE	Not authorized to delete an entry
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error
KADM5_UNK_PRINC	Unknown principal

kadm5_destroy (close a session)

Purpose

Closes a session with the Kerberos administration server.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_destroy (
    void *                              server_handle)
```

Parameters

Input

server_handle
    Specifies the server handle for the session with the administration server.

Usage

The **kadm5_destroy()** routine closes a session established by the **kadm5_init_with_creds()**, **kadm5_init_with_password()**, or **kadm5_init_with_skey()** routine. The server handle is no longer valid upon completion of the **kadm5_destroy()** routine.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_destroy()** routine:

*Table 9. Common errors returned by the **kadm5_destroy()** routine*

Function	Error
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error

kadm5_free_key_list (free a list of keys)

Purpose

Frees a list of keys.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_free_key_list (
    void *                          server_handle,
    krb5_keyblock *                 keys,
    int                             count)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

keys
Specifies an array of keyblocks.

count
Specifies the number of entries in the array.

Usage

The **kadm5_free_key_list()** routine releases the storage allocated for an array of Kerberos keys.

The function return value is always zero.

kadm5_free_name_list (free a list of names)

Purpose

Frees a list of names.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_free_name_list (
    void *                          server_handle,
    char **                         names,
    int                             count)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

names
Specifies the list of names.

count
Specifies the number of entries in the list.

Usage

The **kadm5_free_name_list()** routine releases the storage allocated for a list of names.

The function return value is always zero.

kadm5_free_policy_ent (release policy entry storage)

Purpose

Releases storage allocated for a policy entry.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_free_policy_ent (
    void *                          server_handle,
    kadm5_policy_ent_t              entry)
```

Parameters

Input

server_handle
    Specifies the server handle for the session with the administration server.

Input/Output

entry
    Specifies the policy entry to be released.

Usage

The **kadm5_free_policy_ent()** routine releases storage allocated for a policy entry.

The function return value is always zero.

kadm5_free_principal_ent (release principal entry storage)

Purpose

Releases storage allocated for a principal entry.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_free_principal_ent (
    void *                          server_handle,
    kadm5_principal_ent_t     entry)
```

Parameters

Input

server_handle
    Specifies the server handle for the session with the administration server.

Input/Output

entry
Specifies the principal entry to be released.

Usage

The **kadm5_free_principal_ent()** routine releases storage allocated for a principal entry.

The function return value is always zero.

kadm5_get_policies (return a list of policies)

Purpose

Returns a list of policies matching the specified search expression.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_get_policies (
    void *                          server_handle,
    char *                          expression,
    char ***                        policies,
    int *                           count)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

expression
Specifies the search expression. The maximum string length is 1024 bytes. All policies are listed if NULL is specified for this parameter.

Output

policies
Returns the list of policy names matching the search expression. The list should be released when it is no longer needed by calling the **kadm5_free_name_list()** routine.

count
Returns the number of entries in the list.

Usage

The **kadm5_get_policies()** routine returns a list of policy names matching a search expression. You must have LIST authority.

The search expression can include the "*" and "?" wildcards, where "*" represents zero or more characters, and "?" represents a single character. For example, the expression "*_local" returns all policy names that end with "_local," the expression "def*" returns all default names that begin with "def," and the expression "test_policy?" returns policy names such as **test_policy1**, **test_policy2**, and so forth. You can use "*" and "\?" to search for a "*" or "?" character instead of treating the characters as wildcards.

The search string can also contain paired "[" and "]" characters with one or more characters between the brackets. A match occurs if a name contains one of the characters between the brackets. For example, the expression "[adh]*" returns all names beginning with "a," "d," or "h." You can use "\[" and "\]" to search for a "[" or "]" character.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_get_policies()** routine:

*Table 10. Common errors returned by the **kadm5_get_policies()** routine*

Function	Error
KADM5_AUTH_LIST	Not authorized to list entries
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error

kadm5_get_policy (return policy entry information)

Purpose

Return information from a policy entry in the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_get_policy (
    void *                                  server_handle,
    char *                                  name,
    kadm5_policy_ent_t                      entry)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

name
Specifies the policy entry to be returned.

Output

entry
Returns the requested information. The storage allocated for the policy entry should be released when it is no longer needed by calling the **kadm5_free_policy_ent()** routine.

Usage

The **kadm5_get_policy()** routine returns information from a policy entry in the Kerberos database. Some of the fields may not be available depending upon the Kerberos database implementation. You must have GET authority or the requested policy must be the policy associated with your principal.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_get_policy()** routine:

*Table 11. Common errors returned by the **kadm5_get_policy()** routine*

Function	Error
KADM5_AUTH_GET	Not authorized to get entry
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified

*Table 11. Common errors returned by the **kadm5_get_policy()** routine (continued)*

Function	Error
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error
KADM5_UNK_POLICY	Unknown policy

kadm5_get_principal (get principal information)

Purpose

Returns information from a principal entry in the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_get_principal (
    void *                              server_handle,
    krb5_principal                      principal,
    kadm5_principal_ent_t               entry,
    krb5_flags                          mask)
```

Parameters

Input

server_handle
    Specifies the server handle for the session with the administration server.

principal
    Specifies the principal entry to be returned.

mask
    Specifies the information to be returned. The following flags can be ORed together to define the mask:

*Table 12. Flags for **mask** parameter for **kadm5_get_principal()***

Flag	Explanation
KADM5_ATTRIBUTES	Returns the principal attributes.
KADM5_AUX_ATTRIBUTES	Returns the auxilliary attributes.
KADM5_FAIL_AUTH_COUNT	Returns the number of failed authentication attempts.
KADM5_KEY_DATA	Returns the key data.
KADM5_KVNO	Returns the current key version number.
KADM5_LAST_FAILED	Returns the time of the last failed authentication.
KADM5_LAST_PWD_CHANGE	Returns the last password change time.
KADM5_LAST_SUCCESS	Returns the time of the last successful authentication.
KADM5_MAX_LIFE	Returns the maximum ticket lifetime.
KADM5_MAX_RLIFE	Returns the maximum renewable lifetime.

Table 12. Flags for *mask* parameter for *kadm5_get_principal()* (continued)	
Flag	**Explanation**
KADM5_MKVNO	Returns the master key version number.
KADM5_MOD_NAME	Returns the name of the principal making the last modification.
KADM5_MOD_TIME	Returns the time of the last modification.
KADM5_POLICY	Returns the policy name.
KADM5_PRINCIPAL	Returns the principal name.
KADM5_PRINC_EXPIRE_TIME	Returns the account expiration time.
KADM5_PW_EXPIRATION	Returns the password expiration time.
KADM5_PRINCIPAL_FULL_MASK	Returns all information.
KADM5_PRINCIPAL_NORMAL_MASK	Returns all information except the key data and the tagged data.
KADM5_TL_DATA	Returns the tagged data

Output

entry
Returns the requested information. The storage allocated for the principal entry should be released when it is no longer needed by calling the **kadm5_free_principal_ent()** routine.

Usage

The **kadm5_get_principal()** routine returns information from a principal entry in the Kerberos database. Some of the fields may not be available, depending upon the Kerberos database implementation. For KADM5_KEY_DATA, the key contents are not returned. For KADM5_TL_DATA, the returned data is dependent upon the database implementation. You must have GET authority or the requested principal entry must be your own entry.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_get_principal()** routine:

Table 13. Common errors returned by the *kadm5_get_principal()* routine	
Function	**Error**
KADM5_AUTH_GET	Not authorized to get entry.
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified.
KADM5_BAD_PRINCIPAL	Principal is missing or is not valid.
KADM5_BAD_SERVER_HANDLE	Server handle is not valid.
KADM5_GSS_ERROR	GSS-API error.
KADM5_RPC_ERROR	Communication error.
KADM5_UNK_PRINCIPAL	Unknown principal.

kadm5_get_principals (return a list of principals)

Purpose

Returns a list of principals matching the specified search expression.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_get_principals (
    void *server_handle,
    char *expression,
    char ***princs,
    int *count)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

expression
Specifies the search expression. The maximum string length is 1024 bytes. All principals are listed if NULL is specified for this parameter.

Output

princs
Returns the list of principal names matching the search expression. The list should be released when it is no longer needed by calling the **kadm5_free_name_list()** routine.

count
Returns the number of entries in the list.

Usage

The **kadm5_get_principals()** routine returns a list of principal names matching a search expression. You must have LIST authority to list entries in the Kerberos database. The list of matching principal names may be restricted by additional database authorization checking depending upon the database implementation.

The search expression can include the "*" and "?" wildcards where "*" represents zero or more characters and "?" represents a single character. For example, the expression "*/admin@*" returns all principal names that end with "/admin," the expression "rwh*" returns all principal names that begin with "rwh," and the expression "test_client?@*" returns principal names such as **test_client1**, **test_client2**, and so forth. You can use "*" and "\?" to search for a "*" or "?" character instead of treating the characters as wildcards.

The search string can also contain paired "[" and "]" characters with one or more characters between the brackets. A match occurs if a name contains one of the characters between the brackets. For example, the expression "*/[ad]*" returns all names containing "/a" and "/d" while the expression "[ckr]*" returns all names beginning with "c," "k," or "r." You can use "\[" and "\]" to search for a "[" or "]" character.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_get_principals()** routine:

*Table 14. Common errors returned by the **kadm5_get_principals()** routine*

Function	Error
KADM5_AUTH_LIST	Not authorized to list entries.
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified.
KADM5_BAD_SERVER_HANDLE	Server handle is not valid.

Table 14. Common errors returned by the **kadm5_get_principals()** routine (continued)	
Function	**Error**
KADM5_GSS_ERROR	GSS-API error.
KADM5_RPC_ERROR	Communication error.
KADM5_TOO_MANY_MATCHES	Too many database entries match the search expression.

kadm5_get_privs (return administration privileges)

Purpose

Returns the administration privileges for the authenticated client.

Format

```
#include <skrb/admin.h>

kadm5_ret_t kadm5_get_privs (
    void *                          server_handle,
    krb5_flags *                    privs)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

Output

privs
Returns the administration privileges bit mask. The following flags are defined:

- KADM5_PRIV_ADD - Authorized to add an entry to the database
- KADM5_PRIV_CHPW - Authorized to change the password for a principal
- KADM5_PRIV_DELETE - Authorized to delete an entry from the database
- KADM5_PRIV_GET - Authorized to get an entry from the database
- KADM5_PRIV_LIST - Authorized to list the names of database entries
- KADM5_PRIV_MODIFY - Authorized to modify an entry in the database
- KADM5_PRIV_SETKEY - Authorized to set the key for a principal

Usage

The **kadm5_get_privs()** routine returns the administrative privileges for the authenticated client. Some of the privileges may not be implemented, depending upon the Kerberos database implementation. Additional authorization checking may be performed, depending upon the requested administration function or the database implementation.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_get_privs()** routine:

Table 15. Common errors returned by the **kadm5_get_principals()** routine	
Function	**Error**
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified.

Table 15. Common errors returned by the **kadm5_get_principals()** routine (continued)	
Function	**Error**
KADM5_BAD_SERVER_HANDLE	Server handle is not valid.
KADM5_GSS_ERROR	GSS-API error.
KADM5_RPC_ERROR	Communication error.

kadm5_init_with_creds (establish a session using credentials)

Purpose

Establish a session with the Kerberos administration server using a credentials cache for authentication.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_init_with_creds (
    char *                          client_name,
    krb5_ccache                     ccache,
    char *                          service_name,
    kadm5_config_params *           config_params,
    krb5_ui_4                       struct_version,
    krb5_ui_4                       api_version,
    void **                         server_handle)
```

Parameters

Input

client_name
Specifies the client name for the session. The local realm is used if a fully-qualified name is not specified.

ccache
Specifies the credentials cache for the session. The credentials cache must contain an initial ticket for the administration service. This ticket must be valid for at least the next 10 minutes.

service_name
Specifies the server name for the session. This is usually **kadmin/admin**. The realm name is obtained from the configuration parameters if a fully-qualified name is not specified.

config_params
Specifies configuration parameter override values. Specify NULL for this parameter if no overrides are needed. These mask values may be set:

Table 16. Mask values for **config_params** parameter for **kadm5_init_with_creds()**	
Mask	**Explanation**
KADM5_CONFIG_PROFILE	The profile field contains the name of the Kerberos profile to be used. The default Kerberos profile is used if this value is not specified.
KADM5_CONFIG_REALM	The realm field contains the name of the administration server realm. The client realm is used if this value is not specified.

Table 16. Mask values for *config_params* parameter for *kadm5_init_with_creds()* (continued)	
Mask	**Explanation**
`KADM5_CONFIG_ADMIN_SERVER`	The *admin_server* field contains the name of the host system running the Kerberos administration server in the format *host:port*. The value of the *kadmind_port* field is used for the port number if the port is not explicitly specified. The host name is obtained from the Kerberos profile if neither KADM5_CONFIG_ADMIN_SERVER nor KADM5_CONFIG_ADMIN_SERVER_LIST is specified. The *admin_server* field is used if both KADM5_CONFIG_ADMIN_SERVER and KADM5_CONFIG_ADMIN_SERVER_LIST are specified.
`KADM5_CONFIG_ADMIN_SERVER_LIST`	The *admin_server_list* field contains a list of Kerberos administration servers. Each list entry is in the format *host:port* and the list is terminated by a NULL address. The value of the *kadmind_port* field is used for the port number if an entry does not explicitly specify the port. The host name is obtained from the Kerberos profile if neither KADM5_CONFIG_ADMIN_SERVER nor KADM5_CONFIG_ADMIN_SERVER_LIST is specified. The *admin_server* field is used if both KADM5_CONFIG_ADMIN_SERVER and KADM5_CONFIG_ADMIN_SERVER_LIST are specified.
`KADM5_CONFIG_KADMIND_PORT`	The *kadmind_port* field contains the port number of the Kerberos administration server and defaults to 749.

struct_version
    Specifies the structure version and should be set to KADM5_STRUCT_VERSION to use the current structure version.

api_version
    Specifies the API version and should be set to KADM5_API_VERSION to use the current API version.

Output

server_handle
    Returns the opaque server handle representing the session with the administration server.

Usage

The **kadm5_init_with_creds()** routine establishes a session with the Kerberos administration server using the credentials cache supplied by the caller. The credentials cache must contain an initial ticket to the administration service. The **kadm5_destroy()** routine should be called to end the session and release resources.

The service name can be **kadmin/admin** or **kadmin/changepw**. The **kadmin/admin** service is the administration service, and the **kadmin/changepw** service is the password change service. All of the administration functions are available using **kadmin/admin,** and their use is controlled by the privileges granted to the authenticating principal. Only the following services are available using **kadmin/changepw** and their use requires the principal to be the same as the authenticating principal: **kadm5_chpass_principal**, **kadm5_randkey_principal**, **kadm5_get_principal**, and **kadm5_get_policy**.

The Kerberos administration API does not establish its own signal handlers since this could conflict with the application's use of signals (signal handlers have a process-wide scope). Consequently, the application should set up its own signal handler for the SIGPIPE signal. The action routine can be SIG_IGN unless the application needs to perform its own processing for a broken pipe.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_init_with_creds()** routine:

*Table 17. Common errors returned by the **kadm5_init_with_creds()** routine*

Function	Error
KADM5_BAD_CLIENT_PARAMS	Incorrect parameters specified
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error
KADM5_NO_SRV	No administration server is defined for the target realm
KADM5_SECURE_PRINC_MISSING	Administration server principal is not defined

kadm5_init_with_password (establish a session using a password)

Purpose

Establishes a session with the Kerberos administration server using a password for authentication.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_init_with_password (
    char *                                      client_name,
    char *                                      password,
    char *                                      service_name,
    kadm5_config_params *                       config_params,
    krb5_ui_4                                   struct_version,
    krb5_ui_4                                   api_version,
    void **                                     server_handle)
```

Parameters

Input

client_name
Specifies the client name for the session. The local realm is used if a fully-qualified name is not specified.

password
Specifies the client password. Specify NULL for this parameter to prompt the user to enter the password.

service_name
Specifies the server name for the session. This is usually **kadmin/admin**. The realm name is obtained from the configuration parameters if a fully-qualified name is not specified.

config_params
Specifies configuration parameter override values. Specify NULL for this parameter if no overrides are needed. These mask values may be set:

*Table 18. Mask values for **config_params** parameter for **kadm5_init_with_password()***	
Mask	**Explanation**
KADM5_CONFIG_PROFILE	The profile field contains the name of the Kerberos profile to be used. The default Kerberos profile is used if this value is not specified.
KADM5_CONFIG_REALM	The realm field contains the name of the administration server realm. The client realm is used if this value is not specified.
KADM5_CONFIG_ADMIN_SERVER	The *admin_server* field contains the name of the host system running the Kerberos administration server in the format *host:port*. The value of the *kadmind_port* field is used for the port number if the port is not explicitly specified. The host name is obtained from the Kerberos profile if neither KADM5_CONFIG_ADMIN_SERVER nor KADM5_CONFIG_ADMIN_SERVER_LIST is specified. The *admin_server* field is used if both KADM5_CONFIG_ADMIN_SERVER and KADM5_CONFIG_ADMIN_SERVER_LIST are specified.
KADM5_CONFIG_ADMIN_SERVER_LIST	The *admin_server_list* field contains a list of Kerberos administration servers. Each list entry is in the format *host:port* and the list is terminated by a NULL address. The value of the *kadmind_port* field is used for the port number if an entry does not explicitly specify the port. The host name is obtained from the Kerberos profile if neither KADM5_CONFIG_ADMIN_SERVER nor KADM5_CONFIG_ADMIN_SERVER_LIST is specified. The *admin_server* field is used if both KADM5_CONFIG_ADMIN_SERVER and KADM5_CONFIG_ADMIN_SERVER_LIST are specified.
KADM5_CONFIG_KADMIND_PORT	The *kadmind_port* field contains the port number of the Kerberos administration server and defaults to 749.

struct_version
> Specifies the structure version and should be set to KADM5_STRUCT_VERSION to use the current structure version.

api_version
> Specifies the API version and should be set to KADM5_API_VERSION to use the current API version.

Output

server_handle
> Returns the opaque server handle representing the session with the administration server.

Usage

The **kadm5_init_with_password()** routine establishes a session with the Kerberos administration server. The supplied password is used to obtain an initial ticket for the administration service. The **kadm5_destroy()** routine should be called to end the session and release resources.

The service name can be **kadmin/admin** or **kadmin/changepw**. The **kadmin/admin** service is the administration service, and the **kadmin/changepw** service is the password change service. All of the administration functions are available using **kadmin/admin** and their use is controlled by the privileges granted to the authenticating principal. Only the following services are available using **kadmin/changepw** and their use requires the principal to be the same as the authenticating principal: **kadm5_chpass_principal**, **kadm5_randkey_principal**, **kadm5_get_principal**, and **kadm5_get_policy**.

The Kerberos administration API does not establish its own signal handlers because this could conflict with the application's use of signals (signal handlers have a process-wide scope). Consequently, the application should set up its own signal handler for the SIGPIPE signal. The action routine can be SIG_IGN unless the application needs to perform its own processing for a broken pipe.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_init_with_password()** routine:

Table 19. Common errors returned by the **kadm5_init_with_password()** routine

Function	Error
KADM5_BAD_CLIENT_PARAMS	Incorrect parameters specified
KADM5_BAD_PASSWORD	Incorrect password specified
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error
KADM5_NO_SRV	No administration server is defined for the target realm
KADM5_SECURE_PRINC_MISSING	Administration server principal is not defined

kadm5_init_with_skey (establish a session using a key table)

Purpose

Establish a session with the Kerberos administration server using a key table for authentication.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_init_with_skey (
    char *                              client_name,
    char *                              keytab_name,
    char *                              service_name,
    kadm5_config_params *               config_params,
    krb5_ui_4                           struct_version,
    krb5_ui_4                           api_version,
    void **                             server_handle)
```

Parameters

Input

client_name
> Specifies the client name for the session. The local realm is used if a fully-qualified name is not specified.

keytab_name
> Specifies the key table name. The key table must contain the current key for the client.

service_name
> Specifies the server name for the session. This is usually **kadmin/admin**. The realm name is obtained from the configuration parameters if a fully-qualified name is not specified.

config_params

Specifies configuration parameter override values. Specify NULL for this parameter if no overrides are needed. These mask values may be set:

*Table 20. Mask values for **config_params** parameter for **kadm5_init_with_skey()***

Mask	Explanation
KADM5_CONFIG_PROFILE	The profile field contains the name of the Kerberos profile to be used. The default Kerberos profile is used if this value is not specified.
KADM5_CONFIG_REALM	The realm field contains the name of the administration server realm. The client realm is used if this value is not specified.
KADM5_CONFIG_ADMIN_SERVER	The *admin_server* field contains the name of the host system running the Kerberos administration server in the format *host:port*. The value of the *kadmind_port* field is used for the port number if the port is not explicitly specified. The host name is obtained from the Kerberos profile if neither KADM5_CONFIG_ADMIN_SERVER nor KADM5_CONFIG_ADMIN_SERVER_LIST is specified. The *admin_server* field is used if both KADM5_CONFIG_ADMIN_SERVER and KADM5_CONFIG_ADMIN_SERVER_LIST are specified.
KADM5_CONFIG_ADMIN_SERVER_LIST	The *admin_server_list* field contains a list of Kerberos administration servers. Each list entry is in the format *host:port* and the list is terminated by a NULL address. The value of the *kadmind_port* field is used for the port number if an entry does not explicitly specify the port. The host name is obtained from the Kerberos profile if neither KADM5_CONFIG_ADMIN_SERVER nor KADM5_CONFIG_ADMIN_SERVER_LIST is specified. The *admin_server* field is used if both KADM5_CONFIG_ADMIN_SERVER and KADM5_CONFIG_ADMIN_SERVER_LIST are specified.
KADM5_CONFIG_KADMIND_PORT	The *kadmind_port* field contains the port number of the Kerberos administration server and defaults to 749.

struct_version

Specifies the structure version and should be set to KADM5_STRUCT_VERSION to use the current structure version.

api_version

Specifies the API version and should be set to KADM5_API_VERSION to use the current API version.

