Kerberos Application Developer Guide

Release 1.18.5

MIT

CONTENTS

CHAPTER

ONE

DEVELOPING WITH GSSAPI

The GSSAPI (Generic Security Services API) allows applications to communicate securely using Kerberos 5 or other security mechanisms. We recommend using the GSSAPI (or a higher-level framework which encompasses GSSAPI, such as SASL) for secure network communication over using the libkrb5 API directly.

GSSAPIv2 is specified in **RFC 2743** and **RFC 2744**. Also see **RFC 7546** for a description of how to use the GSSAPI in a client or server program.

This documentation will describe how various ways of using the GSSAPI will behave with the krb5 mechanism as implemented in MIT krb5, as well as krb5-specific extensions to the GSSAPI.

1.1 Name types

A GSSAPI application can name a local or remote entity by calling gss_import_name, specifying a name type and a value. The following name types are supported by the krb5 mechanism:

- **GSS_C_NT_HOSTBASED_SERVICE**: The value should be a string of the form `service` or `service@hostname`. This is the most common way to name target services when initiating a security context, and is the most likely name type to work across multiple mechanisms.
- **GSS_KRB5_NT_PRINCIPAL_NAME**: The value should be a principal name string. This name type only works with the krb5 mechanism, and is defined in the `<gssapi/gssapi_krb5.h>` header.
- **GSS_C_NT_USER_NAME** or **GSS_C_NULL_OID**: The value is treated as an unparsed principal name string, as above. These name types may work with mechanisms other than krb5, but will have different interpretations in those mechanisms. **GSS_C_NT_USER_NAME** is intended to be used with a local username, which will parse into a single-component principal in the default realm.
- **GSS_C_NT_ANONYMOUS**: The value is ignored. The anonymous principal is used, allowing a client to authenticate to a server without asserting a particular identity (which may or may not be allowed by a particular server or Kerberos realm).
- **GSS_C_NT_MACHINE_UID_NAME**: The value is uid_t object. On Unix-like systems, the username of the uid is looked up in the system user database and the resulting username is parsed as a principal name.
- **GSS_C_NT_STRING_UID_NAME**: As above, but the value is a decimal string representation of the uid.
- **GSS_C_NT_EXPORT_NAME**: The value must be the result of a gss_export_name call.
- **GSS_KRB5_NT_ENTERPRISE_NAME**: The value should be a krb5 enterprise name string (see **RFC 6806** section 5), in the form `user@suffix`. This name type is used to convey alias names, and is defined in the `<gssapi/gssapi_krb5.h>` header. (New in release 1.17.)

1.2 Initiator credentials

A GSSAPI client application uses gss_init_sec_context to establish a security context. The *initiator_cred_handle* parameter determines what tickets are used to establish the connection. An application can either pass **GSS_C_NO_CREDENTIAL** to use the default client credential, or it can use gss_acquire_cred beforehand to acquire an initiator credential. The call to gss_acquire_cred may include a *desired_name* parameter, or it may pass **GSS_C_NO_NAME** if it does not have a specific name preference.

If the desired name for a krb5 initiator credential is a host-based name, it is converted to a principal name of the form `service/hostname` in the local realm, where *hostname* is the local hostname if not specified. The hostname will be canonicalized using forward name resolution, and possibly also using reverse name resolution depending on the value of the **rdns** variable in libdefaults.

If a desired name is specified in the call to gss_acquire_cred, the krb5 mechanism will attempt to find existing tickets for that client principal name in the default credential cache or collection. If the default cache type does not support a collection, and the default cache contains credentials for a different principal than the desired name, a **GSS_S_CRED_UNAVAIL** error will be returned with a minor code indicating a mismatch.

If no existing tickets are available for the desired name, but the name has an entry in the default client keytab_definition, the krb5 mechanism will acquire initial tickets for the name using the default client keytab.

If no desired name is specified, credential acquisition will be deferred until the credential is used in a call to gss_init_sec_context or gss_inquire_cred. If the call is to gss_init_sec_context, the target name will be used to choose a client principal name using the credential cache selection facility. (This facility might, for instance, try to choose existing tickets for a client principal in the same realm as the target service). If there are no existing tickets for the chosen principal, but it is present in the default client keytab, the krb5 mechanism will acquire initial tickets using the keytab.