Output

server_handle

Returns the opaque server handle representing the session with the administration server.

Usage

The **kadm5_init_with_skey()** routine establishes a session with the Kerberos administration server. The key table is used to obtain an initial ticket for the administration service. The **kadm5_destroy()** routine should be called to end the session and release resources.

The service name can be **kadmin/admin** or **kadmin/changepw**. The **kadmin/admin** service is the administration service, and the **kadmin/changepw** service is the password change service. All of the administration functions are available using **kadmin/admin** and their use is controlled by the privileges granted to the authenticating principal. Only the following services are available using **kadmin/changepw** and their use requires the principal to be the same as the authenticating principal: **kadm5_chpass_principal**, **kadm5_randkey_principal**, **kadm5_get_principal**, and **kadm5_get_policy**.

The Kerberos administration API does not establish its own signal handlers because this could conflict with the application's use of signals (signal handlers have a process-wide scope). Consequently, the application should set up its own signal handler for the SIGPIPE signal. The action routine can be SIG_IGN unless the application needs to perform its own processing for a broken pipe.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_init_with_skey()** routine:

*Table 21. Common errors returned by the **kadm5_init_with_skey()** routine*

Function	Error
KADM5_BAD_CLIENT_PARAMS	Incorrect parameters specified
KADM5_BAD_PASSWORD	Incorrect password specified
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error
KADM5_NO_SRV	No administration server is defined for the target realm
KADM5_SECURE_PRINC_MISSING	Administration server principal is not defined

kadm5_modify_policy (modify a policy entry)

Purpose

Modifies a policy entry in the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_modify_policy (
    void *                          server_handle,
    kadm5_policy_ent_t              entry,
    krb5_flags                      mask)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

entry
Specifies the information for the policy entry. The policy name is obtained from the *policy* field of the entry (the KADM5_POLICY mask flag must not be set since you cannot change the policy name).

mask
Specifies the fields in the **krb5_policy_ent_t** that are to be used to modify the policy entry. The following flags can be ORed together to define the mask:

- KADM5_PW_HISTORY_NUM - the password history count is set
- KADM5_PW_MIN_CLASSES - the minimum number of password character classes is set
- KADM5_PW_MIN_LENGTH - the minimum password length is set
- KADM5_PW_MIN_LIFE - the minimum password lifetime is set
- KADM5_PW_MAX_LIFE - the maximum password lifetime is set

Usage

The **kadm5_modify_policy()** routine modifies a policy entry in the Kerberos database. You must have MODIFY authority.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_modify_policy()** routine:

*Table 22. Common errors returned by the **kadm5_modify_policy()** routine*

Function	Error
KADM5_AUTH_MODIFY	Not authorized to modify an entry
KADM5_BAD_CLASS	Character class count is not valid
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_BAD_HISTORY	Password history count is not valid
KADM5_BAD_LENGTH	Minimum password length is not valid
KADM5_BAD_MASK	Incorrect policy modification mask specified
KADM5_BAD_MIN_PASS_LIFE	Minimum password lifetime is not valid
KADM5_BAD_POLICY	Policy name is not valid
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error
KADM5_UNK_POLICY	Unknown policy

kadm5_modify_principal (modify a principal entry)

Purpose

Modifies a principal entry in the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_modify_principal (
    void *                              server_handle,
    kadm5_principal_ent_t               entry,
    krb5_flags                          mask)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

entry
Specifies the information for the principal entry. The principal name is obtained from the *principal* field of the entry (the KADM5_PRINCIPAL mask flag must not be set since you cannot change the principal name using the **kadm5_modify_principal()** routine).

mask
Specifies the fields in the **krb5_principal_ent_t** that are to be used to modify the principal entry. The following flags can be ORed together to define the mask:

Table 23. Flags for **mask** parameter for **kadm5_modify_principal()**

Flag	Explanation
KADM5_ATTRIBUTES	The principal attributes are set.
KADM5_FAIL_AUTH_COUNT	The number of failed authentication attempts is set.
KADM5_KVNO	The current key version number is set.
KADM5_MAX_LIFE	The maximum ticket lifetime is set.
KADM5_MAX_RLIFE	The maximum renewable lifetime is set.
KADM5_POLICY	The policy name is set.
KADM5_POLICY_CLR	The policy name is cleared.
KADM5_PRINC_EXPIRE_TIME	The account expiration time is set.
KADM5_PW_EXPIRATION	The password expiration time is set.
KADM5_TL_DATA	The tagged data is set.

Usage

The **kadm5_modify_principal()** routine modifies a principal entry in the Kerberos database. You must have MODIFY authority. The principal name and password cannot be changed using **kadm5_modify_principal()**. The fields that can be modified are dependent upon the Kerberos database implementation.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_modify_principal()** routine:

Table 24. Common errors returned by the **kadm5_modify_principal()** routine

Function	Error
KADM5_AUTH_MODIFY	Not authorized to modify an entry
KADM5_BAD_CLIENT_PARAMS	Incorrect parameters specified
KADM5_BAD_MASK	Incorrect principal modification mask specified
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_GSS_ERROR	GSS-API error
KADM5_RPC_ERROR	Communication error
KADM5_UNK_POLICY	Specified policy does not exist

Table 24. Common errors returned by the *kadm5_modify_principal()* routine (continued)	
Function	Error
KADM5_UNK_PRINC	Specified principal does not exist

kadm5_randkey_principal (generate random keys)

Purpose

Generates a new set of random keys for a principal.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_randkey_principal (
    void *                           server_handle,
    krb5_principal                   principal,
    krb5_keyblock **                 new_keys,
    int *                            n_keys)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

principal
Specifies the principal.

Output

new_keys
Returns an array of Kerberos keys generated as a result of this request. The **kadm5_free_key_list()** routine should be called to release the keys when they are no longer needed. Specify NULL for this parameter if you don't need to have the keys returned.

n_keys
Returns the number of keys in the returned key list. You can specify NULL for this parameter if you specified NULL for the *new_keys* parameter.

Usage

The **kadm5_randkey_principal()** routine generates a new set of random keys for the specified principal. You must have CHANGEPW authority, the specified principal must be your own principal, or the administration session must be with the **kadmin/changepw** service.

The **kadm5_randkey_principal()** routine generates an encryption key for each encryption type supported by the Kerberos administration server. Use the **kadm5_randkey_principal_3()** routine if you want to generate encryption keys for a subset of the available encryption types.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_randkey_principal()** routine:

Table 25. Common errors returned by the *kadm5_randkey_principal()* routine	
Function	Error
KADM5_AUTH_CHANGEPW	Not authorized to change the password
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified

Table 25. Common errors returned by the **kadm5_randkey_principal()** routine (continued)	
Function	**Error**
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_GSS_ERROR	GSS-API error
KADM5_PASS_TOOSOON	The minimum password lifetime has not elapsed
KADM5_PROTECT_PRINCIPAL	The principal is protected and may not be modified
KADM5_RPC_ERROR	Communication error
KADM5_UNK_PRINCIPAL	Unknown principal

kadm5_randkey_principal_3 (generate random keys)

Purpose

Generates a new set of random keys for a principal.

Format

```
#include <skrb/admin.h>

kadm5_ret_t kadm5_randkey_principal_3 (
    void *                              server_handle,
    krb5_principal                      principal,
    krb5_boolean                        keepold,
    int                                 n_ks_entries,
    krb5_key_salt_tuple *               ks_entries,
    krb5_keyblock **                    new_keys,
    int *                               n_keys);
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

principal
Specifies the principal.

keepold
Specifies whether to keep the old key entries.

n_ks_entries
Specifies the number of key-salt entries.

ks_entries
Specifies an array of key-salt entries.

Output

new_keys
Returns an array of Kerberos keys generated as a result of this request. The **kadm5_free_key_list()** routine should be called to release the keys when they are no longer needed. Specify NULL for this parameter if you don't need to have the keys returned.

n_keys
Returns the number of keys in the returned key list. You can specify NULL for this parameter if you specified NULL for the *new_keys* parameter.

Usage

The **kadm5_randkey_principal_3()** routine generates a new set of random keys for the specified principal. You must have CHANGEPW authority, the specified principal must be your own principal, or the administration session must be with the **kadmin/changepw** service.

The **kadm5_randkey_principal_3()** routine allows the specification of the encryption types used to generate encryption keys. It is the same as the **kadm5_randkey_principal()** routine if no key-salt entries are provided. An error is returned if an unsupported encryption type or salt type is specified.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_randkey_principal_3()** routine:

*Table 26. Common errors returned by the **kadm5_randkey_principal()** routine*

Function	Error
KADM5_AUTH_CHANGEPW	Not authorized to change the password.
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified.
KADM5_BAD_SERVER_HANDLE	Server handle is not valid.
KADM5_GSS_ERROR	GSS-API error.
KADM5_PASS_TOOSOON	The minimum password lifetime has not elapsed.
KADM5_PROTECT_PRINCIPAL	The principal is protected and may not be modified.
KADM5_RPC_ERROR	Communication error.
KADM5_UNK_PRINCIPAL	Unknown principal.

kadm5_rename_principal (rename a principal entry)

Purpose

Renames a principal entry in the Kerberos database.

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_rename_principal (
    void *                          server_handle,
    krb5_principal                  old_name,
    krb5_principal                  new_name)
```

Parameters

Input

server_handle
    Specifies the server handle for the session with the administration server.

old_name
    Specifies the name of the entry to be renamed.

new_name
    Specifies the new name for the entry.

Usage

The **kadm5_rename_principal()** routine renames a principal entry in the Kerberos database. You must have both ADD and DELETE authority.

Since the principal name is often used as part of the password salt, you should change the password for the principal after the entry is renamed. Some implementations of the Kerberos administration server do not allow a principal to be renamed if the principal name is used in the password salt. In this case, you must delete the existing principal entry and add the new principal entry using the **kadm5_delete_principal()** and **kadm5_create_principal()** routines.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_rename_principal()** routine:

*Table 27. Common errors returned by the **kadm5_rename_principal()** routine*

Function	Error
KADM5_AUTH_ADD	Not authorized to add an entry
KADM5_AUTH_DELETE	Not authorized to delete an entry
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_DUP	Duplicate entry
KADM5_GSS_ERROR	GSS-API error
KADM5_NO_RENAME_SALT	Password salt type does not allow the principal to be renamed
KADM5_RPC_ERROR	Communication error
KADM5_UNK_PRINCIPAL	Unknown principal

kadm5_setkey_principal (set the key for a principal entry)

Purpose

Sets the key for a principal entry in the Kerberos database

Format

```
#include <skrb/admin.h>
kadm5_ret_t kadm5_setkey_principal (
    void *                          server_handle,
    krb5_principal                  principal,
    krb5_keyblock *                 keys,
    int                             n_keys)
```

Parameters

Input

server_handle
Specifies the server handle for the session with the administration server.

principal
Specifies the principal entry.

keys
Specifies an array of keys.

n_keys
Specifies the number of entries in the key array.

Usage

The **kadm5_setkey_principal()** routine sets the keys for a principal entry in the Kerberos database. You must have SETKEY authority. No policy checks are performed on the new keys. The supplied keys replace the current encryption keys for the principal.

The key array must contain an entry for each unique encryption key that can be used by the principal. However, there must not be duplicate entries for encryption types that use the same encryption key. For example, encryption types ENCTYPE_DES_CBC_CRC and ENCTYPE_DES_CBC_MD5 both use the same 56-bit DES encryption key. You can specify either ENCTYPE_DES_CBC_CRC or ENCTYPE_DES_CBC_MD5, but you cannot specify both.

The **kadm5_setkey_principal()** routine use the default salt for each encryption key. Use the **kadm5_setkey_principal_3()** routine if you want to specify a different salt.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_setkey_principal()** routine:

*Table 28. Common errors returned by the **kadm5_setkey_principal()** routine*

Function	Error
KADM5_AUTH_SETKEY	Not authorized to set the keys for the entry
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified
KADM5_BAD_SERVER_HANDLE	Server handle is not valid
KADM5_GSS_ERROR	GSS-API error
KADM5_PROTECT_PRINCIPAL	Protected principal cannot be modified
KADM5_RPC_ERROR	Communication error
KADM5_SETKEY_DUP_ENCTYPES	Duplicate encryption key types specified
KADM5_UNK_PRINCIPAL	Unknown principal

kadm5_setkey_principal_3 (set the key for a principal entry)

Purpose

Sets the key for a principal entry in the Kerberos database

Format

```
#include <skrb/admin.h>

kadm5_ret_t kadm5_setkey_principal_3 (
    void *                              server_handle,
    krb5_principal                      principal,
    krb5_boolean                        keepold,
    int                                 n_ks_entries,
    krb5_key_salt_tuple *               ks_entries,
    krb5_keyblock *                     keys,
    int                                 n_keys
```

Parameters

Input

server_handle
　　Specifies the server handle for the session with the administration server.

principal
Specifies the principal entry.

keepold
Specifies whether to keep the old key entries.

n_ks_entries
Specifies the number of key-salt entries.

ks_entries
Specifies an array of key-salt entries.

keys
Specifies an array of keys.

n_keys
Specifies the number of entries in the key array.

Usage

The **kadm5_setkey_principal(_3)** routine sets the keys for a principal entry in the Kerberos database. You must have SETKEY authority. No policy checks are performed on the new keys. The supplied keys replace the current encryption keys for the principal.

The key array must contain an entry for each unique encryption key that can be used by the principal. However, there must not be duplicate entries for encryption types that use the same encryption key. For example, encryption types ENCTYPE_DES_CBC_CRC and ENCTYPE_DES_CBC_MD5 both use the same 56-bit DES encryption key. You can specify either ENCTYPE_DES_CBC_CRC or ENCTYPE_DES_CBC_MD5, but you cannot specify both.

The key-salt entries are used to specify the salt associated with each key. The number of key-salt entries must be the same as the number of keys and the encryption type in each key-salt entry must match the encryption type of the corresponding key. The **kadm5_setkey_principal_3()** routine is the same as the k**adm5_setkey_principal()** routine if no key-salt entries are specified.

The function return value is zero if no errors occurred. Otherwise, it is a Kerberos error code. These are some of the common errors returned by the **kadm5_setkey_principal_3()** routine:

Function	Error
*Table 29. Common errors returned by the **kadm5_setkey_principal()** routine*	
KADM5_AUTH_SETKEY	Not authorized to set the keys for the entry.
KADM5_BAD_CLIENT_PARAMS	Incorrect parameter specified.
KADM5_BAD_SERVER_HANDLE	Server handle is not valid.
KADM5_GSS_ERROR	GSS-API error.
KADM5_PROTECT_PRINCIPAL	Protected principal cannot be modified.
KADM5_RPC_ERROR	Communication error.
KADM5_SETKEY_DUP_ENCTYPES	Duplicate encryption key types specified.
KADM5_SETKEY3_ETYPE_MISMATCH	The key-salt entries do not match the key entries.
KADM5_UNK_PRINCIPAL	Unknown principal.

Part 2. GSS-API interfaces

This Part introduces the GSS-API interfaces and describes each one. These topics are covered:

- Introduction to GSS-API
  - General information about GSS-API
  - GSS-API services
  - Error handling
  - Data types
  - GSS-API version compatibility
  - Interoperability with Microsoft Windows 2000 SSPI
- GSS-API programming interfaces
- GSS-API programming interfaces - Kerberos mechanism.

Chapter 4. Introduction to GSS-API

This chapter contains general information about the Generic Security Service Application Programming Interface (GSS-API). It also includes an overview of error handling, data types, and calling conventions. For a list of supported RFCs, see *z/OS Integrated Security Services Network Authentication Service Administration*.

General information about GSS-API

The Generic Security Service Application Programming Interface (GSS-API) provides security services to applications using peer-to-peer communications. Using GSS-API routines, applications can perform these operations:

- Enable an application to determine another application's user identification
- Enable an application to delegate access rights to another application
- Apply security services, such as confidentiality and integrity, on a per-message basis.

A secure connection between two communicating applications is represented by a data structure called a *security context*. The application that establishes the secure connection is called the *context initiator*. The context initiator is similar to a remote procedure call (RPC) client. The application that accepts the secure connection is the *context acceptor*. The context acceptor is similar to an RPC server. The GSS-API routines use *tokens* as input and output values. The communicating applications are responsible for exchanging these tokens using whatever communication channels are appropriate.

There are four stages involved in using the GSS-API:

1. The context initiator acquires a credential for proving its identity to other processes. Similarly, the context acceptor acquires a credential for accepting a security context. Either application may omit this credential acquisition and use its default credential.

   Each application uses credentials to establish its global identity. The global identity can be, but is not necessarily, related to the local user name the application runs under. Credentials can be obtained from an existing login context or can be created using a principal name and key obtained from a key table.

2. The communicating applications establish a joint security context by exchanging authentication tokens.

   The security context is a pair of GSS-API data structures containing information that is shared between the communicating applications. The information describes the state of each application. This security context is required for per-message security services.

   To establish a security context, the context initiator calls the **gss_init_sec_context()** routine to get a token. The token is cryptographically protected, opaque data. The context initiator transfers the token to the context acceptor, which in turn passes the token to the **gss_accept_sec_context()** routine to decode and extract the shared information.

   As part of establishing the security context, the context initiator is authenticated to the context acceptor. The context initiator can require the context acceptor to authenticate itself in return by requesting mutual authentication.

   The context initiator can delegate rights to allow the context acceptor to act as its agent. Delegation means the context initiator gives the context acceptor the ability to initiate additional security contexts as an agent of the context initiator. To delegate, the context initiator sets a flag on the call to the **gss_init_sec_context()** routine indicating that it wants to delegate, and sends the returned token in the normal way to the context acceptor. The acceptor passes this token to the **gss_accept_sec_context()** routine, which generates a delegated credential. The context acceptor can use the returned credential to initiate additional security contexts with other applications.

3. The applications exchange protected messages and data.

The applications can call GSS-API routines to protect data exchanged in messages. GSS-API treats application data as arbitrary octet strings. The GSS-API message security services can provide either integrity and authentication of data origin or confidentiality, integrity, and authentication of data origin. The capability to provide data confidentiality is dependent upon the capabilities of the underlying data encryption support.

4. When the applications have finished communicating, either one may instruct GSS-API to delete the security context.

There are several types of GSS-API routines:

- Standard GSS-API routines. These routines have the prefix **gss_**.
- Kerberos extensions to the GSS-API. These are additional routines that enable an application to use Kerberos security services. These routines have the prefix **gss_krb5**.

GSS-API services

Message integrity and confidentiality

GSS-API provides message security services. Depending upon the underlying security mechanism capabilities, message integrity and message confidentiality services are available. When a security context is established, the GSS-API routines return two flags to indicate the set of message protection security services available for the context:

- The GSS_C_INTEG_FLAG indicates whether message integrity and origin authenticity services are available
- The GSS_C_CONF_FLAG indicates whether message confidentiality services are available. This flag is never TRUE unless the GSS_C_INTEG_FLAG is also TRUE.

GSS-API callers that want message security services should check the values of these flags at context establishment time and must be aware that a returned FALSE value means that the invocation of the **gss_get_mic()** and **gss_wrap()** routines applies no cryptographic protection to user data messages.

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

Message replay and sequencing

GSS-API also provides message sequencing and replay detection services. These selectable protection features are distinct from the replay detection and sequencing features supplied by the context establishment operation. The presence or absence of context-level replay or sequencing is a function of the underlying security mechanism layer capabilities and is not selected or omitted as a caller option.

The caller initiating a context provides two flags to specify whether the use of message replay detection and sequencing features is wanted on the context being established:

- GSS_C_REPLAY_FLAG indicates whether message replay detection services are to be used
- GSS_C_SEQUENCE_FLAG indicates whether message sequencing services are to be used.

The GSS-API implementation at the initiator system can determine whether these services are supported as a function of the mechanism type. When enabled, these services provide recipients with indicators as a result of GSS-API processing on incoming messages, identifying whether those messages were detected as duplicate or out-of-sequence. Detection of such events does not prevent a suspect message from being provided to a recipient; the appropriate course of action on a suspect message is a matter of caller policy.

When replay detection is enabled, the possible **major_status** returns for well-formed and correctly signed messages are:

- GSS_S_COMPLETE indicates that the message was within the window (of time or sequence space) allowing replay events to be detected, and the message was not a replay of a previously-processed message within that window.
- GSS_S_DUPLICATE_TOKEN indicates that the cryptographic check value on the received message was correct, but the message was recognized as a duplicate of a previously-processed message.
- GSS_S_OLD_TOKEN indicates that the cryptographic check value on the received message was correct, but the message is too old to be checked for duplication.

When message sequencing is enabled, the possible returns for well-formed and correctly signed messages are:

- GSS_S_COMPLETE indicates that:
  – The message was within the window (of time or sequence space) allowing replay events to be detected
  – The message was not a replay of a previously-processed message within that window, *and*
  – No predecessor sequenced messages are missing relative to the last received message processed on the context with a correct cryptographic check value.
- GSS_S_DUPLICATE_TOKEN indicates that the integrity check value on the received message was correct, but the message was recognized as a duplicate of a previously-processed message.
- GSS_S_OLD_TOKEN indicates that the integrity check value on the received message was correct, but the token is too old to be checked for duplication.
- GSS_S_UNSEQ_TOKEN indicates that the cryptographic check value on the received message was correct, but it is earlier in a sequence stream than a message already processed on the context.
- GSS_S_GAP_TOKEN indicates that the cryptographic check value on the received message was correct, but one or more predecessor sequenced messages have not be successfully processed relative to the last received message on the context with a correct cryptographic check value.

Quality of protection

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

Anonymity

In certain situations or environments, an application may want to authenticate a peer or protect communications (or both) using GSS-API message services without revealing its own identity. In ordinary GSS-API usage, a context initiator's identity is made available to the context acceptor as part of the context establishment process.

To provide for anonymity support, a GSS_C_ANON_FLAG is provided for context initiators to request that their identity not be given to the context acceptor. Mechanisms are not required to honor this request, but a caller is informed through the return flags whether the request was honored. Note that authentication as the anonymous principal does not necessarily imply that credentials are not required in order to establish a context

Error handling

Each GSS-API routine returns two status values:

Major status
> Major status values are generic API errors. They are the same for all implementations of GSS-API and are not dependent upon the underlying mechanism. For more details, see *z/OS Integrated Security Services Network Authentication Service Administration*.

Minor status
> Minor status values are mechanism-specific errors that further define the error reported. Minor status values are not portable between implementations of GSS-API and vary across mechanisms.

When designing portable applications, use major status values for handling errors. Use minor status values to debug applications and to display error and error-recovery information to users. The **gss_display_status()** routine is used to obtain printable text strings for major and minor status values.

Major status values

GSS-API routines return GSS status codes as their **OM_uint32** function value. These codes indicate generic API errors and are common across GSS-API implementations. A GSS status code indicates a single API error from the routine and a single calling error. Additional status information can be contained in the GSS status code as supplementary information. The errors are encoded into a 32-bit GSS status code as follows:

MSB   LSB

Calling Error	Routine Error	Supplementary Information

Bit 31   24 23   16 15   0
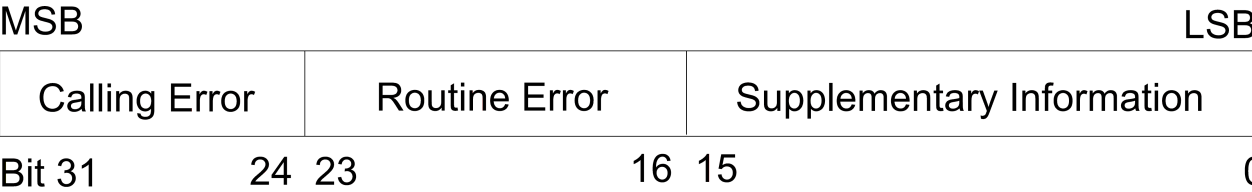
Figure 1. GSS status code bit locations

If a GSS-API routine returns a GSS status code whose upper 16 bits contain a nonzero value, the call failed. If the calling error field is nonzero, the application's call of the routine was in error. In addition, the routine can indicate additional information by setting one or more bits in the supplementary information field of the status code.

For reference information on GSS-API calling errors, routing errors, and supplementary status bits and their meanings, see *z/OS Integrated Security Services Network Authentication Service Administration*.

All **GSS_S_ symbols** equate to complete **OM_uint32** status codes rather than to bit field values.

The major status code GSS_S_FAILURE indicates that an error was detected that has no major status code. Check the minor status code for details about the error.

The GSS-API provides three macros for manipulating major status values:

- GSS_CALLING_ERROR()
- GSS_ROUTINE_ERROR()
- GSS_SUPPLEMENTARY_INFO()

Each macro takes a GSS status code and masks all but the relevant field. For example, when you use the GSS_ROUTINE_ERROR() macro on a status code, it returns a value. The value of the macro is arrived at by using only the routine errors field and zeroing the values of the calling error and supplementary information fields.

An additional macro, GSS_ERROR(), lets you determine whether the status code indicates a calling or routine error. If the status code indicates a calling or routine error, the macro returns a nonzero value. If no calling or routine error is indicated, the macro returns zero.

Note that an inaccessible read or write error may not be returned. Instead, a signal may be generated as a result of the attempt to access the storage location.

Minor status values

The GSS-API routines return a *minor_status* parameter to indicate errors from either the GSS-API interface layer or the underlying security mechanism layer. The parameter contains a single error, indicated by an **OM_uint32** value. For the Kerberos mechanism, this value is equivalent to the Kerberos **krb5_error_code** data type and contains a Kerberos return code. The **gss_display_status()** routine is used to generate a displayable message describing the minor status code.

Data types

Integer

The GSS-API defines the integer data type:

```
OM_uint32        32-bit unsigned integer
```

This integer data type is a portable data type that the GSS-API routine definitions use for guaranteed minimum bit counts.

String

Many of the GSS-API routines take arguments and return values that describe contiguous multiple-byte data, such as opaque data and character strings. Use the **gss_buffer_t** data type, which is a pointer to the **gss_buffer_desc** buffer descriptor, to pass the data between the GSS-API routines and the application.

The **gss_buffer_t** data type has this definition:

```
typedef struct gss_buffer_desc_struct {
    size_t                                length;
    void *                                value;
} gss_buffer_desc, *gss_buffer_t;
```

The length field contains the total number of bytes in the data. The value field contains a pointer to the actual data.

When using the **gss_buffer_t** data type, the GSS-API routine allocates storage for any data it passes to the application. The calling application is responsible for allocating the **gss_buffer_desc** object. It initializes **gss_buffer_desc** objects with the value GSS_C_EMPTY_BUFFER. To free the storage allocated by a GSS-API routine, the application calls the **gss_release_buffer()** routine. Since the GSS-API routine may use different storage management algorithms, the application should never attempt to release storage allocated by a GSS-API routine by any other means.

Object identifier

Applications use the **gss_oid** data type to specify a security mechanism and to specify name types.

Select a security mechanism by using the following object identifier (OID):

- For the Kerberos security mechanism, specify **gss_mech_krb5**. This corresponds to object identifier {1 2 840 113554 1 2 2}. The Kerberos mechanism is used when the initiator will use a Kerberos service ticket for authentication. For backward compatability, you can specify **gss_mech_krb5_old** which corresponds to object identifier {1 3 5 1 5 2}. **gss_mech_krb5_old** is only valid with DES and DES3 session keys.

- For SPKM (Simple Public Key Mechanism), specify gss_mech_spkm3. This corresponds to object identifier {1 3 6 1 5 5 1 3}. The SPKM mechanism is used when the initiator will use an X.509 certificate for authentication.

- For LIPKEY (Low Infrastructure Public Key Mechanism), specify gss_mech_lipkey. This corresponds to object identifier {1 3 6 1 5 5 9}. The LIPKEY mechanism is used when the initiator will use a userid and password for authentication.

Select a name type by using the following OIDs:

- For a name, specify GSS_C_NT_USER_NAME. This corresponds to object identifier {1 2 840 113554 1 2 1 1}.
- For the Kerberos mechanism, the user name is the character string representation of a Kerberos principal and is either the fully-qualified *principal@realm* or the unqualified *principal*. The local realm will be added if an unqualified principal name is specified.
- For the SPKM mechanism, the user name is either the distinguished name for the user or just the common name component. A name is assumed to be a distinguished name if it contains an '=' character, otherwise it is assumed to be the common name component. For example, "CN=John Doe,O=IBM,C=US" is a distinguished name while "John Doe" is the common name component.
- For the LIPKEY mechanism, the user name is interpreted differently depending upon whether it is a source name or a target name. A target name is handled as described for the SPKM mechanism. A source name must be a name acceptable as a system userid on the target system.
- For a service, specify GSS_C_NT_HOSTBASED_SERVICE. This corresponds to object identifier {1 2 840 113554 1 2 1 4}. For the Kerberos mechanism, a service is a character string that is fully-qualified (*service@host*) or unqualified (*service*). The local host name will be added if an unqualified service name is specified.
- For the Kerberos mechanism, the service name is converted to *service/canonical-name@kerberos-realm*. The canonical-name is obtained by doing a DNS lookup for the supplied host name and obtaining the canonical host name from the name server.
- For the SPKM and LIPKEY mechanisms, the service name is converted to 'service/host' and used as the common name component for the server providing the service. Note that the supplied host name is used without conversion to a canonical host name.
- For a Kerberos principal name, specify **gss_nt_krb5_name**. This name type is supported only by the Kerberos mechanism and corresponds to object identifier {1 2 840 113554 1 2 2 1}. This is the same as **GSS_C_NT_USER_NAME** except internal name representations are not created for the SPKM and LIPKEY mechanisms.
- For a principal structure created by the **krb5_parse_name()** routine, specify **gss_nt_krb5_principal**. This name type is supported only by the Kerberos mechanism and corresponds to object identifier {1 2 840 113554 1 2 2 2}.
- For a user identifier, specify GSS_C_NT_STRING_UID_NAME for the string representation of the **uid** or GSS_C_NT_MACHINE_UID_NAME for the binary representation of the **uid**. These correspond to object identifiers {1 2 840 113554 1 2 1 3} and {1 2 840 113554 1 2 1 2}. The **uid** will be mapped to a host userid on the local system. For the Kerberos mechanism, the userid will then be further mapped to a Kerberos principal. For the SPKM and LIPKEY mechanisms, the host userid becomes the user name.

The **gss_OID** data type contains tree-structured values defined by ISO and has the following definition:

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

The elements field of the structure points to the first byte of an octet string containing the ASN.1 BER (Basic Encoding Rules) encoding of the value of the **gss_OID** data type. The length field contains the number of bytes in the value.

The **gss_OID_desc** values returned by GSS-API routines are read-only values. The application should not attempt to release them by calling the **gss_release_oid()** function.

Object identifier sets

The **gss_OID_set** data type represents one or more object identifiers. The values of the **gss_OID_set** data type are used to:

- Report the available mechanisms supported by GSS-API
- Request specific mechanisms
- Indicate the mechanisms supported by a GSS-API credential
- Report the available name types supported by GSS-API.

The **gss_OID_set** data type is defined:

```
typedef struct gss_OID_set_desc_struct {
    int                                 count;
    gss_OID                             elements;
} gss_OID_set_desc, *gss_OID_set;
```

The *count* field contains the number of OIDs in the set. The *elements* field is a pointer to an array of **gss_oid_desc** objects, each describing a single OID. The application calls the **gss_release_oid_set()** routine to release the storage associated with **gss_OID_set** values that are returned by GSS-API routines.

Credentials

Credentials establish, or prove, the identity of an application or other principal. The gss_cred_id_t is an atomic data type that identifies a GSS-API credential data structure. The data type is opaque to the caller. The credential identifier is valid only within the process that acquired the credential.

Contexts

The security context is a pair of GSS-API data structures that contain information shared between the communicating applications. The information describes the cryptographic state of each application. This security context is required for per-message security services and is created by a successful authentication exchange. The gss_ctx_id_t data type contains an atomic value that identifies one end of a GSS-API security context. The data type is opaque to the caller. The context identifier is valid only within the process that initialized or accepted the security context.

Tokens

GSS-API uses tokens to maintain the synchronization between the communicating applications sharing a security context. The token is a cryptographically-protected octet string. The string is generated by the underlying security mechanism at one end of the GSS-API security context for use by the peer application at the other end of the security context. The data type is opaque to the caller. The caller uses the **gss_buffer_t** data type as tokens to GSS-API routines.

GSS-API uses two types of tokens. Context-level tokens are used to establish the security context between the communicating applications. Per-message tokens are used to provide integrity and confidentiality services for messages exchanged by the applications.

Names

Names identify principals. The GSS-API authenticates the relationship between a name and the principal claiming the name.

Names are represented in two forms:

- A printable form, for presentation to an application
- An internal, canonical form that is used by the GSS-API and is opaque to applications.

The **gss_import_name()** and **gss_display_name()** routines convert names between their printable and internal forms. Each security mechanism has its own name format. The **gss_import_name()** routine creates internal representations of the supplied name for use by each of the supported security

mechanisms. Internal names created by a specific security mechanism contain internal representations for just that security mechanism. The **gss_compare_name()** routine can be used to compare two names in their internal format.

Channel bindings

You can define and use channel bindings to associate the security context with the communications channel that carries the context. Channel bindings are communicated to the GSS-API by using the following structure:

```
typedef struct gss_channel_binding_struct {
    OM_uint32                                   initiator_addrtype;
    gss_buffer_desc                             initiator_address;
    OM_uint32                                   acceptor_addrtype;
    gss_buffer_desc                             acceptor_address;
    gss_buffer_desc                             application_data;
} gss_channel_bindings_desc, *gss_channel_bindings_t;
```

Use the *initiator_addrtype* and *acceptor_addrtype* fields to indicate the type of addresses contained in the *initiator_address* and *acceptor_address* buffers. The following table lists the address types and their address type values:

Table 30. Channel bindings address types	
Address Type	**Values**
GSS_C_AF_UNSPEC	Unspecified
GSS_C_AF_LOCAL	Host local address
GSS_C_AF_INET	DARPA Version 4 internet address (IPv4).
GSS_C_AF_IMPLINK	ARPAnet IMP
GSS_C_AF_PUP	pup protocols (for example, BSP)
GSS_C_AF_CHAOS	MIT CHAOS protocol
GSS_C_AF_NS	XEROX NS
GSS_C_AF_NBS	nbs
GSS_C_AF_ECMA	ECMA
GSS_C_AF_DATAKIT	datakit protocols
GSS_C_AF_CCITT	CCITT protocols (for example, X.25)
GSS_C_AF_SNA	IBM SNA
GSS_C_AF_DECnet	Digital DECnet
GSS_C_AF_DLI	Direct data link interface
GSS_C_AF_LAT	LAT
GSS_C_AF_HYLINK	NSC Hyperchannel
GSS_C_AF_APPLETALK	AppleTalk
GSS_C_AF_BSC	BISYNC 2780/3780
GSS_C_AF_DSS	Distributed system services
GSS_C_AF_OSI	OSI TP4
GSS_C_AF_X25	X25
GSS_C_AF_INET6	DARPA Version 6 internet address (IPv6)

Table 30. Channel bindings address types (continued)	
Address Type	**Values**
GSS_C_AF_NULLADDR	No address specified

The tags specify address families rather than addressing formats. For address families that contain several alternative address forms, the *initiator_address* and *acceptor_address* fields should contain sufficient information to determine which address form is being used. Format the bytes that contain the addresses in the order the bytes are transmitted across the network.

The GSS-API creates an octet string by concatenating all of the fields in the gss_channel_bindings_desc data structure. The security mechanism signs the octet string and binds the signature to the token generated by the gss_init_sec_context() routine. The context acceptor presents the same bindings to the gss_accept_sec_context() routine, which generates its own signature and compares it to the signature in the token. If the signatures differ, the gss_accept_sec_context() routine returns a GSS_S_BAD_BINDINGS error and the context is not established.

Some security mechanisms check that the *initiator_address* field of the channel bindings presented to the gss_init_sec_context() routine contains the correct network address of the local system. Therefore, portable applications should use either the correct address type and value or specify GSS_C_AF_NULLADDR for the *initiator_addrtype* field. Some security mechanisms include the channel binding data in the token instead of a signature, so portable applications should not use confidential data as channel binding components. The Kerberos GSS-API does not verify the address or include the plain text binding information in the token.

Optional parameters

In some of the routine descriptions, optional parameters allow the application to request default behavior by passing a default value for a parameter. The conventions shown in the table are used for optional parameters:

Table 31. GSS-API optional parameters	
Data Types	
gss_buffer_t data types	GSS_C_NO_BUFFER
Output integer data types	NULL
OID data types	GSS_C_NO_OID
OID set data types	GSS_C_NO_OID_SET
Credential data types	GSS_C_NO_CREDENTIAL
Context data types	GSS_C_NO_CONTEXT
Channel binding data types	GSS_C_NO_CHANNEL_BINDINGS
Name data types	GSS_C_NO_NAME
Empty buffer descriptor initialization	GSS_C_EMPTY_BUFFER

GSS-API version compatibility

Some of the type definitions used by GSS-API function prototypes have changed between Version 1 and Version 2 of the GSS-API specifications (Internet RFC 2744). The default definitions are those defined by Version 2 of the specifications. You can use the Version 1 definitions by defining the GSSAPI_V1_COMPAT compiler variable when compiling your source code.

The following function names have changed between GSS-API Version 1 and GSS-API Version 2. The original function names are still supported for compatibility with applications written to the GSS-API Version 1 specifications.

- The **gssapi_sign()** routine is now the **gssapi_get_mic()** routine
- The **gssapi_verify()** routine is now the **gssapi_verify_mic()** routine
- The **gssapi_seal()** routine is now the **gssapi_wrap()** routine
- The **gssapi_unseal()** routine is now the **gssapi_unwrap()** routine.

Interoperability with Microsoft Windows 2000 SSPI

A GSS-API application can communicate with a Microsoft Windows 2000 SSPI application using the Kerberos security mechanism.

Creating the security context

The InitializeSecurityContext() function is used to create the SSPI security context. The ISC_REQ_MUTUAL_AUTH, ISC_REQ_REPLAY_DETECT, ISC_REQ_SEQUENCE_DETECT, ISC_REQ_INTEGRITY, and ISC_REQ_CONFIDENTIALITY flags are used to specify the context attributes. The gss_accept_sec_context() function is then used to accept the security context on the remote partner. Since channel bindings are not supported by SSPI, you must specify GSS_C_NO_CHANNEL_BINDINGS on the gss_accept_sec_context() function call.

Accepting the security context

The AcceptSecurityContext() function is used to accept a GSS-API security context created by the gss_init_sec_context() function. Since channel bindings are not supported by SSPI, you must specify GSS_C_NO_CHANNEL_BINDINGS on the gss_init_sec_context() function call.

Message signature

The **MakeSignature()** function is used to sign a message and the **VerifySignature()** function is used to verify a signature. The **gss_get_mic()** and **gss_verify_mic()** functions are the corresponding GSS-API functions.

Message encryption

The **EncryptMessage()** function is used to encrypt a message and the **DecryptMessage()** function is used to decrypt a message. The **gss_wrap()** and **gss_unwrap()** functions are the corresponding GSS-API functions.

Message sequence numbers

The application is responsible for supplying the proper message sequence number when processing a message with the SSPI message functions. The first message is always message 0 and the sequence number is incremented for each successive message. The sequence numbers for sent messages are separate from the sequence numbers for received messages.