If the target name cannot be used to select a client principal (because the credentials are used in a call to gss_inquire_cred), or if the credential cache selection facility cannot choose a principal for it, the default credential cache will be selected if it exists and contains tickets.

If the default credential cache does not exist, but the default client keytab does, the krb5 mechanism will try to acquire initial tickets for the first principal in the default client keytab.

If the krb5 mechanism acquires initial tickets using the default client keytab, the resulting tickets will be stored in the default cache or collection, and will be refreshed by future calls to gss_acquire_cred as they approach their expire time.

1.3 Acceptor names

A GSSAPI server application uses gss_accept_sec_context to establish a security context based on tokens provided by the client. The *acceptor_cred_handle* parameter determines what keytab_definition entries may be authenticated to by the client, if the krb5 mechanism is used.

The simplest choice is to pass **GSS_C_NO_CREDENTIAL** as the acceptor credential. In this case, clients may authenticate to any service principal in the default keytab (typically DEFKTNAME, or the value of the **KRB5_KTNAME** environment variable). This is the recommended approach if the server application has no specific requirements to the contrary.

A server may acquire an acceptor credential with gss_acquire_cred and a *cred_usage* of **GSS_C_ACCEPT** or **GSS_C_BOTH**. If the *desired_name* parameter is **GSS_C_NO_NAME**, then clients will be allowed to authenticate to any service principal in the default keytab, just as if no acceptor credential was supplied.

If a server wishes to specify a *desired_name* to gss_acquire_cred, the most common choice is a host-based name. If the host-based *desired_name* contains just a *service*, then clients will be allowed to authenticate to any host-based service principal (that is, a principal of the form `service/hostname@REALM`) for the named service, regardless

of hostname or realm, as long as it is present in the default keytab. If the input name contains both a *service* and a *hostname*, clients will be allowed to authenticate to any host-based principal for the named service and hostname, regardless of realm.

Note: If a *hostname* is specified, it will be canonicalized using forward name resolution, and possibly also using reverse name resolution depending on the value of the **rdns** variable in libdefaults.

Note: If the **ignore_acceptor_hostname** variable in libdefaults is enabled, then *hostname* will be ignored even if one is specified in the input name.

Note: In MIT krb5 versions prior to 1.10, and in Heimdal's implementation of the krb5 mechanism, an input name with just a *service* is treated like an input name of `service@localhostname`, where *localhostname* is the string returned by gethostname().

If the *desired_name* is a krb5 principal name or a local system name type which is mapped to a krb5 principal name, clients will only be allowed to authenticate to that principal in the default keytab.

1.4 Name Attributes

In release 1.8 or later, the gss_inquire_name and gss_get_name_attribute functions, specified in **RFC 6680**, can be used to retrieve name attributes from the *src_name* returned by gss_accept_sec_context. The following attributes are defined when the krb5 mechanism is used:

- "auth-indicators" attribute:

This attribute will be included in the gss_inquire_name output if the ticket contains authentication indicators. One indicator is returned per invocation of gss_get_name_attribute, so multiple invocations may be necessary to retrieve all of the indicators from the ticket. (New in release 1.15.)

1.5 Importing and exporting credentials

The following GSSAPI extensions can be used to import and export credentials (declared in `<gssapi/gssapi_ext.h>`):

```
OM_uint32 gss_export_cred(OM_uint32 *minor_status,
                          gss_cred_id_t cred_handle,
                          gss_buffer_t token);

OM_uint32 gss_import_cred(OM_uint32 *minor_status,
                          gss_buffer_t token,
                          gss_cred_id_t *cred_handle);
```

The first function serializes a GSSAPI credential handle into a buffer; the second unseralizes a buffer into a GSSAPI credential handle. Serializing a credential does not destroy it. If any of the mechanisms used in *cred_handle* do not support serialization, gss_export_cred will return **GSS_S_UNAVAILABLE**. As with other GSSAPI serialization functions, these extensions are only intended to work with a matching implementation on the other side; they do not serialize credentials in a standardized format.

A serialized credential may contain secret information such as ticket session keys. The serialization format does not protect this information from eavesdropping or tampering. The calling application must take care to protect the serialized credential when communicating it over an insecure channel or to an untrusted party.