Chapter 5. GSS-API programming interfaces

This chapter lists the GSS-API programming interfaces in alphabetical order and provides information about the purpose, format, parameters, use, and status codes of each.

gss_accept_sec_context (accept a security context)

Purpose

Accepts a security context created by the context initiator.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_accept_sec_context (
    OM_uint32 *             minor_status,
    gss_ctx_id_t *          context_handle,
    gss_cred_id_t           acceptor_cred_handle,
    gss_buffer_t            input_token,
    gss_channel_bindings_t  input_chan_bindings,
    gss_name_t *            src_name,
    gss_OID *               mech_type,
    gss_buffer_t            output_token,
    gss_flags_t *           ret_flags,
    OM_uint32 *             time_rec,
    gss_cred_id_t *         delegated_cred_handle)
```

Parameters

Input

acceptor_cred_handle
Specifies the GSS-API credential for the identity claimed by the context acceptor. The credential must be either an ACCEPT type credential or a BOTH type credential.

input_token
Specifies the token received from the context initiator.

input_chan_bindings
Specifies the bindings describing the communications channel used between the communicating applications. The channel bindings specified by the context acceptor must match the bindings that were specified by the context initiator when the input token was created. Specify GSS_C_NO_CHANNEL_BINDINGS if there are no channel bindings.

Input/Output

context_handle
Specifies a context handle for the context. The first time that the context acceptor calls the **gss_accept_sec_context()** routine, the context handle value must be set to GSS_C_NO_CONTEXT. For subsequent calls to continue setting up the context, the context handle must be the value returned by the previous call to the **gss_accept_sec_context()** routine.

Output

src_name
Returns the authenticated name of the context initiator. If the authenticated name is not required, specify NULL for this parameter. The returned name is an anonymous internal name if the GSS_C_ANON_FLAG is set in the returned flags. The application should release the name when it is no longer needed by calling the **gss_release_name()** routine.

mech_type
Returns the security mechanism with which the context was established. If the security mechanism type is not required, specify NULL for this parameter. The **gss_OID** value returned for this parameter points to a read-only structure and must not be released by the application. The returned security mechanism will be one of the following:

- **gss_mech_krb5_old** - Beta Kerberos V5 mechanism
- **gss_mech_krb5** - Kerberos V5 mechanism
- **gss_mech_spkm3** - Low infrastructure version of the simple public key mechanism (SPKM)
- **gss_mech_lipkey** - Low infrastructure public key mechanism (LIPKEY)

output_token
Returns a token to be returned to the context initiator. If no token is to be passed to the context initiator, the **gss_accept_sec_context()** routine sets the *output_token* length field to zero. Otherwise, the *output_token* length and value fields are set to nonzero values. The application should release the output token when it is no longer needed by calling the **gss_release_buffer()** routine.

ret_flags
Returns a bitmask containing independent flags representing services that the initiating application has requested. Specify NULL for this parameter if the flag values are not required. The following symbolic definitions are provided to test the individual flags and should be logically ANDed with the value of *ret_flags* to test whether the context supports the service option.

- GSS_C_DELEG_FLAG - Delegated credentials are available if this flag is TRUE
- GSS_C_MUTUAL_FLAG - Mutual authentication is required if this flag is TRUE
- GSS_C_REPLAY_FLAG - Replayed signed or sealed messages will be detected if this flag is TRUE
- GSS_C_SEQUENCE_FLAG - Out-of-sequence signed or sealed messages will be detected if this flag is TRUE
- GSS_C_CONF_FLAG - Confidentiality services are available if this flag is TRUE
- GSS_C_INTEG_FLAG - Integrity services are available if this flag is TRUE
- GSS_C_ANON_FLAG - Anonymous services are available if this flag is TRUE. The *src_name* parameter returns an anonymous internal name
- GSS_C_PROT_READY_FLAG - Protection services, as specified by the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG, are available if the accompanying major status is GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED. Otherwise, protection services are available only if the accompanying major status is GSS_S_COMPLETE.
- GSS_C_TRANS_FLAG - If this flag is set, the **gss_export_sec_context()** function can be used to export the security context. The **gss_export_sec_context()** function is not available if this flag is not set.

time_rec
Returns the number of seconds remaining before the context is no longer valid. If the mechanism does not support credential expiration, the return value is GSS_C_INDEFINITE. Specify NULL for this parameter if the remaining time is not required.

delegated_cred_handle
Returns the credential handle for delegated credentials received from the context initiator. Specify NULL for this parameter if the delegated credentials are not required. A credential handle is returned only if the GSS_C_DELEG_FLAG flag is set in the return flags. The returned credential can then be used to initiate a new security context by calling the **gss_init_sec_context()** routine. The returned credential should be released when it is no longer needed by calling the **gss_release_cred()** routine.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_accept_sec_context()** routine is the second step in establishing a security context between the context initiator and the context acceptor. In the first step, the context initiator calls the **gss_init_sec_context()** routine, which returns a token for the security context. The context initiator then passes this security token to the context acceptor. In the second step, the context acceptor takes the token supplied by the context initiator and calls the **gss_accept_sec_context()** routine to accept the context.

If the Kerberos security server is running on the same system as the application, it is not necessary to provide a key table. Instead, the GSS-API uses the local instance of the Kerberos security server to decrypt the ticket. In order to activate this support, the KRB5_SERVER_KEYTAB environment variable needs to be set to one of the following values and depending on the value set, the following requirements must also be met:

1. If the KRB5_SERVER_KEYTAB environment variable is set to 1:

   a. The application must be running with a user or group that has at least READ access to the IRR.RUSERMAP resource in the FACILITY class.

   b. The Kerberos principal associated with the current system identity must match the principal for the GSS-API credential.

2. If the KRB5_SERVER_KEYTAB environment variable is set to 2:

   a. The current system identity must have an associated Kerberos principal that matches the server principal in the ticket or have at least READ access in the KERBLINK class to the server principal in the ticket.

If the length value in the output_token is not zero, the context acceptor must pass the returned token to the context initiator. The context initiator must then call **gss_init_sec_context()** and specify the context identifier returned by the original call to **gss_init_sec_context()** as well as the output token that was returned by the context acceptor.

To complete the context establishment, one or more reply tokens may be required from the peer application. If so, **gss_accept_sec_context()** returns a status flag of GSS_S_CONTINUE_NEEDED, in which case it should be called again when the reply token is received from the peer application, passing the token to **gss_accept_sec_context()** through the *input_token* parameter.

The availability of confidentiality services depends on the underlying security mechanism and the features that have been installed on the system. The GSS_C_CONF_FLAG is returned only if confidentiality services are available on both the local and remote systems. If confidentiality services are available on the remote system but not on the local system, an error is returned by the **gss_unwrap()** routine if an encrypted message is received (that is, confidentiality was requested on the call to the **gss_wrap()** routine on the remote system).

Whenever the GSS_S_CONTINUE_NEEDED status flag is set, the context is not fully established and the following restrictions apply to the output parameters:

- The value that the *time_rec* parameter returns is undefined.

- Unless the accompanying ret_flags parameter contains the bit GSS_C_PROT_READY_FLAG, indicating that per-message services may be applied in advance of a successful completion status, the value returned by the mech_type parameter may be undefined until the routine returns a major status of GSS_S_COMPLETE.

- The values of the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG, and GSS_C_ANON_FLAG bits returned through the *ret_flags* parameter contain the values that the implementation expects to be valid if context establishment is to succeed.

- The value of the GSS_C_PROT_READY_FLAG bit returned through the *ret_flags* parameter indicates the actual state at the time **gss_accept_sec_context()** returns, whether or not the context is fully established.

Kerberos Mechanism

The **gss_accept_sec_context()** routine needs a key to decrypt the token provided by the context initiator. The token contains the unencrypted principal name of the context acceptor. This name identifies the key that the context initiator used to encrypt the token. The default key table is used to obtain the key for the indicated principal. The KRB5_KTNAME environment variable can be set to use a different key table.

The context expiration time is obtained from the service ticket that was obtained by the context initiator as part of the **gss_init_sec_context()** processing.

When delegation is used, the forwarded Kerberos credentials are stored in a new Kerberos credentials cache that is associated with the GSS-API credential returned for the *delegated_cred_handle* parameter. This GSS-API credential can then be used to initiate new security contexts on behalf of the original context initiator.

SPKM mechanism

The **gss_accept_sec_context()** routine needs an X.509 certificate and associated private key in order to accept the token provided by the context initiator. The certificate will be obtained from the supplied GSS-API credential. If no credential is provided, the default certificate for the application will be used.

The target name in the input token can be a distinguished name or the common name (CN) component of a distinguished name. See "Object identifier" on page 187 for more details on distinguished names and common names. The target name is verified against the target certificate as follows:

- If the target name is a distinguished name, it must match either of the following in the target certificate:
  - the subject name
  - a Data Name (DN) value of the subject alternate name.
- Otherwise, if the target name is a common name (which may be in the form *service-name/host-name*), one of the following checks must be satisfied against the target certificate:
  - a common name (CN) component of the certificate subject name matches the target name, or the *host-name* component of the target name
  - a CN component of a DN value of the certificate subject alternative name matches the target name, or the *host-name* component of the target name
  - a DNS value in the certificate subject alternate name extension matches the *host-name* component of the target name.

Diffie-Hellman key agreement is used to compute the secret value required by the key generation process. This is a two-pass algorithm requiring inputs from both the initiator and the acceptor. Mutual authentication is required if the initiator does not provide its Diffie-Hellman public value in the initial output token returned by the **gss_init_sec_context()** routine. Mutual authentication is optional if the initiator does provide its Diffie-Hellman public value in the initial token (Diffie-Hellman key agreement is the default key establishment algorithm for the context).

LIPKEY mechanism

The **gss_accept_sec_context()** routine needs an X.509 certificate and associated private key in order to accept the token provided by the context initiator. The certificate will be obtained from the supplied GSS-API credential. If no credential is provided, the default certificate for the application will be used.

The target name in the input token can be a distinguished name or the common name (CN) component of a distinguished name. See "Object identifier" on page 187 for more details on distinguished names and common names. The target name is verified against the target certificate as follows:

- If the target name is a distinguished name, it must match either of the following in the target certificate:
  - the subject name
  - a Data Name (DN) value of the subject alternate name.
- Otherwise, if the target name is a common name (which may be in the form *service-name/host-name*), one of the following checks must be satisfied against the target certificate:

– a common name (CN) component of the certificate subject name matches the target name, or the *host-name* component of the target name

– a CN component of a DN value of the certificate subject alternative name matches the target name, or the *host-name* component of the target name

– a DNS value in the certificate subject alternate name extension matches the *host-name* component of the target name.

The **__passwd()** system routine is called to validate the user name and password supplied by the context initiator. If the BPX.DAEMON facility class profile is defined, then the system userid associated with the context acceptor application must have at least READ access to the BPX.DAEMON class profile and all modules within the address space must be loaded from controlled libraries. This includes all modules in the application and run-time libraries.

The z/OS Network Authentication Service and System SSL load modules are located in SYS1.SIEALNKE, and the C/C++ runtime load modules are located in CEE.SCEERUN and CEE.SCEERUN2. The **extattr** command with the **+p** option can be used to define programs in UNIX files to program control. Refer to *z/OS UNIX System Services Planning* for more information on setting up a program-controlled runtime environment.

Key database usage

The SPKM and LIPKEY mechanisms use X.509 certificates. These certificates and associated certification authority certificates are obtained from a key database or SAF key ring. The **GSS_KEYRING_NAME** environment variable specifies the name of the key database or SAF key ring. The **GSS_KEYRING_PW** or **GSS_KEYRING_STASH** environment variable specifies the password for the key database (**GSS_KEYRING_STASH** is ignored if **GSS_KEYRING_PW** is defined). A SAF key ring is used if neither **GSS_KEYRING_PW** nor **GSS_KEYRING_STASH** is defined. The **GSS_KEY_LABEL** environment variable specifies the label of the default certificate for the application. The default certificate for the key database or SAF key ring will be used if this variable is not defined.

Status Codes

Table 32. Status Codes for *gss_accept_sec_context()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_BINDINGS	The *input_token* parameter contains different channel bindings from those specified with the *input_chan_bindings* parameter.
GSS_S_BAD_MECH	The security mechanism used by the context initiator is not available on the acceptor system.
GSS_S_BAD_SIG	The received input token contains an incorrect signature.
GSS_S_CONTINUE_NEEDED	Control information in the returned output token must be sent to the initiator and a response must be received and passed as the *input_token* argument to a continuation call to the **gss_accept_sec_context()** routine.
GSS_S_CREDENTIALS_EXPIRED	Credentials are no longer valid.
GSS_S_DEFECTIVE_CREDENTIAL	Consistency checks performed on the credential structure referenced by the *verifier_cred_handle* parameter failed.
GSS_S_DEFECTIVE_TOKEN	Consistency checks performed on the input token failed.

*Table 32. Status Codes for **gss_accept_sec_context()** (continued)*

Status Code	Meaning
GSS_S_DUPLICATE_TOKEN	The token is a duplicate of a token that has already been processed. This is a fatal error during context establishment.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The context identifier provided by the caller does not refer to a valid security context.
GSS_S_NO_CRED	No credentials are available or the credentials are valid for context initiation use only.
GSS_S_OLD_TOKEN	The token is too old to be checked for duplication against previous tokens. This is a fatal error during context establishment.

gss_acquire_cred (acquire a GSS-API credential)

Purpose

Allows an application to acquire a GSS-API credential.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_acquire_cred (
    OM_uint32 *           minor_status,
    gss_name_t            desired_name,
    OM_uint32             time_req,
    gss_OID_set           desired_mechs,
    gss_cred_usage_t      cred_usage,
    gss_cred_id_t *       output_cred_handle,
    gss_OID_set *         actual_mechs,
    OM_uint32 *           time_rec)
```

Parameters

Input

desired_name
Specifies the principal name to be used for the credential. Specify GSS_C_NO_NAME for this parameter to use the name obtained from the default credentials cache.

time_req
Specifies the number of seconds that the credential remains valid. Specify GSS_C_INDEFINITE to request the maximum credential lifetime. Specify zero for the default lifetime of 2 hours. For the Kerberos mechanism, the actual credential lifetime will be limited by the lifetime of the underlying ticket-granting ticket for GSS_C_INITIATE and GSS_C_BOTH credentials. For the SPKM and LIPKEY mechanisms, the actual credential lifetime will be limited by the expiration date of the underlying X.509 certificate

desired_mechs
Specifies the desired security mechanisms for use with the credential. Mechanisms that are not available on the local system are ignored. The actual mechanisms that can be used with the credential

are returned in the *actual_mechs* parameter. Specify GSS_C_NO_OID_SET for this parameter to use the default mechanism of gss_mech_krb5.

The following security mechanisms are supported:

- gss_mech_krb5_old - Beta Kerberos V5 mechanism. The source and target are authenticated using a Kerberos ticket. This mechanism is deprecated and should not be used by new applications. It is only valid with DES and DES3 session keys.
- gss_mech_krb5- Kerberos V5 mechanism. The source and target are authenticated using a Kerberos ticket.
- gss_mech_spkm3 - Low infrastructure version of the simple public key mechanism (SPKM). The source and target are authenticated using X.509 certificates.
- gss_mech_lipkey - Low infrastructure public key mechanism (LIPKEY). The source is authenticated using a userid and password. The target is authenticated using an X.509 certificate.

cred_usage
Specifies the desired credential usage as follows:

- GSS_C_INITIATE if the credential can be used only to initiate security contexts
- GSS_C_ACCEPT if the credential can be used only to accept security contexts
- GSS_C_BOTH if the credential can be used to both initiate and accept security contexts.

Output

output_cred_handle
Returns the handle for the GSS-API credential.

actual_mechs
Returns the set of mechanism identifiers the credential is valid for. If the actual mechanisms are not required, specify NULL for this parameter. The gss_OID_setreturned for this parameter should be released by calling the gss_release_oid_set() routine when it is no longer needed.

time_rec
Returns the number of seconds the credential remains valid. If the time remaining is not required, specify NULL for this parameter.

minor_status
Returns a status code from the security mechanism.

Usage

The gss_acquire_cred() routine allows an application to obtain a GSS-API credential. The application can then use the credential with the gss_init_sec_context() and gss_accept_sec_context() routines.

Kerberos mechanism

If GSS_C_INITIATE or GSS_C_BOTH is specified for the credential usage, the application must have a valid ticket in the default credentials cache and the ticket must not expire for at least 10 minutes. The gss_acquire_cred() routine will use the first valid ticket-granting ticket (or the first valid service ticket if there is no TGT) to create the GSS-API credential. The principal specified by the *desired_name* parameter must match the principal obtained from the credentials cache or must be specified as GSS_C_NO_NAME. The KRB5CCNAME environment variable is used to identify the credentials cache to be used.

If GSS_C_ACCEPT or GSS_C_BOTH is specified for the credential usage, the principal specified by the *desired_name* parameter must be defined in a key table. The KRB5_KTNAME environment variable is used to identify the key table to be used. If the Kerberos security server is running on the same system as the application, it is not necessary to provide a key table for GSS_C_ACCEPT or GSS_C_BOTH credentials. Instead, GSSAPI uses the local instance of the Kerberos security server to decrypt the ticket. In order to activate this support, the KRB5_SERVER_KEYTAB environment variable needs to be set to one of the following values and, depending on the value set, the following other requirements must also be met:

1. If the KRB5_SERVER_KEYTAB environment variable is set to 1:

    a. The application must be running with a user or group that has at least READ access to the IRR.RUSERMAP resource in the FACILITY class.

    b. The Kerberos principal associated with the current system identity must match the principal for the GSSAPI credential.

2. If the KRB5_SERVER_KEYTAB environment variable is set to 2:

    a. No requirements – processing is done during gss_accept_sec_context() call.

All credentials created by the Kerberos mechanism can be used with the gss_mech_krb5 security mechanism identifier whereas only some credentials can be used with the gss_mech_krb5_old security mechanism identifier. If the *desired_mechs* parameter was set to GSS_C_NO_OID then the mechanism set returned for the *actual_mechs* parameter will contain gss_mech_krb5, otherwise it will contain what was specified in the *desired_mechs* parameter.

SPKM mechanism

The application must have a key database or a SAF key ring containing the application certificate and associated private key. The default certificate for the application will be used if GSS_C_NO_NAME is specified for the *desired_name* parameter, and GSS_KEY_LABEL is not specified. See *z/OS Integrated Security Services Network Authentication Service Administration* for more details on GSS_KEY_LABEL. Otherwise, a certificate with a subject name matching the desired name must be found in the key database or SAF key ring. An error will be returned if multiple certificates are found with matching subject name values.

The desired name can be a distinguished name or just the common name component of a distinguished name. For example, 'CN=John Doe,O=IBM,C=US' is a distinguished name while 'John Doe' is the common name component. An exact match with the certificate subject name is required when a distinguished name is supplied while just the common names must match when a common name component is supplied. Refer to RFC 2253 (UTF-8 String Representation of Distinguished Names) for more information on the string representation of a distinguished name. Refer to the description of the gsk_dn_to_name() routine in *z/OS System SSL Programming* for more information on how a distinguished name is converted to an X.509 name.

LIPKEY mechanism

If GSS_C_INITIATE or GSS_C_BOTH is specified for the credential usage, the user will be prompted for the password associated with the desired name. The default certificate for the application will be used if GSS_C_NO_NAME is specified for the *desired_name* parameter, and GSS_KEY_LABEL is not specified. Seeee *z/OS Integrated Security Services Network Authentication Service Administration* for more details on GSS_KEY_LABEL. The desired name for GSS_C_INITIATE or GSS_C_BOTH must be a character string usable on the target system as a user name. In the case of GSS_C_BOTH, the desired name must also be usable as a search argument in order to locate the X.509 certificate in the key database or SAF key ring.

If GSS_C_ACCEPT or GSS_C_BOTH is specified for the credential usage, the application must have a key database or a SAF key ring containing the application certificate and associated private key. The default certificate for the application will be used if GSS_C_NO_NAME is specified for the desired_name parameter. Otherwise, a certificate with a subject name matching the desired name must be found in the key database or SAF key ring. An error will be returned if multiple certificates are found with matching subject name values.

The desired name for GSS_C_ACCEPT can be a distinguished name or just the common name component of a distinguished name. For example, 'CN=John Doe,O=IBM,C=US' is a distinguished name while 'John Doe' is the common name component. An exact match with the certificate subject name is required when a distinguished name is supplied while just the common names must match when a common name component is supplied. Refer to RFC 2253 (UTF-8 String Representation of Distinguished Names) for more information on the string representation of a distinguished name. Refer to the description of the gsk_dn_to_name() routine in *z/OS System SSL Programming* for more information on how a distinguished name is converted to an X.509 name.

Key database usage

The SPKM and LIPKEY mechanisms use X.509 certificates. These application certificates and the associated certification authority certificates are obtained from a key database or SAF key ring. The GSS_KEYRING_NAME environment variable specifies the name of the key database or SAF key ring. The GSS_KEYRING_PW or GSS_KEYRING_STASH environment variable specifies the password for the key database (GSS_KEYRING_STASH is ignored if GSS_KEYRING_PW is defined). A SAF key ring is used if neither GSS_KEYRING_PW nor GSS_KEYRING_STASH is defined. The GSS_KEY_LABEL environment variable specifies the label of the default certificate for the application. The default certificate for the key database or SAF key ring will be used if this variable is not defined.

Status Codes

*Table 33. Status Codes for **gss_acquire_cred()***

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_MECH	None of the requested mechanisms are supported by the local system.
GSS_S_BAD_NAME	The name specified for the *desired_name* parameter is not valid.
GSS_S_BAD_NAMETYPE	The name specified for the *desired_name* parameter is not supported by the applicable underlying GSS-API mechanisms.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CRED	Default credentials are not available.

gss_add_cred (add a credential)

Purpose

Adds a credential element to an existing GSS-API credential.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_add_cred (
    OM_uint32 *          minor_status,
    gss_cred_id_t        input_cred_handle,
    gss_name_t           desired_name,
    gss_OID              mech_type,
    gss_cred_usage_t     cred_usage,
    OM_uint32            init_time_req,
    OM_uint32            accept_time_req,
    gss_cred_id_t *      output_cred_handle,
    gss_OID_set *        actual_mechs,
    OM_uint32 *          init_time_rec,
    OM_uint32 *          accept_time_rec)
```

Parameters

Input

input_cred_handle
Specifies the GSS-API credential to be modified. Specify GSS_C_NO_CREDENTIAL to modify the default GSS-API credential.

desired_name
Specifies the principal name to be used for the credential.

mech_type
Specifies the mechanism element to be added to the credential. The credential must not already contain an element for this mechanism. The supported security mechanisms are as follows:

- gss_mech_krb5_old - Beta Kerberos V5 mechanism. The source and target are authenticated using a Kerberos ticket. This mechanism is deprecated and should not be used by new applications. It is only valid with DES and DES3 session keys.
- gss_mech_krb5 - Kerberos V5 mechanism. The source and target are authenticated using a Kerberos ticket.
- gss_mech_spkm3 - Low infrastructure version of the simple public key mechanism (SPKM). The source and target are authenticated using X.509 certificates.
- gss_mech_lipkey - Low infrastructure public key mechanism (LIPKEY). The source is authenticated using a userid and password. The target is authenticated using an X.509 certificate.

cred_usage
Specifies the desired credential use as follows:

- GSS_C_INITIATE - The credential can be used only to initiate security contexts
- GSS_C_ACCEPT - The credential can be used only to accept security contexts
- GSS_C_BOTH - The credential can be used to both initiate and accept security contexts

init_time_req
Specifies the number of seconds the credential remains valid for initiating contexts. The z/OS Kerberos implementation of GSS-API does not support separate initiate and accept expiration times. The actual expiration time is the smaller of the initiate and accept times. Specify zero to request the default lifetime of 2 hours. Specify GSS_C_INDEFINITE to request the maximum lifetime.

accept_time_req
Specifies the number of seconds the credential remains valid for accepting contexts. The z/OS Kerberos implementation of GSS-API does not support separate initiate and accept expiration times. The actual expiration time is the smaller of the initiate and accept times. Specify zero to request the default lifetime of 2 hours. Specify GSS_C_INDEFINITE to request the maximum lifetime.

Output

output_cred_handle
Returns the credential handle for the updated credential. If NULL is specified for this parameter, the new credential element is added to the input credential. Otherwise, a new credential is created from the input credential and contains all of the credential elements of the input credential plus the new credential element. NULL may not be specified for this parameter if GSS_C_NO_CREDENTIAL is specified for the input credential.

actual_mechs
Returns the total set of mechanisms supported by the GSS-API credential. Specify NULL for this parameter if the actual mechanisms are not required. The **gss_OID_set** returned for this parameter should be released by calling the **gss_release_oid_set()** routine when it is no longer needed.

init_time_rec
Returns the initiate expiration time in seconds. Specify NULL for this parameter if the initiate time is not required.

accept_time_rec

Returns the accept expiration time in seconds. Specify NULL for this parameter if the accept time is not required.

minor_status

Returns a status code from the security mechanism.

Usage

The **gss_add_cred()** routine adds a new mechanism element to a GSS-API credential. The credential must not already contain an element for the mechanism. A GSS-API credential must contain an element for each mechanism that is used for contexts that are initiated or accepted using the credential.

The **gss_add_cred()** routine performs the same function as the **gss_acquire_cred()** routine does for a single mechanism.

Status Codes

Table 34. Status Codes for **gss_add_cred()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_MECH	The specified mechanism is not supported.
GSS_S_BAD_NAME	The supplied name is not valid.
GSS_S_BAD_NAMETYPE	The supplied name does not contain an internal representation for the requested mechanism.
GSS_S_DUPLICATE_ELEMENT	The credential already contains an element for the specified mechanism.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CRED	The referenced credential does not exist.

gss_add_oid_set_member (add to an OID set)

Purpose

Adds an OID to an OID set.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_add_oid_set_member (
    OM_uint32 *                minor_status,
    gss_OID                    input_oid,
    gss_OID_set *              oid_set)
```

Parameters

Input

input_oid

Specifies the OID you want to add to the OID set.

Input/Output

oid_set
> Specifies the OID set. The **gss_OID** array referred to by the elements field of the **gss_OID_set** is reallocated to hold the new OID. The application should call the **gss_release_oid_set()** routine to release the OID set when it is no longer needed.

Output

minor_status
> Returns a status code from the security mechanism.

Usage

The **gss_add_oid_set_member()** routine adds a new OID to an existing OID set. You can create an empty OID set by calling the **gss_create_empty_oid_set()** routine. The **gss_add_oid_set_member()** routine makes a copy of the input OID, so any future changes to the input OID have no effect on the copy in the OID set.

Status Codes

Table 35. Status Codes for *gss_add_oid_set_member()*

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_canonicalize_name (reduce to a mechanism name)

Purpose

Reduces a GSS-API internal name to a mechanism name.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_canonicalize_name (
    OM_uint32 *                minor_status,
    gss_name_t                 input_name,
    gss_OID                    mech_type,
    gss_name_t *               output_name)
```

Parameters

Input

input_name
> Specifies the name to be processed. An error is returned if GSS_C_NO_NAME is specified for this parameter.

mech_type
> Specifies the security mechanism to be used:
>
> - gss_mech_krb5_old - Beta Kerberos V5 mechanism. This mechanism is deprecated and should not be used by new applications. It is only valid with DES and DES3 session keys.
> - gss_mech_krb5 - Kerberos V5 mechanisms

- gss_mech_spkm3 - Low infrastructure version of the simple public key mechanism (SPKM)
- gss_mech_lipkey - Low infrastructure public key mechanism (LIPKEY)

Output

output_name
> Returns the mechanism name. The **gss_name_t** returned by this parameter should be released by calling the **gss_release_name()** function when it is no longer needed.

minor_status
> Returns a status code from the security mechanism.

Usage

The **gss_canonicalize_name()** routine takes a GSS-API internal name that contains multiple internal representations and returns a new GSS-API internal name with a single name representation that corresponds to the specified security mechanism. A name that represents a single security mechanism is called a *mechanism name*.

Status Codes

*Table 36. Status Codes for **gss_canonicalize_name()***

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_MECH	The requested mechanism is not supported.
GSS_S_BAD_NAME	The input name is not valid.
GSS_S_BAD_NAMETYPE	The input name does not contain an element for the requested mechanism.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_compare_name (compare two internal names)

Purpose

Allows an application to compare two internal names to determine if they refer to the same object.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_compare_name (
    OM_uint32 *            minor_status,
    gss_name_t             name1,
    gss_name_t             name2,
    int *                  name_equal)
```

Parameters

Input

name1
> Specifies the first internal name.

name2
> Specifies the second internal name.

Output

name_equal
> Returns 1 if the names refer to the same object and 0 otherwise.

minor_status
> Returns a status code from the security mechanism.

Usage

The **gss_compare_name()** routine lets an application compare two internal names to determine whether they refer to the same object. The two names must have an internal representation format in common in order to be comparable. The names are considered not equal if either name denotes an anonymous principal.

Status Codes

Table 37. Status Codes for **gss_compare_name()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_NAME	One of the input names is not valid.
GSS_S_BAD_NAMETYPE	The two name types cannot be compared. The names must have an internal representation in common in order to be comparable.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_context_time (return number of valid context seconds)

Purpose

Returns the number of seconds that the context remains valid.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_context_time (
    OM_uint32 *            minor_status,
    gss_ctx_id_t           context_handle,
    OM_uint32 *            time_rec)
```

Parameters

Input

context_handle
> Specifies the context to be checked.

Output

time_rec
    Returns the number of seconds that the context remains valid.

minor_status
    Returns a status code from the security mechanism.

Usage

The **gss_context_time()** routine checks the specified security context and returns the number of seconds that the context remains valid. The returned value is GSS_C_INDEFINITE if the context does not have an expiration time. The Kerberos security mechanism supports context expiration and returns the time remaining before the underlying service ticket expires, if the context was created by **gss_accept_sec_context()**, or the lesser of the requested expiration time and the ticket expiration time, if the context was created by **gss_init_sec_context()**.

Status Codes

Table 38. Status Codes for *gss_context_time()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_CONTEXT_EXPIRED	The referenced context has expired.
GSS_S_CREDENTIALS_EXPIRED	The credentials associated with the context referred to have expired.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The context referred to does not exist.

gss_create_empty_oid_set (create a new OID set)

Purpose

Creates a new, empty OID set.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_create_empty_oid_set (
    OM_uint32 *                      minor_status,
    gss_OID_set *                 oid_set)
```

Parameters

Output

oid_set
    Returns the OID set created by this routine. The application should call the **gss_release_oid_set()** routine when the OID set is no longer needed.

minor_status
    Returns a status code from the security mechanism.

Usage

The **gss_create_empty_oid_set()** routine creates a new, empty OID set. Members can be added to the OID set by calling the **gss_add_oid_set_member()** routine. The OID set should be released when it is no longer needed by calling the **gss_release_oid_set()** routine.

Status Codes

Table 39. Status Codes for *gss_create_empty_oid_set()*	
Status Code	**Meaning**
GSS_C_COMPLETE	The routine completed successfully.
GSS_C_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_delete_sec_context (delete a security context)

Purpose

Deletes a security context.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_delete_sec_context (
    OM_uint32 *                     minor_status,
    gss_ctx_id_t *                  context_handle,
    gss_buffer_t                    output_token)
```

Parameters

Input/Output

context_handle
Specifies the context to be deleted. Upon successful completion, the context_handle value is set to GSS_C_NO_CONTEXT.

Output

output_token
Returns a token to be sent to the partner application. The partner application then passes this token to the **gss_process_context_token()** routine to delete the other end of the security context. The **gss_delete_sec_context()** routine sets the *output_token* length field to zero if no token needs to be sent to the partner application.

GSS_C_NO_BUFFER may be specified for the *output_token* parameter. In this case, no token is returned by the **gss_delete_sec_context()** routine. Both of the communicating applications must call **gss_delete_sec_context()** in order to delete both ends of the security context.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_delete_sec_context()** routine deletes one end of a security context. It also deletes the local data structures associated with the security context. When it deletes the context, the routine can generate a

token. The application must then pass this token to the partner application. The partner application calls the **gss_process_context_token()** routine to process the token and complete the process of deleting the security context.

If no token is returned, it is up to both client and server to issue **gss_delete_sec_context()** independently of each other when the security context is no longer needed.

This call can be made by either peer in a security context to flush context-specific information. Both communicating applications must call the **gss_delete_sec_context()** routine if GSS_C_NO_BUFFER is specified for the output_token parameter.

The context_handle may not be used for additional security services after the **gss_delete_sec_context()** routine has successfully completed.

Status Codes

Table 40. Status Codes for *gss_delete_sec_context()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The supplied context handle did not refer to a valid context.

gss_display_name (provide the text value of an internal name)

Purpose

Provides the textual representation of an opaque internal name.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_display_name (
    OM_uint32 *            minor_status,
    gss_name_t            input_name,
    gss_buffer_t          output_name_buffer,
    gss_OID *             output_name_type)
```

Parameters

Input

input_name
Specifies the internal name to be converted to a text string.

Output

output_name_buffer
Return buffer for the character string. The **gss_release_buffer()** routine should be called to release the storage when it is no longer needed.

output_name_type
Returns the name type corresponding to the returned character string. The *gss_OID* value returned for this parameter points to read-only storage and must not be released by the application. Specify NULL if the name type is not needed.

minor_status
> Returns a status code from the security mechanism.

Usage

The **gss_display_name()** routine provides an application with the text form of an opaque internal name. The syntax of the text representation is determined by the mechanism which was used to convert the name. The Kerberos name format is the preferred format when an internal name contains multiple name components.

Kerberos names are formatted as *principal-name@realm-name* and the name type is set to **gss_nt_krb5_name**.

Names created by a security mechanism will have a name component for just that mechanism. The Kerberos mechanism uses Kerberos principal names for both source and target names. The SPKM mechanism uses the string representation of the subject name obtained from the authenticating X.509 certificate for both source and target names. The LIPKEY mechanism uses the string representation of the subject name obtained from the authenticating X.509 certificate for the target name and the host userid for the source name.

If the internal name does not have a Kerberos name component, the name is formatted as name-string and the name type is set to **gss_nt_user_name**. The SPKM and LIPKEY mechanisms support anonymous context initiators. An anonymous name is formatted as the string "<anonymous>" and the name type is set to **gss_nt_anonymous**.

Status Codes

Table 41. Status Codes for *gss_display_name()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_NAME	The provided name is not valid.
GSS_S_BAD_NAMETYPE	The internal name provided does not have an internal representation for any of the supported mechanisms.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_display_status (provide the text name of a status code)

Purpose

Provides an application with the textual representation of a GSS or mechanism status code.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_display_status (
    OM_uint32 *             minor_status,
    OM_uint32               status_value,
    int                     status_type,
    gss_OID                 mech_type,
    gss_msg_ctx_t *         message_context,
    gss_buffer_t            status_string)
```

Parameters

Input

status_value
Specifies the status value to be converted. A status value of zero is not valid and causes the **gss_display_status()** routine to return a major status of GSS_S_BAD_STATUS to the application.

status_type
Specifies the status value type and must be one of the following:

- GSS_C_GSS_CODE - GSS major status code
- GSS_C_MECH_CODE - Mechanism minor status code

mech_type
Specifies the security mechanism associated with a minor status code. This parameter is used only when converting a minor status code.

Input/Output

message_context
Indicates whether the status code has multiple messages to be processed. The first time an application calls **gss_display_status()**, the *message_context* parameter must be initialized to zero. The **gss_display_status()** routine returns the first message and sets the *message_context* parameter to a nonzero value if more messages are available. The application then continues to call the **gss_display_status()** routine to obtain the additional messages until the *message_context* value is zero upon return from the **gss_display_status()** routine.

Output

status_string
Returns the text message for the status value.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_display_status()** routine provides the application with a textual representation of a status code. The returned message can then be displayed to the user or written to a log file.

The *message_context* parameter indicates which error message should be returned when a status code has multiple messages. The first time an application calls the **gss_display_status()** routine, it must initialize the *message_context* value to zero. The **gss_display_status()** routine then returns the first message for the status code and sets *message_context* to a nonzero value if there are additional messages available. The application can then continue to call **gss_display_status()** until the *message_context* value is zero upon return.

Status Codes

Table 42. Status Codes for **gss_display_status()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_MECH	The mechanism specified by the *mech_type* parameter is not supported.
GSS_S_BAD_STATUS	The value of the status_type parameter is not GSS_C_GSS_CODE or GSS_C_MECH_CODE, or the value of the *status_value* parameter is not a valid status code.

Table 42. Status Codes for *gss_display_status()* (continued)	
Status Code	**Meaning**
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_duplicate_name (create a duplicate internal name)

Purpose

Creates a duplicate of a GSS-API internal name.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_duplicate_name (
    OM_uint32 *               minor_status,
    gss_name_t                input_name,
    gss_name_t *              output_name)
```

Parameters

Input

input_name
Specifies the name to be duplicated. An error is returned if GSS_C_NO_NAME is specified for this parameter.

Output

output_name
Returns the new GSS-API internal name. The **gss_name_t** returned for this parameter should be released by calling the **gss_release_name()** function when it is no longer needed.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_duplicate_name()** routine makes a copy of a GSS-API internal name.

Status Codes

Table 43. Status Codes for *gss_duplicate_name()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_NAME	The input name is not valid.
GSS_S_BAD_NAMETYPE	The input name type is not supported.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_export_cred (create a GSS-API credential)

Purpose

Creates a credential token for a GSS-API credential.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_export_cred (
    OM_uint32 *                          minor_status,
    gss_cred_id_t                        cred_handle,
    gss_buffer_t                         cred_token)
```

Parameters

Input/Output

cred_handle
    Specifies the credential handle of the GSS-API credential to be used to create the credential token. The credential must be an initiate credential.

Output

cred_token
    Returns the credential token. The storage for the token should be released when it is no longer needed by calling the **gss_release_buffer()** routine.

minor_status
    Returns a status code from the security mechanism.

Usage

The **gss_export_cred()** routine creates a credential token for a GSS-API credential. This credential token can then be given to another process on the same system or on a different system. This second process calls **gss_import_cred()** to create a GSS-API credential from the credential token. In order to use the credential on a different system, the security mechanism must allow the credential to be used from any system. In this case of the Kerberos security mechanism, this means the Kerberos ticket must not contain a client address list.

A credential can be exported only if it is an initiate credential (GSS_C_INITIATE was specified when the credential was created). If the credential is not an initiate credential, the major status is set to GSS_S_NO_CRED. The credential remains available upon completion of the export operation and can be used in subsequent GSS-API operations.

The credential token created by one implementation of GSS-API cannot be used with a different implementation of GSS-API.

Status Codes

Table 44. Status Codes for **gss_export_cred()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

Table 44. Status Codes for *gss_export_cred()* (continued)	
Status Code	**Meaning**
GSS_S_NO_CRED	The supplied credential handle does not refer to a valid credential.

gss_export_name (export an opaque token)

Purpose

Exports a mechanism name as an opaque token.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_export_name (
    OM_uint32 *                         minor_status,
    gss_name_t                          input_name,
    gss_buffer_t                        exported_name)
```

Parameters

Input

input_name
Specifies the GSS-API name to be exported. This must be a mechanism name.

Output

output_token
Returns a token representing the GSS-API name. The **gss_release_buffer()** routine should be called to release the token when it is no longer needed.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_export_name()** routine creates an opaque token for a mechanism name. The **gss_canonicalize_name()** routine converts a GSS-API internal name with multiple mechanism representations to a mechanism name. The **gss_canonicalize_name()** and **gss_export_name()** calls enable callers to acquire and process exported name objects, canonicalized and translated in accordance with the procedures of a particular GSS-API mechanism. Exported name objects can, in turn, be input to **gss_import_name()**, yielding equivalent mechanism names. These facilities are designed specifically to enable efficient storage and comparison of names (for example, for use in access control lists).

Status Codes

Table 45. Status Codes for *gss_export_name()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_NAME	The input name is not valid.
GSS_S_BAD_NAMETYPE	The input name type is not supported.

Table 45. Status Codes for *gss_export_name()* (continued)	
Status Code	**Meaning**
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NAME_NOT_MN	The supplied name is not a mechanism name. Use the **gss_canonicalize_name()** routine to convert an internal name to a mechanism name.

gss_export_sec_context (create a security context token)

Purpose

Creates a security context token for a GSS-API security context.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_export_sec_context (
    OM_uint32 *                         minor_status,
    gss_ctx_id_t *                      context_handle,
    gss_buffer_t                        context_token)
```

Parameters

Input/Output

context_handle
Specifies the context handle of the GSS-API security context to be used to create the security context token. The context handle is set to GSS_C_NO_CONTEXT upon successful completion.

Output

context_token
Returns the security context token. The storage for the token should be released when it is no longer needed by calling the **gss_release_buffer()** routine.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_export_sec_context()** routine creates a context token for a GSS-API security context. This context token can then be given to another process on the same system. This second process calls **gss_import_sec_context()** to create a GSS-API security context from the context token.

Upon successful completion of **gss_export_sec_context()**, the security context is no longer available for use by the current process.

The security context token created by one implementation of GSS-API cannot be used with a different implementation of GSS-API.

Status Codes

Table 46. Status Codes for *gss_export_sec_context()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_CONTEXT_EXPIRED	The supplied context is no longer valid.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The supplied context handle does not refer to a valid context.
GSS_S_UNAVAILABLE	The security context can not be exported.

gss_get_mic (generate a signature)

Purpose

Generates a cryptographic signature for a message.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_get_mic (
    OM_uint32 *         minor_status,
    gss_ctx_id_t        context_handle,
    gss_qop_t           qop_req,
    gss_buffer_t        input_message,
    gss_buffer_t        output_token)
```

Parameters

Input

context_handle
    Specifies the context to be associated with the message when it is sent to the partner application.

qop_req
    Specifies the requested quality of protection for the message. Specify GSS_C_QOP_DEFAULT to use the default quality of protection as defined by the selected security mechanism.

    GSS_C_QOP_DEFAULT should always be specified unless it is necessary to select a specific quality-of-protection algorithm, in which case the application must ensure that the selected algorithm is compatible with the security mechanism associated with the security context. The quality of protection value is the integrity algorithm values. For more information on integrity algorithm values, see your selected security mechanism in the Usage section.

input_message
    Specifies the message for which a signature is to be generated.

Output

output_token
    Returns a token containing the message signature. The message and this token is then sent to the partner application, which calls the **gss_verify_mic()** function to verify the authenticity of

the message. The output token should be released when it is no longer needed by calling the gss_release_buffer() routine.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_get_mic()** routine generates an encrypted signature for a message and returns this signature in a token that can be sent to a partner application. The partner application then calls the **gss_verify_mic()** routine to validate the signature. The **gss_get_qop_list()** routine can be called to obtain a list of supported integrity algorithms for the security context.

Kerberos mechanism

Version 2 of the Kerberos mechanism has deprecated the specification of the quality of protection parameter thereby ignoring any value specified, and performing GSS_C_QOP_DEFAULT behaviour. Version 1 of the Kerberos mechanism only supported DES and DES3 so, for backward compatability, IBM will use Version 1 when the session key is DES or DES3 and Version 2 for any other encryption type.

The Kerberos integrity algorithms are:

- GSS_KRB5_INTEG_C_QOP_DEFAULT - Use the integrity algorithm selected during the **gss_init_sec_context** call. This will be a DES-encrypted MD5 checksum for a DES session key or an encrypted HMAC-SHA1 checksum for a DESD, DES3, AES128 or AES256 session key
- GSS_KRB5_INTEG_C_QOP_MD5 - Truncated MD5 checksum
- GSS_KRB5_INTEG_C_QOP_DES_MD5 - DES-encrypted MD5 checksum
- GSS_KRB5_INTEG_C_QOP_DES_MAC - DES-MAC checksum
- GSS_KRB5_INTEG_C_QOP_HMAC_SHA1 - HMAC-SHA1 checksum

The encryption key associated with the security context determines which quality-of-protection algorithms are available. The GSS_KRB5_INTEG_C_QOP_MD5, GSS_KRB5_INTEG_C_QOP_DES_MD5 and GSS_KRB5_INTEG_C_QOP_DES_MAC algorithms require a 56-bit DES key while the GSS_KRB5_INTEG_C_QOP_HMAC_SHA1 algorithm requires a 168-bit DES3 key. The default integrity algorithm can be requested by specifying GSS_C_QOP_DEFAULT, which is equivalent to specifying GSS_KRB5_INTEG_C_QOP_DEFAULT.

SPKM mechanism

The SPKM integrity algorithms are:

- GSS_SPKM_INTEG_C_QOP_DEFAULT - Default integrity algorithm (HMAC-MD5)
- GSS_SPKM_INTEG_C_QOP_HMAC_MD5 - HMAC-MD5 checksum (uses 128-bit key)
- GSS_SPKM_INTEG_C_QOP_DES_MAC - DES-MAC checksum (uses 56-bit key)
- GSS_SPKM_INTEG_C_QOP_RSA_MD5 - RSA signature using an MD5 checksum (uses X.509 certificate)
- GSS_SPKM_INTEG_C_QOP_RSA_SHA1 - RSA signature using a SHA-1 checksum (uses X.509 certificate)
- GSS_SPKM_INTEG_C_QOP_DSA_SHA1 - DSS signature using a SHA-1 checksum (uses X.509 certificate)

An alternative to specifying an integrity algorithm is to specify a generic integrity level. The security mechanism will select an integrity algorithm which meets the requirements of the specified generic level. The SPKM generic integrity levels are:

- GSS_SPKM_INT_ALG_NON_REP_SUPPORT - Non-repudiable signature (uses X.509 certificate)
- GSS_SPKM_INT_ALG_REPUDIABLE - Negotiated key used to generate integrity checksum

The default integrity algorithm can be requested by specifying GSS_C_QOP_DEFAULT, which is equivalent to specifying GSS_SPKM_INTEG_C_QOP_DEFAULT. An anonymous initiator cannot use an X.509 signature since there is no source certificate.