A krb5 GSSAPI credential may contain references to a credential cache, a client keytab, an acceptor keytab, and a replay cache. These resources are normally serialized as references to their external locations (such as the filename of the credential cache). Because of this, a serialized krb5 credential can only be imported by a process with similar privileges to the exporter. A serialized credential should not be trusted if it originates from a source with lower privileges than the importer, as it may contain references to external credential cache, keytab, or replay cache resources not accessible to the originator.

An exception to the above rule applies when a krb5 GSSAPI credential refers to a memory credential cache, as is normally the case for delegated credentials received by gss_accept_sec_context. In this case, the contents of the credential cache are serialized, so that the resulting token may be imported even if the original memory credential cache no longer exists.

1.6 Constrained delegation (S4U)

The Microsoft S4U2Self and S4U2Proxy Kerberos protocol extensions allow an intermediate service to acquire credentials from a client to a target service without requiring the client to delegate a ticket-granting ticket, if the KDC is configured to allow it.

To perform a constrained delegation operation, the intermediate service must submit to the KDC an "evidence ticket" from the client to the intermediate service. An evidence ticket can be acquired when the client authenticates to the intermediate service with Kerberos, or with an S4U2Self request if the KDC allows it. The MIT krb5 GSSAPI library represents an evidence ticket using a "proxy credential", which is a special kind of gss_cred_id_t object whose underlying credential cache contains the evidence ticket and a krbtgt ticket for the intermediate service.

To acquire a proxy credential during client authentication, the service should first create an acceptor credential using the **GSS_C_BOTH** usage. The application should then pass this credential as the *acceptor_cred_handle* to gss_accept_sec_context, and also pass a *delegated_cred_handle* output parameter to receive a proxy credential containing the evidence ticket. The output value of *delegated_cred_handle* may be a delegated ticket-granting ticket if the client sent one, or a proxy credential if not. If the library can determine that the client's ticket is not a valid evidence ticket, it will place **GSS_C_NO_CREDENTIAL** in *delegated_cred_handle*.

To acquire a proxy credential using an S4U2Self request, the service can use the following GSSAPI extension:

```
OM_uint32 gss_acquire_cred_impersonate_name(OM_uint32 *minor_status,
                                            gss_cred_id_t icred,
                                            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,
                                            gss_OID_set *actual_mechs,
                                            OM_uint32 *time_rec);
```

The parameters to this function are similar to those of gss_acquire_cred, except that *icred* is used to make an S4U2Self request to the KDC for a ticket from *desired_name* to the intermediate service. Both *icred* and *desired_name* are required for this function; passing **GSS_C_NO_CREDENTIAL** or **GSS_C_NO_NAME** will cause the call to fail. *icred* must contain a krbtgt ticket for the intermediate service. The result of this operation is a proxy credential. (Prior to release 1.18, the result of this operation may be a regular credential for *desired_name*, if the KDC issues a non-forwardable ticket.)

Once the intermediate service has a proxy credential, it can simply pass it to gss_init_sec_context as the *initiator_cred_handle* parameter, and the desired service as the *target_name* parameter. The GSSAPI library will present

the krbtgt ticket and evidence ticket in the proxy credential to the KDC in an S4U2Proxy request; if the intermediate service has the appropriate permissions, the KDC will issue a ticket from the client to the target service. The GSSAPI library will then use this ticket to authenticate to the target service.

If an application needs to find out whether a credential it holds is a proxy credential and the name of the intermediate service, it can query the credential with the **GSS_KRB5_GET_CRED_IMPERSONATOR** OID (new in release 1.16, declared in <gssapi/gssapi_krb5.h>) using the gss_inquire_cred_by_oid extension (declared in <gssapi/gssapi_ext.h>):

```
OM_uint32 gss_inquire_cred_by_oid(OM_uint32 *minor_status,
                                  const gss_cred_id_t cred_handle,
                                  gss_OID desired_object,
                                  gss_buffer_set_t *data_set);
```

If the call succeeds and *cred_handle* is a proxy credential, *data_set* will be set to a single-element buffer set containing the unparsed principal name of the intermediate service. If *cred_handle* is not a proxy credential, *data_set* will be set to an empty buffer set. If the library does not support the query, gss_inquire_cred_by_oid will return **GSS_S_UNAVAILABLE**.