LIPKEY mechanism

The LIPKEY security mechanism uses the SPKM security mechanism for integrity processing; therefore the LIPKEY QOP values are the same as the SPKM QOP values. The initiator cannot use an X.509 signature since there is no source certificate.

Status Codes

Table 47. Status Codes for **gss_get_mic()**	
Status Code	**Meaning**
`GSS_S_COMPLETE`	The routine completed successfully.
`GSS_S_BAD_QOP`	The requested quality of protection value is not valid.
`GSS_S_CONTEXT_EXPIRED`	The context referred to has expired.
`GSS_S_CREDENTIALS_EXPIRED`	The credentials associated with the referred-to context have expired.
`GSS_S_FAILURE`	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
`GSS_S_NO_CONTEXT`	The context referred to does not exist.

gss_get_qop_list (generate protection level list)

Purpose

Return a list of quality of protection levels for a security context

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_get_qop_list (
    OM_uint32 *          minor_status,
    gss_ctx_id_t         context_handle,
    gss_buffer_t         integ_list,
    gss_buffer_t         conf_list)
```

Parameters

Input

context_handle
    Specifies the context to be queried.

Output

integ_list
    Returns an array of **gss_qop_t** values representing the available integrity algorithms for the security context. The number of elements in the array can be determined by dividing the buffer length by the size of a **gss_qop_t** element. The buffer length will be zero if integrity services are not available. The

array should be released when it is no longer needed by calling the **gss_release_buffer()** routine. Specify NULL for this parameter if the integrity algorithms are not needed.

conf_list

Returns an array of **gss_qop_t** values representing the available confidentiality algorithms for the security context. The number of elements in the array can be determined by dividing the buffer length by the size of a **gss_qop_t** element. The buffer length will be zero if confidentiality services are not available. The array should be released when it is no longer needed by calling the **gss_release_buffer()** routine. Specify NULL for this parameter if the confidentiality algorithms are not needed.

minor_status

Returns a status code from the security mechanism.

Usage

The **gss_get_qop_list()** routine returns the quality of protection values available for use with the **gss_get_mic()**, **gss_wrap()**, and **gss_wrap_size_limit()** routines.

Status Codes

Table 48. Status Codes for **gss_get_qop_list**

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons which are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The referenced context does not exist.
GSS_S_CONTEXT_EXPIRED	The referenced context has expired.

gss_import_cred (create GSS-API credential)

Purpose

Creates a GSS-API credential from a credential token created by the **gss_export_cred()** routine.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_import_cred (
    OM_uint32 *                         minor_status,
    gss_buffer_t                        cred_token,
    gss_ctx_id_t *                      cred_handle)
```

Parameters

Input

cred_token

Specifies the credential token created by the **gss_export_cred()** routine.

Output

cred_handle
> Returns the credential handle for the GSS-API credential created from the credential token. The **gss_release_cred()** routine should be called to release the credential when it is no longer needed.

minor_status
> Returns a status code from the security mechanism.

Usage

The **gss_import_cred()** routine accepts a credential token created by the **gss_export_cred()** routine and creates a GSS-API credential.

The **gss_release_cred()** routine should be called to release the GSS-API credential when it is no longer needed.

The credential token created by one implementation of GSS-API cannot be used with a different implementation of GSS-API.

Status Codes

Table 49. Status Codes for **gss_import_cred()**

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_DEFECTIVE_TOKEN	The supplied credential token is not valid.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_import_name (convert to GSS-API internal format)

Purpose

Converts a printable name to the GSS-API internal format.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_import_name (
    OM_uint32 *             minor_status,
    gss_buffer_t            input_name_buffer,
    gss_OID                 input_name_type,
    gss_name_t *            output_name)
```

Parameters

Input

input_name_buffer
> Specifies the buffer containing the name to convert. The *value* field contains the address of the name, and the *length* field contains the length of the name.

input_name_type
> Specifies the object identifier for the external name type. The following name types are supported:
>
> - GSS_C_NO_OID - specifies the default name type. For the z/OS Kerberos implementation of GSS-API, the default is GSS_C_NT_USER_NAME.

- GSS_C_NT_USER_NAME - specifies a user name.

  For the Kerberos mechanism, the user name is the character string representation of a Kerberos principal and is either the fully-qualified *principal@realm* or the unqualified *principal*. The local realm will be added if an unqualified principal name is specified.

  For the SPKM mechanism, the user name is either the distinguished name for the user or just the common name component. A name is assumed to be a distinguished name if it contains an '=' character, otherwise it is assumed to be the common name component. For example, "CN=John Doe,O=IBM,C=US" is a distinguished name while "John Doe" is the common name component. Refer to RFC 2253 (UTF-8 String Representation of Distinguished Names) for more information on the syntax of the string representation of a distinguished name.

  For the LIPKEY mechanism, the user name is interpreted differently depending upon whether it is a source name or a target name. A target name is handled as described for the SPKM mechanism. A source name must be a name acceptable as a system userid on the target system.

- GSS_C_NT_HOSTBASED_SERVICE - specifies a service that is related to a particular host and is specified as *service@host*. For the Kerberos mechanism, the service name is converted to *service/canonical-name@kerberos-realm*. The canonical-name is obtained by doing a DNS lookup for the supplied host name and obtaining the canonical host name from the name server.

  For the SPKM and LIPKEY mechanisms, the service name is converted to 'service/host' and used as the common name component for the server providing the service. Note that the supplied host name is used without conversion to a canonical host name.

- GSS_C_NT_HOSTBASED_SERVICE_X - specifies a service that is related to a particular host. This is the same as GSS_C_NT_HOSTBASED_SERVICE and should not be used by new applications.

- GSS_C_NT_MACHINE_UID_NAME - specifies the machine representation of a UID (user identifier). The **getpwuid()** function is called to map the UID to a user name. For the Kerberos mechanism, the IRRSIM00 function is then called to map the user name to a Kerberos principal. The application must have at least READ access to the IRR.RUSERMAP facility in order to use this name type.

  The **uid_t** is passed by reference, not by value. That is, the *value* field contains the address of the **uid_t**.

- GSS_C_NT_STRING_UID_NAME - specifies the string representation of a UID (user identifier). The string value is converted to a numeric value and then the **getpwuid()** function is called to map the UID to a user name. For the Kerberos mechanism, the IRRSIM00 function is then called to further map the user name to a Kerberos principal. The application must have at least READ access to the IRR.RUSERMAP facility in order to use this name type.

- GSS_C_NT_EXPORT_NAME - specifies an exported name created by the **gss_export_name()** routine.

- GSS_C_NT_ANONYMOUS - specifies an anonymous name. The input name buffer is not used for an anonymous name and may be specified as GSS_C_NO_BUFFER.

- **gss_nt_krb5_name** - specifies a Kerberos name in the format *principal@realm*. This name type is valid only for the Kerberos mechanism.

- **gss_nt_krb5_principal** - specifies a **krb5_principal** created by the **krb5_parse_name()** routine. This name type is valid only for the Kerberos mechanism.

  The **krb5_principal** is passed by reference, not by value. That is, the *value* field contains the address of the **krb5_principal**.

Output

output_name
    Returns the name in the GSS-API internal format. The internal format contains an internal representation for each of the supported security mechanisms.

minor_status
    Returns a status code from the security mechanism.

Usage

The **gss_import_name()** routine converts a printable name to the internal GSS-API format. The **gss_name_t** object created by this routine can then be used as input to other GSS-API routines. The **gss_name_t** object created by the **gss_import_name()** routine contains an internal representation for each of the supported security mechanisms.

Status Codes

Table 50. Status Codes for **gss_import_name()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_NAME	The input name is not formatted properly as defined by the name type specification.
GSS_S_BAD_NAMETYPE	The name type specified by the *input_name_type* parameter is not valid.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_import_sec_context (create a GSS-API security context)

Purpose

Creates a GSS-API security context from a security context token created by the **gss_export_sec_context()** routine.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_import_sec_context (
    OM_uint32 *                 minor_status,
    gss_buffer_t                context_token,
    gss_ctx_id_t *              context_handle)
```

Parameters

Input

context_token
    Specifies the security context token created by the **gss_export_sec_context()** routine.

Output

context_handle
    Returns the context handle for the security context created from the context token. The **gss_delete_sec_context()** routine should be called to delete the security context when it is no longer needed.

minor_status
    Returns a status code from the security mechanism.

Usage

The **gss_import_sec_context()** routine accepts a security context token created by the **gss_export_sec_context()** routine and creates a GSS-API security context. Since the security context contains message sequencing information, it is usually not feasible to create multiple security contexts from a single context token.

The **gss_delete_sec_context()** routine should be called to delete the GSS-API security context when it is no longer needed.

The security context token created by one implementation of GSS-API cannot be used with a different implementation of GSS-API.

Status Codes

*Table 51. Status Codes for **gss_import_sec_context()***

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_DEFECTIVE_TOKEN	The supplied context token is not valid.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_indicate_mechs (indicate security mechanisms)

Purpose

Allows an application to determine which security mechanisms are available.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_indicate_mechs (
    OM_uint32 *          minor_status,
    gss_OID_set *        mech_set)
```

Parameters

Output

mech_set
Returns the set of supported security mechanisms. The application should release the **gss_OID_set** returned for this parameter by calling the **gss_release_oid_set()** routine.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_indicate_mechs()** routine enables an application to determine which security mechanisms are available on the local system.

Status Codes

Table 52. Status Codes for *gss_indicate_mechs()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_init_sec_context (initiate security context)

Purpose

Initiates a security context for use by two communicating applications.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_init_sec_context (
    OM_uint32 *             minor_status,
    gss_cred_id_t           cred_handle,
    gss_ctx_id_t *          context_handle,
    gss_name_t              target_name,
    gss_OID                 mech_type,
    gss_flags_t             req_flags,
    OM_uint32               time_req,
    gss_channel_bindings_t  input_chan_bindings,
    gss_buffer_t            input_token,
    gss_OID *               actual_mech_type,
    gss_buffer_t            output_token,
    gss_flags_t *           ret_flags,
    OM_uint32 *             time_rec)
```

Parameters

Input

cred_handle
Specifies the credential handle of the GSS-API credential used to initiate the security context. The specified credential must be either an INITIATE or BOTH credential. Specify GSS_C_NO_CREDENTIAL to use the default credential obtained from the current login context.

target_name
Specifies the name of the context acceptor. This must be a Kerberos service name if delegation is requested for the Kerberos security mechanism. Otherwise, it can be any principal defined in the security registry, subject to registry policy rules.

mech_type
Specifies the desired security mechanism:

- **gss_mech_krb5_old** - Beta Kerberos V5 mechanism. The source and target are authenticated using a Kerberos ticket. This mechanism is deprecated and should not be used by new applications. It is only valid with DES and DES3 session keys.

- **gss_mech_krb5** - Kerberos V5 mechanism. The source and target are authenticated using a Kerberos ticket.

- **gss_mech_spkm3** - Low infrastructure version of the simple public key mechanism (SPKM). The source and target are authenticated using X.509 certificates.

- **gss_mech_lipkey** - Low infrastructure public key mechanism (LIPKEY). The source is authenticated using a userid and password and the target is authenticated using an X.509 certificate.
- GSS_C_NO_OID - Default mechanism. For the z/OS Kerberos implementation of GSS-API, this is the Kerberos V5 mechanism.

req_flags
Specifies a bitmask containing independent flags representing requested GSS services. GSS-API does not guarantee that a requested service will be available on all systems. The application should check the **ret_flags** parameter to determine which of the requested services are actually provided for the security context. The following symbolic definitions are provided to correspond to each flag. The symbolic names should be logically ORed to form the bitmask value. Integrity and confidentiality services are always available if they are supported by the security mechanism, thus GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG are ignored when specified as part of the request flags.

- GSS_C_DELEG_FLAG - Request delegated credentials for use by the context acceptor. This flag is ignored for the SPKM-3 and LIPKEY security mechanisms because delegation is not supported.
- GSS_C_MUTUAL_FLAG - Request mutual authentication to validate the identity of both the context initiator and the context acceptor. When both the application client and the application server support the Kerberos Cryptosystem Negotiation Extension and both are capable of using an encryption type that is stronger than the session key selected by the KDC, a new session key will be selected during the mutual authentication using the stronger encryption type. This flag is ignored for the SPKM-3 security mechanism if the GSS_C_ANON_FLAG is set since the initiator cannot be authenticated in this case. Mutual authentication will always be performed for the LIPKEY security mechanism unless the GSS_C_ANON_FLAG is set.
- GSS_C_REPLAY_FLAG - Request message replay detection for signed or sealed messages
- GSS_C_SEQUENCE_FLAG - Request message sequence checking for signed or sealed messages
- GSS_C_ANON_FLAG - Request initiator anonymity. This flag is ignored for the Kerberos security mechanism because the initiator is always identified by the Kerberos service ticket used to establish the security context. Specifying GSS_C_ANON_FLAG for the LIPKEY security mechanism results in the use of the SPKM-3 security mechanism.

time_req
Specifies the desired number of seconds that the security context remains valid. Specify zero for the default lifetime of 2 hours. Specify GSS_C_INDEFINITE to request the maximum lifetime.

input_chan_bindings
Specifies the bindings describing the communications channel to be used between the communicating applications. The channel bindings information is placed into the output token generated by the **gss_init_sec_context()** routine and is validated by the **gss_accept_sec_context()** routine. Specify GSS_C_NO_CHANNEL_BINDINGS if there are no channel bindings.

input_token
Specifies the token received from the context acceptor. GSS_C_NO_BUFFER should be specified if this is the first call to the **gss_init_sec_context()** routine.

Input/Output

context_handle
Specifies the context handle for the context. The first time that the context initiator calls the **gss_init_sec_context()** routine, the context handle must be set to GSS_C_NO_CONTEXT. For subsequent calls to continue setting up the context, the context handle must be the value returned by the previous call to the **gss_init_sec_context()** routine.

Output

actual_mech_type
Returns the security mechanism to be used with the context. The **gss_OID** value returned for this parameter points to read-only storage and must not be released by the application. Specify NULL for this parameter if the actual mechanism type is not needed.

output_token

Returns a token to be sent to the context acceptor. If no token is to be sent to the context acceptor, the **gss_init_sec_context()** routine sets the *output_token* length field to zero. Otherwise, the *output_token* length and value fields are set. The application should release the output token when it is no longer needed by calling the **gss_release_buffer()** routine.

ret_flags

Returns a bitmask containing independent flags indicating which GSS services are available for the context. Specify NULL for this parameter if the flags are not needed. The following symbolic definitions are provided to test the individual flags and should be logically ANDed with the value of *ret_flags* to test whether the context supports the service options.

- GSS_C_DELEG_FLAG - Delegated credentials are available to the context acceptor
- GSS_C_MUTUAL_FLAG - Mutual authentication will be performed. The **gss_accept_sec_context()** routine generates an output token that the context acceptor must return to the context initiator to complete the security context setup.
- GSS_C_REPLAY_FLAG - Message replay detection will be performed
- GSS_C_SEQUENCE_FLAG - Message sequence checking will be performed
- GSS_C_CONF_FLAG - Message confidentiality services are available
- GSS_C_INTEG_FLAG - Message integrity services are available
- GSS_C_ANON_FLAG - The initiator identity will not be provided to the context acceptor
- GSS_C_PROT_READY_FLAG - If this flag is set, protection services, as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG, are available for use if the accompanying major status return value is GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED. Otherwise, protection services are available for use only if the accompanying major status return value is GSS_S_COMPLETE.
- GSS_C_TRANS_FLAG - If this flag is set, the **gss_export_sec_context()** function can be used to export the security context. The **gss_export_sec_context()** function is not available if this flag is not set.

time_rec

Return the number of seconds the context remains valid. If the mechanism does not support context expiration, the return value is GSS_C_INDEFINITE. Specify NULL for this parameter if the context expiration time is not required.

minor_status

Returns a status code from the security mechanism.

Usage

The **gss_init_sec_context()** routine is the first step in the establishment of a security context between the context initiator and the context acceptor. To ensure the portability of the application, use the default credential by specifying GSS_C_NO_CREDENTIAL for the *cred_handle* parameter.

The first time the application calls the **gss_init_sec_context()** routine, the *input_token* parameter should either be specified as GSS_C_NO_BUFFER or the buffer length field should be set to zero. If no token needs to be sent to the context acceptor, the **gss_init_sec_context()** routine sets the *output_token* length field to zero.

To finish establishing the context, the calling application can require one or more tokens from the context acceptor. If the application requires reply tokens, the **gss_init_sec_context()** routine returns GSS_S_CONTINUE_NEEDED in the supplementary information portion of the major status value. The application must call the **gss_init_sec_context()** routine again when it receives the reply token from the context acceptor and pass the token by way of the *input_token* parameter. When the **gss_init_sec_context()** routine is called to continue processing a context, the same request values must be used as for the initial call.

The availability of confidentiality services is dependent upon the underlying security mechanism and the features that have been installed on the system. The GSS_C_CONF_FLAG is returned only if confidentiality

services are available on the local system. However, this does not guarantee that confidentiality services are also available on the remote system. If confidentiality services are available on the local system but not on the remote system, an error is returned by the **gss_unwrap()** routine on the remote system if an encrypted message is received (that is, confidentiality was requested on the call to the **gss_wrap()** routine on the local system).

Whenever the routine returns a major status that includes the value GSS_S_CONTINUE_NEEDED, the context is not fully established and the following restrictions apply to the output parameters:

- The value returned by the *time_rec* parameter is undefined.
- Unless the accompanying *ret_flags* parameter contains the bit GSS_C_PROT_READY_FLAG, indicating that per-message services may be applied in advance of a successful completion status, the value returned by the *actual_mech_type* parameter is undefined until the routine returns a major status value of GSS_S_COMPLETE.
- The values of the GSS_C_DELEG_FLAG, GSS_C_MUTUAL_FLAG, GSS_C_REPLAY_FLAG, GSS_C_SEQUENCE_FLAG, GSS_C_CONF_FLAG, GSS_C_INTEG_FLAG, and GSS_C_ANON_FLAG bits returned by the *ret_flags* parameter contain the values that would be returned if the context establishment were to succeed. In particular, if the application has requested a service such as delegation or anonymous authentication by means of the *req_flags* parameter, and such a service is unavailable from the underlying mechanism, **gss_init_sec_context()** generates a token that does not provide the service and indicates through the *ret_flags* parameter that the service is not supported. The application may choose to stop the context establishment by calling **gss_delete_sec_context()** or it may choose to transmit the token and continue context establishment.
- The value of the GSS_C_PROT_READY_FLAG bit returned by the *ret_flags* parameter indicates the actual state at the time **gss_init_sec_context()** returns, whether or not the context is fully established.

Kerberos Mechanism

In order for delegation to be used, the target principal name must be a service name. A service name is created by calling the **gss_import_name()** routine with the name type specified as GSS_C_NT_HOSTBASED_SERVICE (object identifier {1 2 840 113554 1 2 1 4}). The service name is specified as *name@host* and results in a Kerberos principal of *name/host@host-realm*. The local host name is used if no host is specified. If a host name alias is specified, the primary host name returned by the domain name service is used when constructing the principal name. The target principal name is not required to be a service name if the ticket-granting ticket (TGT) does not contain a client address list. You can obtain a TGT without a client address list by specifying the **- A** option on the **kinit** command. Otherwise, the service name must correctly identify the host the target service is running on.

The requested context lifetime is used to specify the endtime when obtaining a Kerberos service ticket to the target application. The actual context lifetime is then set to the lifetime of the ticket, which may be less than the requested lifetime as determined by the registry policy.

If delegation is requested, the TGT contained in the login context must allow forwardable tickets. If the TGT is not forwardable, the **gss_init_sec_context()**) request will be successful but the GSS_C_DELEG_FLAG will not be set in the returned flags. In addition, the service ticket obtained for the target principal must allow delegation. If the target server is not enabled for delegation, the **gss_init_sec_context()** request will be successful but the GSS_C_DELEG_FLAG will not be set in the returned flags. You can use the **klist** command with the **-f** option to display the ticket flags. The TGT must have the F flag set and the service ticket must have the O flag set.

SPKM mechanism

The target name is created by calling **gss_import_name**. It is validated against the target certificate during **gss_accept_sec_context**. For more details, see "gss_import_name (convert to GSS-API internal format)" on page 220.

The initiator is authenticated using an X.509 certificate. The certificate is obtained from the supplied GSS-API credential. If no credential is provided, the default certificate for the application will be used.

The acceptor is always authenticated to the initiator and results in a two-way token exchange (gss_init_sec_context followed by gss_accept_sec_context followed by gss_init_sec_context). The initiator will be authenticated to the acceptor if the GSS_C_MUTUAL_FLAG is specified and results in a three-way token exchange (gss_init_sec_context followed by gss_accept_sec_context followed by gss_init_sec_context followed by gss_accept_sec_context). The GSS_C_MUTUAL_FLAG will be ignored if the GSS_C_ANON_FLAG is also specified since the initiator cannot be authenticated in this case.

Diffie-Hellman key agreement is used to compute the secret value required by the key generation process. This is a two-pass algorithm requiring inputs from both the initiator and the acceptor. The initial output token created by the gss_init_sec_context() routine will specify Diffie-Hellman key agreement as the default key establishment algorithm for the context and will contain the Diffie-Hellman public value for the initiator. The output token created by the gss_accept_sec_context() routine will contain the Diffie-Hellman public value for the acceptor.

LIPKEY mechanism

The target name is created by calling **gss_import_name**. It is validated against the target certificate during **gss_accept_sec_context**. For more details, see "gss_import_name (convert to GSS-API internal format)" on page 220.

The initiator is authenticated using a userid and password known to the target application. The userid and password will be obtained from the supplied GSS-API credential. If no credential is provided, the current system userid will be used and the user will be prompted to supply the password associated with this userid. The **gss_init_sec_context()** routine will return an error if confidentiality services are not available since the user name and password cannot be sent without encryption.

Key database usage

The SPKM and LIPKEY mechanisms use X.509 certificates. These certificates and associated certification authority certificates are obtained from a key database or SAF key ring.

The GSS_KEYRING_NAME environment variable specifies the name of the key database or SAF key ring. The GSS_KEYRING_PW or GSS_KEYRING_STASH environment variable specifies the password for the key database (GSS_KEYRING_STASH is ignored if GSS_KEYRING_PW is defined). A SAF key ring is used if neither GSS_KEYRING_PW nor GSS_KEYRING_STASH is defined.

The GSS_KEY_LABEL environment variable specifies the label of the default certificate. The default certificate for the key database or SAF key ring will be used if this variable is not defined.

Status Codes

Table 53. Status Codes for **gss_init_sec_context()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_BINDINGS	The channel bindings are not valid.
GSS_S_BAD_MECH	The request security mechanism is not supported
GSS_S_BAD_NAME	The *target_name* parameter is not valid.
GSS_S_BAD_SIG	The input token contains an incorrect integrity check value.
GSS_S_CONTINUE_NEEDED	To complete the context, the **gss_init_sec_context()** routine must be called again with a token created by the **gss_accept_sec_context()** routine.
GSS_S_CREDENTIALS_EXPIRED	The supplied credentials are no longer valid.

Table 53. Status Codes for **gss_init_sec_context()** (continued)	
Status Code	**Meaning**
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_DEFECTIVE_CREDENTIAL	Consistency checks performed on the credential failed.
GSS_S_DEFECTIVE_TOKEN	Consistency checks performed on the input token failed.
GSS_S_DUPLICATE_TOKEN	The token is a duplicate of a token that has already been processed.
GSS_S_NO_CONTEXT	The supplied context handle does not refer to a valid context.
GSS_S_NO_CRED	The supplied credential handle does not refer to a valid credential, the supplied credential is not valid for context initiation, or there are no default credentials available.
GSS_S_OLD_TOKEN	The token is too old to be checked for duplication against tokens that have already been processed.

gss_inquire_context (obtain security context information)

Purpose

Returns information about a security context.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_inquire_context (
    OM_uint32 *              minor_status,
    gss_ctx_id_t             context_handle,
    gss_name_t *             src_name,
    gss_name_t *             tgt_name,
    OM_uint32 *              lifetime,
    gss_OID *                mech_type,
    gss_flags_t *            ret_flags,
    int *                    local,
    int *                    open)
```

Parameters

Input

context_handle
    Specifies the handle for the security credential.

Output

src_name
    Returns the principal name associated with the context initiator. Specify NULL for this parameter if the principal name is not required.

tgt_name
    Returns the principal name associated with the context acceptor. Specify NULL for this parameter if the principal name is not required.

lifetime
Returns the number of seconds the context remains valid. Specify NULL for this parameter if the context lifetime is not required. The returned value is GSS_C_INDEFINITE if the security mechanism does not support context expiration. The returned value is 0 if the context is expired.

mech_type
Returns the mechanism used to create the security context. The *gss_OID* value returned for this parameter points to read-only storage and must not be released by the application. Specify NULL for this parameter if the mechanism type is not required.

ret_flags
Returns a bitmask containing independent flags indicating which GSS services are available for the context. Specify NULL for this parameter if the available service flags are not required. The following symbolic definitions are provided to test the individual flags and should be logically ANDed with the value of *ret_flags* to test whether the context supports the service options.

- GSS_C_DELEG_FLAG - Delegated credentials are available to the context acceptor.
- GSS_C_MUTUAL_FLAG - Mutual authentication will be performed. The **gss_accept_sec_context()** routine generates an output token that the context acceptor must return to the context initiator to complete the security context setup.
- GSS_C_REPLAY_FLAG - Message replay detection will be performed
- GSS_C_SEQUENCE_FLAG - Message sequence checking will be performed.
- GSS_C_CONF_FLAG - Message confidentiality services are available.
- GSS_C_INTEG_FLAG - Message integrity services are available.
- GSS_C_ANON_FLAG - The initiator identity will not be provided to the context acceptor.
- GSS_C_PROT_READY_FLAG - If set, protection services, as specified by the states of the GSS_C_CONF_FLAG and GSS_C_INTEG_FLAG bits, are available for use even if the context is not fully established. Otherwise, protection services are available only if the value returned by the open parameter is TRUE.
- GSS_C_TRANS_FLAG - If this flag is set, the **gss_export_sec_context()** function can be used to export the security context. The **gss_export_sec_context()** function is not available if this flag is not set.

local
Returns TRUE if the context was initiated locally and FALSE otherwise. Specify NULL for this parameter if the local indication is not required.

open
Returns TRUE if context establishment has been completed and FALSE otherwise. Specify NULL for this parameter if the open indication is not required.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_inquire_context()** routine provides information about a security context to the calling application.

Status Codes

Table 54. Status Codes for **gss_inquire_context()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.

Table 54. Status Codes for **gss_inquire_context()** (continued)	
Status Code	**Meaning**
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The context referred to does not exist.

gss_inquire_cred (obtain GSS-API credential information)

Purpose

Returns information about a GSS-API credential.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_inquire_cred (
    OM_uint32 *             minor_status,
    gss_cred_id_t           cred_handle,
    gss_name_t *            name,
    OM_uint32 *             lifetime,
    gss_cred_usage_t *      cred_usage,
    gss_OID_set *           mechanisms)
```

Parameters

Input

cred_handle
    Specifies the handle for the GSS-API credential. Specify GSS_C_NO_CREDENTIAL to get information about the default credential for the default security mechanism.

Output

name
    Returns the principal name associated with the credential. Specify NULL for this parameter if the principal name is not required. The name should be released when it is no longer needed by calling the **gss_release_name()** routine.

lifetime
    Returns the number of seconds the credential remains valid. The return value is set to zero if the credential has expired. Specify NULL for this parameter if the credential lifetime is not required.

cred_usage
    Returns one of these values describing how the application can use the credential. Specify NULL for this parameter if the credential usage is not required.

- GSS_C_INITIATE - the application may initiate a security context
- GSS_C_ACCEPT - the application may accept a security context
- GSS_C_BOTH - the application may both initiate and accept security contexts

mechanisms
    Returns the set of security mechanisms supported by the credential. Specify NULL for this parameter if the mechanism set is not required. The **gss_OID_set** returned for this parameter should be released when it is no longer needed by calling the **gss_release_oid_set()** routine.

minor_status
    Returns a status code from the security mechanism.

Usage

The **gss_inquire_cred()** routine provides information about a GSS-API credential to the calling application. If GSS_C_NO_CREDENTIAL is specified for the *cred_handle* parameter, the default security mechanism is used to process the request. A credential for the LIPKEY security mechanism that is used for both initiate and accept will have two names associated with it. The initiate name is the name used to authenticate the initiator on the target system while the accept name is the subject name obtained from the X.509 certificate associated with the credential. In this case, the gss_inquire_cred() routine will return the initiate name for the name parameter.

Status Codes

*Table 55. Status Codes for **gss_inquire_cred()***

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_CREDENTIALS_EXPIRED	The credentials have expired. Credential information will still be returned for an expired credential but the lifetime value will be returned as zero.
GSS_S_DEFECTIVE_CREDENTIAL	The credentials are not valid.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CRED	The cred_handle does not refer to a valid credential or there are no default credentials available.

gss_inquire_cred_by_mech (obtain single mechanism credential information)

Purpose

Returns information about a GSS-API credential for a single security mechanism.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_inquire_cred_by_mech (
    OM_uint32 *            minor_status,
    gss_cred_id_t          cred_handle,
    gss_OID                mech_type,
    gss_name_t *           name,
    OM_uint32 *            init_lifetime,
    OM_uint32 *            accept_lifetime,
    gss_cred_usage_t *     cred_usage)
```

Parameters

Input

cred_handle
Specifies the handle for the GSS-API credential. Specify GSS_C_NO_CREDENTIAL to get information about the default credential for the specified security mechanism.

mech_type
Specifies the mechanism to be used to obtain the return information as follows:

- **gss_mech_krb5_old** - Beta Kerberos V5 mechanism
- **gss_mech_krb5** - Kerberos V5 mechanisms
- **gss_mech_spkm3** - Low infrastructure version of the simple public key mechanism (SPKM)
- **gss_mech_lipkey** - Low infrastructure public key mechanism (LIPKEY)

Output

name
Returns the principal name associated with the credential. Specify NULL for this parameter if the principal name is not required. The name should be released when it is no longer needed by calling the **gss_release_name()** routine.

init_lifetime
Returns the number of seconds the credential remains valid for initiating contexts. Specify NULL for this parameter if the credential lifetime is not required.

accept_lifetime
Returns the number of seconds the credential remains valid for accepting contexts. Specify NULL for this parameter if the credential lifetime is not required.

cred_usage
Returns one of the following values describing how the application can use the credential. Specify NULL for this parameter if the credential usage is not required.

- GSS_C_INITIATE - the application may initiate a security context
- GSS_C_ACCEPT - the application may accept a security context
- GSS_C_BOTH - the application may both initiate and accept security contexts

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_inquire_cred_by_mech()** routine provides information about a GSS-API credential to the calling application. The information is obtained using the specified security mechanism.

Status Codes

Table 56. Status Codes for *gss_inquire_cred_by_mech()*

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_MECH	The requested mechanism is not supported.
GSS_S_CREDENTIALS_EXPIRED	The credentials have expired. Credential information is still returned for an expired credential but the lifetime value is returned as zero.
GSS_S_DEFECTIVE_CREDENTIAL	The credentials are not valid.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CRED	The *cred_handle* does not refer to a valid credential or there are no default credentials available.

gss_inquire_mechs_for_name (obtain available mechanisms)

Purpose

Returns the mechanisms with which a name may be processed.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_inquire_mechs_for_name (
    OM_uint32 *                         minor_status,
    gss_name_t                          input_name,
    gss_OID_set *                       mech_types)
```

Parameters

Input

input_names
Specifies the name to be queried.

Output

mech_types
Returns the mechanisms that can be used with the specified name. The **gss_OID_set** returned for this parameter should be released by calling the **gss_release_oid_set()** routine when it is no longer needed.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_inquire_mechs_for_name()** routine returns the set of mechanisms that can be used with a given name.

Status Codes

Table 57. Status Codes for **gss_inquire_mechs_for_name()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_NAME	The supplied name is not valid.
GSS_S_BAD_NAMETYPE	The name type is not supported.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_inquire_names_for_mech (obtain supported mechanisms)

Purpose

Returns the name types supported by a security mechanism.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_inquire_names_for_mech (
```

```
    OM_uint32 *         minor_status,
    gss_OID            mech_type,
    gss_OID_set *      mech_names)
```

Parameters

Input

mech_type
Specifies the mechanism to be queried as follows:

- **gss_mech_krb5_old** - Beta Kerberos V5 mechanism
- **gss_mech_krb5** - Kerberos V5 mechanism
- **gss_mech_spkm3** - Low infrastructure version of the simple public key mechanism (SPKM)
- **gss_mech_lipkey** - Low infrastructure public key mechanism (LIPKEY)

Output

mech_names
Returns the name types supported by the specified mechanism. The **gss_OID_set** returned for this parameter should be released by calling the **gss_release_oid_set()** routine when it is no longer needed.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_inquire_names_for_mech()** routine returns the set of name types that are supported by a particular security mechanism.

Status Codes

Table 58. Status Codes for **gss_inquire_names_for_mech()**

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_MECH	The requested mechanism is not supported.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_oid_to_str (convert to a string)

Purpose

Converts a gss_OID object to a string representation of the object identifier.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_oid_to_str (
    OM_uint32 *             minor_status,
    gss_OID                input_oid,
    gss_buffer_t           output_string)
```

Parameters

Input

input_oid
> Specifies the **gss_OID** to be converted.

Output

output_string
> Returns the string representation of the object identifier. The **gss_buffer_t** returned for this parameter should be released by calling the **gss_release_buffer()** routine when it is no longer needed.

minor_status
> Returns a status code from the security mechanism.

Usage

The **gss_oid_to_str()** routine converts a **gss_OID** object to a string representation of the object identifier. The string representation consists of a series of blank-separated numbers enclosed in braces. The **gss_str_to_oid()** routine can be used to convert the string representation back to a **gss_OID** object.

Status Codes

*Table 59. Status Codes for **gss_oid_to_str()***

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_process_context_token (process a context token)

Purpose

Processes a context token received from the partner application.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_process_context_token (
    OM_uint32 *             minor_status,
    gss_ctx_id_t            context_handle,
    gss_buffer_t            input_token)
```

Parameters

Input

context_handle
> Specifies the context to be used when processing the token.

input_token
> Specifies the token received from the partner application.

Output

minor_status
  Returns a status code from the security mechanism.

Usage

The **gss_process_context_token()** routine processes tokens generated by the partner application. Tokens are usually associated with either the context establishment or with message security services. If the tokens are associated with the context establishment, they are processed by the **gss_init_sec_context()** and **gss_accept_sec_context()** routines. If the tokens are associated with message security services, they are processed by the **gss_verify_mic()** and **gss_unwrap()** routines. Tokens generated by the **gss_delete_sec_context()** routine, however, are processed by the **gss_process_context_token()** routine.

Status Codes

*Table 60. Status Codes for **gss_process_context_token()***

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_SIG	The token signature was not correct.
GSS_S_DEFECTIVE_TOKEN	Consistency checks performed on the input token failed.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The context handle does not refer to a valid security context.

gss_release_buffer (release buffer storage)

Purpose

Releases storage associated with a **gss_buffer_t** buffer. The **gss_buffer_desc** structure itself is not released.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_release_buffer (
    OM_uint32 *           minor_status,
    gss_buffer_t          buffer)
```

Parameters

Input/Output

buffer
  The buffer to be released. Upon successful completion, the length and value fields will be set to zero.

Output

minor_status
  Returns a status code from the security mechanism.

Usage

The **gss_release_buffer()** routine releases storage associated with a **gss_buffer_t** buffer. It does not release the storage for the **gss_buffer_desc** structure itself.

Status Codes

Table 61. Status Codes for *gss_release_buffer()*	
Status Code	**Meaning**
`GSS_S_COMPLETE`	The routine completed successfully.
`GSS_S_FAILURE`	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_release_cred (release local credentials)

Purpose

Releases local data structures associated with a GSS-API credential.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_release_cred (
    OM_uint32 *          minor_status,
    gss_cred_id_t *      cred_handle)
```

Parameters

Input/Output

cred_handle
Specifies the credential to be released. Upon successful completion, the *cred_handle* value is set to GSS_C_NO_CREDENTIAL. If the *cred_handle* value is GSS_C_NO_CREDENTIAL, the major status is set to GSS_S_COMPLETE and nothing is released.

Output

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_release_cred()** routine releases the local data structures for the specified credential. If GSS_C_NO_CREDENTIAL is specified for the *cred_handle* parameter, no credential is released and GSS_S_COMPLETE is returned for the major status.

Status Codes

Table 62. Status Codes for *gss_release_cred()*	
Status Code	**Meaning**
`GSS_S_COMPLETE`	The routine completed successfully.

Table 62. Status Codes for **gss_release_cred()** (continued)	
Status Code	**Meaning**
GSS_S_DEFECTIVE_CREDENTIAL	Consistency checks performed on the credential structure failed.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CRED	The *cred_handle* parameter does not refer to a valid credential.

gss_release_name (release internal name storage)

Purpose

Releases storage associated with a **gss_name_t** internal name.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_release_name (
    OM_uint32 *            minor_status,
    gss_name_t *           name)
```

Parameters

Input/Output

name
Specifies the name to be released. Upon successful completion, the name value is set to GSS_C_NO_NAME

Output

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_release_name()** routine releases storage associated with a GSS-API internal name.

Status Codes

Table 63. Status Codes for **gss_release_name()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_NAME	The specified name is not valid.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_release_oid (release gss_OID storage)

Purpose

Releases the storage associated with a **gss_OID** object.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_release_oid (
    OM_uint32 *           minor_status,
    gss_OID *             oid)
```

Parameters

Input/Output

oid

Specifies the **gss_OID** to be released. Upon successful completion, the *oid* value is set to GSS_C_NO_OID.

Output

minor_status

Returns a status code from the security mechanism.

Usage

The **gss_release_oid()** routine releases the storage associated with a **gss_OID** object.

Status Codes

Status Code	Meaning
*Table 64. Status Codes for **gss_release_oid()***	
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_release_oid_set (release gss_OID_set storage)

Purpose

Releases the storage associated with a **gss_OID_set** object.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_release_oid_set (
    OM_uint32 *           minor_status,
    gss_OID_set *         oid_set)
```

Parameters

Input/Output

oid_set
Specifies the **gss_OID_set** to be released. Upon successful completion, the *oid_set* value is set to GSS_C_NO_OID_SET.

Output

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_release_oid_set()** routine releases the storage associated with a **gss_OID_set** object.

Status Codes

*Table 65. Status Codes for **gss_release_oid_set()***

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_str_to_oid (convert to gss_OID)

Purpose

Converts the string representation of an object identifier to a **gss_OID** object.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_str_to_oid (
    OM_uint32 *            minor_status,
    gss_buffer_t           input_string,
    gss_OID *              output_oid)
```

Parameters

Input

input_string
Specifies the string to be converted.

Output

output_oid
Returns the object identifier. The **gss_OID** returned for this parameter should be released by calling the **gss_release_oid()** routine when it is no longer needed.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_str_to_oid()** routine converts the string representation of an object identifier to a **gss_OID** object. The string representation is a series of blank-separated or period-separated numbers enclosed in braces. For example, the Kerberos V5 security mechanism object identifier is represented as {1 2 840 113554 1 2 2}.

While the blank-separated form should be used for portability, the **gss_str_to_oid()** routine also accepts the period-separated form for compatibility with other applications. However, the **gss_oid_to_str()** routine always generates the blank-separated form.

Status Codes

Table 66. Status Codes for **gss_str_to_oid()**

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_test_oid_set_member (check OID for membership)

Purpose

Checks an OID set to see if a specified OID is in the set.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_test_oid_set_member (
    OM_uint32 *                  minor_status,
    gss_OID                      member_oid,
    gss_OID_set                  oid_set,
    int *                        is_present)
```

Parameters

Input

member_oid
Specifies the OID to search for in the OID set.

oid_set
Specifies the OID set to check.

Output

is_present
Is set to 1 if the OID is a member of the OID set and to zero otherwise.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_test_oid_set_member()** routine checks an OID set to see if the specified OID is a member of the set. The **gss_create_empty_oid_set()** routine can be used to create an empty OID set and the **gss_add_oid_set_member()** routine can be used to add an OID to an existing OID set.

Status Codes

Table 67. Status Codes for *gss_test_oid_set_member()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_unwrap (unwrap and verify a message)

Purpose

Unwraps a message sealed by the **gss_wrap()** routine and verifies the embedded signature.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_unwrap (
    OM_uint32 *              minor_status,
    gss_ctx_id_t             context_handle,
    gss_buffer_t             input_message,
    gss_buffer_t             output_message,
    int *                    conf_state,
    gss_qop_t *              qop_state)
```

Parameters

Input

context_handle
Specifies the context on which the message arrived.

input_message
Specifies the sealed message token generated by the **gss_wrap()** routine.

Output

output_message
Returns the unsealed message.

conf_state
Returns the level of confidentiality applied to the message. Specify NULL for this parameter if the confidentiality state is not needed. The return value is:

- TRUE - Both confidentiality and integrity services were applied.
- FALSE - Only integrity services were applied.

qop_state
Returns the quality of protection applied to the message. Specify NULL for this parameter if the quality of protection is not needed.

minor_status
> Returns a status code from the security mechanism.

Usage

The **gss_unwrap()** routine extracts a message from the sealed token created by the **gss_wrap()** routine and verifies the embedded signature. The *conf_state* return parameter indicates whether the message had been encrypted.

Status Codes

Table 68. Status Codes for **gss_unwrap()**

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_SIG	The token signature is not correct.
GSS_S_CONTEXT_EXPIRED	The context referred to has expired.
GSS_S_CREDENTIALS_EXPIRED	The credentials associated with the context referred to have expired.
GSS_S_DEFECTIVE_TOKEN	Consistency checks performed on the input token failed.
GSS_S_DUPLICATE_TOKEN	The token is a duplicate of a token that has already been processed.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_GAP_TOKEN	One or more predecessor tokens have not been processed.
GSS_S_NO_CONTEXT	The context referred to is not valid.
GSS_S_OLD_TOKEN	The token is too old to be checked for duplication against tokens that have already been processed.
GSS_S_UNSEQ_TOKEN	A later token has already been processed.

gss_verify_mic (verify a signature)

Purpose

Verifies that the cryptographic signature for a message is correct.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_verify_mic (
    OM_uint32 *        minor_status,
    gss_ctx_id_t       context_handle,
    gss_buffer_t       input_message,
    gss_buffer_t       input_token,
    gss_qop_t *        qop_state)
```

Parameters

Input

context_handle
Specifies the context on which the message arrived.

input_message
Specifies the message to be verified.

input_token
Specifies the signature token generated by the **gss_get_mic()** routine.

Output

qop_state
Returns the quality of protection that was applied to the message. Specify NULL for this parameter if the quality of protection is not needed.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_verify_mic()** routine checks that the encrypted signature is the correct signature for the supplied message. This ensures that the message has not been modified since the signature was generated.

Status Codes

*Table 69. Status Codes for **gss_verify_mic()***

Status Code	Meaning
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_SIG	The input token is not valid.
GSS_S_CONTEXT_EXPIRED	The context referred to has expired.
GSS_S_CREDENTIALS_EXPIRED	The credentials associated with the context referred to have expired.
GSS_S_DEFECTIVE_CREDENTIAL	The credential is defective.
GSS_S_DEFECTIVE_TOKEN	Consistency checks performed on the input token failed
GSS_S_DUPLICATE_TOKEN	The input token is a duplicate of a token that has already been processed.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_GAP_TOKEN	One or more predecessor tokens have not been processed.
GSS_S_NO_CONTEXT	The context referred to is not valid.
GSS_S_OLD_TOKEN	The input token is too old to be checked for duplication against tokens that have already been processed.
GSS_S_UNSEQ_TOKEN	A later token has already been processed.

gss_wrap (sign and encrypt a message)

Purpose

Cryptographically signs and optionally encrypts a message.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_wrap (
    OM_uint32 *        minor_status,
    gss_ctx_id_t       context_handle,
    int                conf_req_flag,
    gss_qop_t          qop_req,
    gss_buffer_t       input_message,
    int *              conf_state,
    gss_buffer_t       output_message)
```

Parameters

Input

context_handle
Specifies the context to be associated with the message when it is sent to the partner application.

conf_req_flag
Specifies the requested level of confidentiality and integrity services as follows:

- TRUE - Both confidentiality and integrity services are requested.
- FALSE - Only integrity services are requested.

qop_req
Specifies the requested quality of protection for the message. Specify GSS_C_QOP_DEFAULT to use the default quality of protection as defined by the selected security mechanism.

GSS_C_QOP_DEFAULT should always be specified unless it is necessary to select a specific quality-of-protection algorithm, in which case the application must ensure that the selected algorithm is compatible with the security mechanism associated with the security context. The quality of protection value is formed by or'ing together one of the integrity algorithm values and one of the sealing algorithm values. For more information on integrity and sealing algorithm values, see your selected security mechanism in the Usage section.

input_message
Specifies the message to be wrapped.

Output

conf_state
Returns the level of confidentiality that was applied to the message. Specify NULL for this parameter if the confidentiality state is not required. The return value is:

- TRUE - Both confidentiality and integrity services have been applied.
- FALSE - Only integrity services have been applied.

output_message
Returns the wrapped message. The buffer should be released when it is no longer needed by calling the **gss_release_buffer()** routine.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_wrap()** routine cryptographically signs and optionally encrypts a message. The token returned in the *output_message* parameter contains both the signature and the message. This token is then sent to the partner application, which calls the **gss_unwrap()** routine to extract the original message and verify its authenticity.

If confidentiality is requested (the *conf_req_flag* is TRUE) but confidentiality services are not available for the security context, no error is returned and only integrity services are performed. The *conf_state* return parameter indicates whether the requested confidentiality services were performed. The strong cryptographic algorithms may not be available on a particular system due to government export regulations. The gss_get_qop_list() routine can be called to obtain a list of supported integrity and confidentiality algorithms for the security context.

Kerberos mechanism

Version 2 of the Kerberos mechanism has deprecated the specification of the quality of protection parameter thereby ignoring any value specified, and performing GSS_C_QOP_DEFAULT behavior. Version 1 of the Kerberos mechanism only supported DES and DES3 so, for backward compatability, IBM uses Version 1 when the session key is DES or DES3 and Version 2 for any other encryption type.

The Kerberos integrity algorithms are:

- GSS_KRB5_INTEG_C_QOP_DEFAULT - Default integrity algorithm. Use the integrity algorithm selected during the **gss_init_sec_context** call. This will be a DES-encrypted MD5 checksum for a DES session key or an encrypted HMAC-SHA1 checksum for a DESD, DES3, AES128 or AES256 session key.
- GSS_KRB5_INTEG_C_QOP_MD5 - Truncated MD5 checksum
- GSS_KRB5_INTEG_C_QOP_DES_MD5 - DES-encrypted MD5 checksum
- GSS_KRB5_INTEG_C_QOP_DES_MAC - DES-MAC checksum
- GSS_KRB5_INTEG_C_QOP_HMAC_SHA1 - HMAC-SHA1 checksum

The Kerberos confidentiality algorithms are:

- GSS_KRB5_CONF_C_QOP_DEFAULT - Default confidentiality algorithm (56-bit DES for a DES session key or 168-bit DES3 for a DES3 session key)
- GSS_KRB5_CONF_C_QOP_DES - 56-bit DES encryption
- GSS_KRB5_CONF_C_QOP_DES3_KD - 168-bit DES3 encryption with key derivation

The encryption key associated with the security context determines which quality-of-protection algorithms are available. The GSS_KRB5_CONF_C_QOP_DES, GSS_KRB5_INTEG_C_QOP_MD5, GSS_KRB5_INTEG_C_QOP_DES_MD5 and GSS_KRB5_INTEG_C_QOP_DES_MAC algorithms require a 56-bit DES key while the GSS_KRB5_CONF_C_QOP_DES3_KD and GSS_KRB5_INTEG_C_QOP_HMAC_SHA1 algorithms require a 168-bit DES3 key. The default algorithms can be requested by specifying GSS_C_QOP_DEFAULT, which is equivalent to specifying GSS_KRB5_INTEG_C_QOP_DEFAULT | GSS_KRB5_CONF_C_QOP_DEFAULT.

SPKM mechanism

The SPKM integrity algorithms are:

- GSS_SPKM_INTEG_C_QOP_DEFAULT - Default integrity algorithm (HMAC-MD5)
- GSS_SPKM_INTEG_C_QOP_HMAC_MD5 - HMAC-MD5 checksum
- GSS_SPKM_INTEG_C_QOP_DES_MAC - DES-MAC checksum (uses 56-bit key)
- GSS_SPKM_INTEG_C_QOP_RSA_MD5 - RSA signature using an MD5 checksum (uses X.509 certificate)
- GSS_SPKM_INTEG_C_QOP_RSA_SHA1 - RSA signature using a SHA-1 checksum (uses X.509 certificate)
- GSS_SPKM_INTEG_C_QOP_DSA_SHA1 - DSS signature using a SHA-1 checksum (uses X.509 certificate)

An alternative to specifying an integrity algorithm is to specify a generic integrity level. The security mechanism will select an integrity algorithm which meets the requirements of the specified generic level. The SPKM generic integrity levels are:

- GSS_SPKM_INT_ALG_NON_REP_SUPPORT - Non-repudiable signature (uses X.509 certificate)
- GSS_SPKM_INT_ALG_REPUDIABLE - Negotiated key used to generate integrity checksum

The SPKM confidentiality algorithms are:

- GSS_SPKM_CONF_C_QOP_DEFAULT - Default confidentiality algorithm (128-bit CAST5)
- GSS_SPKM_CONF_C_QOPT_CAST5 - 128-bit CAST5 encryption
- GSS_SPKM_CONF_C_QOPT_DES - 56-bit DES encryption

An alternative to specifying a confidentiality algorithm is to specify a generic confidentiality level. The security mechanism will select a confidentiality algorithm which meets the requirements of the specified generic level. The SPKM generic confidentiality levels are:

- GSS_SPKM_SYM_ALG_STRENGTH_STRONG - The encryption key is 80 bits or greater
- GSS_SPKM_SYM_ALG_STRENGTH_MEDIUM - The encryption key is between 40 and 80 bits
- GSS_SPKM_SYM_ALG_STRENGTH_WEAK - The encryption key is 40 bits or less

The default algorithms can be requested by specifying GSS_C_QOP_DEFAULT, which is equivalent to specifying GSS_SPKM_INTEG_C_QOP_DEFAULT | GSS_SPKM_CONF_C_QOPT_DEFAULT. An anonymous initiator cannot use an X.509 signature since there is no source certificate.

LIPKEY mechanism

The LIPKEY security mechanism uses the SPKM security mechanism for integrity and confidentiality processing. The LIPKEY QOP values are thus the same as the SPKM QOP values. The initiator cannot use an X.509 signature since there is no source certificate.

Status Codes

Table 70. Status Codes for **gss_wrap()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_QOP	The quality of protection value is not valid.
GSS_S_CONTEXT_EXPIRED	The context referred to has expired.
GSS_S_CREDENTIALS_EXPIRED	The credentials associated with the context referred to have expired.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The context referred to is not valid.

gss_wrap_size_limit (determine the largest message)

Purpose

Determines that largest message that can be wrapped without exceeding a maximum size limit.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_wrap_size_limit (
    OM_uint32 *            minor_status,
    gss_ctx_id_t           context_handle,
    int                    conf_req,
    gss_qop_t              qop_req,
    OM_uint32              tsize_req,

    OM_uint32 *            max_size)
```

Parameters

Input

context_handle
Specifies the security context associated with the messages.

conf_req
Specifies whether confidentiality services are requested for the messages as follows:

- TRUE - Confidentiality services are requested in addition to integrity and authentication services.
- FALSE - Only integrity and authentication services are requested.

qop_req
Specifies the quality of protection to be used for the messages. Specify GSS_C_QOP_DEFAULT to use the default quality of protection as defined by the selected security mechanism.

GSS_C_QOP_DEFAULT should always be specified unless it is necessary to select a specific quality-of-protection algorithm, in which case the application must ensure that the selected algorithm is compatible with the security mechanism associated with the security context. The quality of protection value is formed by or'ing together one of the integrity algorithm values and one of the sealing algorithm values. For more information on integrity and sealing algorithm values, see your selected security mechanism in the Usage section of the **gss_wrap** call.

size_req
Specifies the maximum output token size.

Output

max_size
Returns the maximum message size that can be processed without exceeding the specified maximum token size.

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_wrap_size_limit()** routine returns the maximum input message size that can be processed by the **gss_wrap()** routine without exceeding the specified output token size.

For a list of supported mechanisms, see "gss_wrap (sign and encrypt a message)" on page 246.

Status Codes

Table 71. Status Codes for **gss_wrap_size_limit()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_BAD_QOP	The quality of protection requested is not valid.

Table 71. Status Codes for *gss_wrap_size_limit()* (continued)	
Status Code	**Meaning**
GSS_S_CONTEXT_EXPIRED	The context referred to has expired.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The context referred to does not exist.

Chapter 6. GSS-API programming interfaces - Kerberos mechanism

gss_krb5_acquire_cred_ccache (acquire a GSS-API credential)

Purpose

Acquires a GSS-API credential using a Kerberos credentials cache.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_krb5_acquire_cred_ccache (
    OM_uint32 *                         minor_status,
    krb5_ccache                         ccache,
    OM_uint32                           time_req,
    gss_cred_usage_t                    cred_usage,
    gss_cred_id_t *                     output_cred_handle,
    OM_uint32 *                         time_rec)
```

Parameters

Input

ccache
Specifies the Kerberos credentials cache to be used for the credential. The principal name for the GSS-API credential is obtained from the credentials cache. The credentials cache must contain a valid ticket-granting ticket for this principal if a GSS_C_INITIATE or GSS_C_BOTH credential is requested.

time_req
Specifies the number of seconds that the credential remains valid. Specify GSS_C_INDEFINITE to request the maximum credential lifetime. Specify zero for the default lifetime of 2 hours. The actual credential lifetime is limited by the lifetime of the underlying ticket-granting ticket for GSS_C_INITIATE and GSS_C_BOTH credentials.

cred_usage
Specifies the desired credential usage as follows:

- GSS_C_INITIATE if the credential can be used only to initiate security contexts.
- GSS_C_ACCEPT if the credential can be used only to accept security contexts.
- GSS_C_BOTH if the credential can be used to both initiate and accept security contexts.

Output

output_cred_handle
Returns the handle for the GSS-API credential.

time_rec
Returns the number of seconds the credential remains valid. If the time remaining is not required, specify NULL for this parameter.

minor_status
Returns a status code from the security mechanism.

Usage

The gss_krb5_acquire_cred_ccache() routine allows an application to obtain a GSS-API credential for use with the Kerberos mechanism. The application can then use the credential with the gss_accept_sec_context() and gss_accept_sec_context() routines. The Kerberos credentials cache must not be closed until the GSS-API credential is no longer needed and has been deleted.

If GSS_C_INITIATE or GSS_C_BOTH is specified for the credential usage, the application must have a valid ticket in the credentials cache and the ticket must not expire for at least 10 minutes. The gss_krb5_acquire_cred_ccache() routine uses the first valid ticket-granting ticket (or the first valid service ticket if there is no TGT) to create the GSS-API credential.

If GSS_C_ACCEPT or GSS_C_BOTH is specified for the credential usage, the principal associated with the GSS-API credential must be defined in a key table. The KRB5_KTNAME environment variable is used to identify the key table used by the Kerberos security mechanism.

If the Kerberos security server is running on the same system as the application, it is not necessary to have a key table for GSS_C_ACCEPT or GSS_C_BOTH credentials. Instead, GSS-API uses the local instance of the Kerberos security server to decrypt the ticket. In order to activate this support, the KRB5_SERVER_KEYTAB environment variable needs to be set to one of the following values, and depending on the value set, other requirements must also be met:

1. If the KRB5_SERVER_KEYTAB environment variable is set to 1:

   a. The application must be running with a user or group that has at least READ access to the IRR.RUSERMAP resource in the FACILITY class.

   b. The Kerberos principal associated with the current system identity must match the principal for the GSSAPI credential.

2. If the KRB5_SERVER_KEYTAB environment variable is set to 2:

   a. No requirements – processing is done during a gss_accept_sec_context()call.

Status Codes

Table 72. Status Codes for **gss_krb5_acquire_cred_ccache()**	
Status Code	**Meaning**
GSS_S_BAD_MECH	None of the requested mechanisms are supported by the local system.
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CRED	The Kerberos credentials cache does not contain a valid ticket-granting ticket.

gss_krb5_ccache_name (set the default credentials cache name)

Purpose

Sets the default credentials cache name for use by the Kerberos mechanism.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_krb5_ccache_name (
    OM_uint32 *                          minor_status,
    char *                               new_name,
    char **                              old_name)
```

Parameters

Input

new_name
   Specifies the new name for the default GSS-API Kerberos credentials cache.

Output

old_name
   Returns the name of the current default credentials cache or NULL if the default credentials cache has not been set. Specify NULL for this parameter if you do not need the current credentials cache name. The returned name should be released by calling **krb5_free_string()** when it is no longer needed.

minor_status
   Returns a status code from the security mechanism.

Usage

The **gss_krb5_ccache_name()** routine sets the default credentials cache name for use by the Kerberos mechanism. The default credentials cache is used by **gss_acquire_cred()** to create a GSS-API credential. It is also used by **gss_init_sec_context()** when GSS_C_NO_CREDENTIAL is specified for the GSS-API credential used to establish the security context.

Status Codes

Table 73. Status Codes for *gss_krb5_ccache_name()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.

gss_krb5_copy_ccache (copy the credentials cache tickets)

Purpose

Copies the tickets from the Kerberos credentials cache associated with a GSS-API credential.

Format

```
#include <skrb/gssapi.h>
OM_uint32 gss_krb5_copy_ccache (
    OM_uint32 *                     minor_status,
    gss_cred_id_t                   cred_handle,
    krb5_ccache                     ccache)
```

Parameters

Input

cred_handle
Specifies the GSS-API credential handle. This must be a GSS_C_INITIATE or GSS_C_BOTH credential.

ccache
Specifies the Kerberos credentials cache.

Output

minor_status
Returns a status code from the security mechanism.

Usage

The **gss_krb5_copy_ccache()** routine copies the tickets from the Kerberos credentials cache associated with a GSS-API credential to a credentials cache provided by the caller. The supplied Kerberos credentials cache must have been initialized by **krb5_cc_initialize()** before calling **gss_krb5_copy_ccache()**. The GSS-API credential must have been created by specifying GSS_C_INITIATE or GSS_C_BOTH.

Status Codes

Table 74. Status Codes for **gss_krb5_copy_ccache()**	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CRED	The credential handle does not refer to a valid GSS-API credential.

gss_krb5_get_ccache (return the credentials cache)

Purpose

Returns the Kerberos credentials cache associated with a GSSAPI credential.

Format

```
#include <skrb/gssapi.h>

OM_uint32 gss_krb5_get_ccache (
    OM_uint32 *              minor_status,
    gss_cred_id_t            cred_handle,
    krb5_ccache *            ccache)
```

Parameters

Input

cred_handle
    Specifies the handle for the GSSAPI credential.

Output

ccache
    Returns the handle for the credentials cache. A NULL value is returned if there is no credentials cache associated with the GSSAPI credential.

minor_status
    Returns a status code from the security mechanism.

Usage

The **gss_krb5_get_ccache()** routine returns the handle for the credentials cache that is associated with the GSSAPI credential. The application must not close nor destroy this credentials cache. The returned handle is no longer valid once the GSSAPI credential has been released.

Status Codes

Table 75. Status Codes for *gss_krb5_get_ccache()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CRED	The credential handle does not refer to a valid GSSAPI credential.

gss_krb5_get_tkt_flags (return the ticket flags)

Purpose

Returns the Kerberos ticket flags from the service ticket.

Format

```
#include <skrb/gssapi.h>
om_uint32 gss_krb5_get_tkt_flags (
    om_uint32 *                     minor_status,
    gss_ctx_id_t                    context_handle,
    krb5_flags *                    tkt_flags)
```

Parameters

Input

context_handle
    Specifies the handle for the GSSAPI security context.

Output

tkt_flags
    Returns the ticket flags from the Kerberos ticket associated with the security context.

minor_status
    Returns a status code from the security mechanism.

Usage

The **gss_krb5_get_tkt_flags()** routine returns the ticket flags from the Kerberos ticket associated with the security context. Refer to the Kerberos API documentation for a description of the various flags.

Status Codes

Table 76. Status Codes for *gss_krb5_get_tkt_flags()*	
Status Code	**Meaning**
GSS_S_COMPLETE	The routine completed successfully.

Table 76. Status Codes for **gss_krb5_get_tkt_flags()** (continued)	
Status Code	**Meaning**
GSS_S_FAILURE	The routine failed for reasons that are not defined at the GSS level. The *minor_status* return parameter contains a mechanism-dependent error code describing the reason for the failure.
GSS_S_NO_CONTEXT	The context handle does not refer to a valid security context.

Appendix A. POSIX-based portable character set

The following table presents the POSIX-based portable character set.

Table 77. POSIX-based portable character set	
Contents	**Character**
<space>	
<exclamation-mark>	!
<quotation-mark>	"
<number-sign>	#
<dollar-sign>	$
<percent-sign>	%
<ampersand>	&
<apostrophe>	'
<left-parenthesis>	(
<right-parenthesis>	)
<asterisk>	*
<plus-sign>	+
<comma>	,
<hyphen>	-
<colon>	:
<semi-colon>	;
<period>	.
<slash>	/
<back-slash>	\
<less-than>	<
<equal-to>	=
<greater-than>	>
<question-mark>	?
<commercial-at>	@
<left-square-bracket>	[
<right-square-bracket>	]
<left-brace>	{
<right-brace>	}
<circumflex>	^
<underscore>	_

Table 77. POSIX-based portable character set (continued)	
Contents	**Character**
<grave-accent>	`
<tilde>	~
<vertical-bar>	|
<zero>	0
<one>	1
<two>	2
<three>	3
<four>	4
<five>	5
<six>	6
<seven>	7
<eight>	8
<nine>	9
<A>	A
<B>	B
<C>	C
<D>	D
<E>	E
<F>	F
<G>	G
<H>	H
<I>	I
<J>	J
<K>	K
<L>	L
<M>	M
<N>	N
<O>	O
<P>	P
<Q>	Q
<R>	R
<S>	S
<T>	T
<U>	U

Contents	Character
Table 77. POSIX-based portable character set (continued)	
Contents	**Character**
<V>	V
<W>	W
<X>	X
<Y>	Y
<Z>	Z
<a>	a
<b>	b
<c>	c
<d>	d
<e>	e
<f>	f
<g>	g
<h>	h
<i>	i
<j>	j
<k>	k
<l>	l
<m>	m
<n>	n
<o>	o
<p>	p
<q>	q
<r>	r
<s>	s
<t>	t
<u>	u
<v>	v
<w>	w
<x>	x
<y>	y
<z>	z

Appendix B. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Documentation with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1)

are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

* indicates an optional syntax element that is repeatable
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included
The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Index

IBM

Product Number: 5650-ZOS