1.7 AEAD message wrapping

The following GSSAPI extensions (declared in <gssapi/gssapi_ext.h>) can be used to wrap and unwrap messages with additional "associated data" which is integrity-checked but is not included in the output buffer:

```
OM_uint32 gss_wrap_aead(OM_uint32 *minor_status,
                        gss_ctx_id_t context_handle,
                        int conf_req_flag, gss_qop_t qop_req,
                        gss_buffer_t input_assoc_buffer,
                        gss_buffer_t input_payload_buffer,
                        int *conf_state,
                        gss_buffer_t output_message_buffer);

OM_uint32 gss_unwrap_aead(OM_uint32 *minor_status,
                          gss_ctx_id_t context_handle,
                          gss_buffer_t input_message_buffer,
                          gss_buffer_t input_assoc_buffer,
                          gss_buffer_t output_payload_buffer,
                          int *conf_state,
                          gss_qop_t *qop_state);
```

Wrap tokens created with gss_wrap_aead will successfully unwrap only if the same *input_assoc_buffer* contents are presented to gss_unwrap_aead.

1.8 IOV message wrapping

The following extensions (declared in <gssapi/gssapi_ext.h>) can be used for in-place encryption, fine-grained control over wrap token layout, and for constructing wrap tokens compatible with Microsoft DCE RPC:

```
typedef struct gss_iov_buffer_desc_struct {
    OM_uint32 type;
    gss_buffer_desc buffer;
} gss_iov_buffer_desc, *gss_iov_buffer_t;

OM_uint32 gss_wrap_iov(OM_uint32 *minor_status,
```

```
                        gss_ctx_id_t context_handle,
                        int conf_req_flag, gss_qop_t qop_req,
                        int *conf_state,
                        gss_iov_buffer_desc *iov, int iov_count);

OM_uint32 gss_unwrap_iov(OM_uint32 *minor_status,
                         gss_ctx_id_t context_handle,
                         int *conf_state, gss_qop_t *qop_state,
                         gss_iov_buffer_desc *iov, int iov_count);

OM_uint32 gss_wrap_iov_length(OM_uint32 *minor_status,
                              gss_ctx_id_t context_handle,
                              int conf_req_flag,
                              gss_qop_t qop_req, int *conf_state,
                              gss_iov_buffer_desc *iov,
                              int iov_count);

OM_uint32 gss_release_iov_buffer(OM_uint32 *minor_status,
                                 gss_iov_buffer_desc *iov,
                                 int iov_count);
```

The caller of gss_wrap_iov provides an array of gss_iov_buffer_desc structures, each containing a type and a gss_buffer_desc structure. Valid types include:

- **GSS_C_BUFFER_TYPE_DATA**: A data buffer to be included in the token, and to be encrypted or decrypted in-place if the token is confidentiality-protected.
- **GSS_C_BUFFER_TYPE_HEADER**: The GSSAPI wrap token header and underlying cryptographic header.
- **GSS_C_BUFFER_TYPE_TRAILER**: The cryptographic trailer, if one is required.
- **GSS_C_BUFFER_TYPE_PADDING**: Padding to be combined with the data during encryption and decryption. (The implementation may choose to place padding in the trailer buffer, in which case it will set the padding buffer length to 0.)
- **GSS_C_BUFFER_TYPE_STREAM**: For unwrapping only, a buffer containing a complete wrap token in standard format to be unwrapped.
- **GSS_C_BUFFER_TYPE_SIGN_ONLY**: A buffer to be included in the token's integrity protection checksum, but not to be encrypted or included in the token itself.

For gss_wrap_iov, the IOV list should contain one HEADER buffer, followed by zero or more SIGN_ONLY buffers, followed by one or more DATA buffers, followed by a TRAILER buffer. The memory pointed to by the buffers is not required to be contiguous or in any particular order. If *conf_req_flag* is true, DATA buffers will be encrypted in-place, while SIGN_ONLY buffers will not be modified.

The type of an output buffer may be combined with **GSS_C_BUFFER_FLAG_ALLOCATE** to request that gss_wrap_iov allocate the buffer contents. If gss_wrap_iov allocates a buffer, it sets the **GSS_C_BUFFER_FLAG_ALLOCATED** flag on the buffer type. gss_release_iov_buffer can be used to release all allocated buffers within an iov list and unset their allocated flags. Here is an example of how gss_wrap_iov can be used with allocation requested (*ctx* is assumed to be a previously established gss_ctx_id_t):

```
OM_uint32 major, minor;
gss_iov_buffer_desc iov[4];
char str[] = "message";

iov[0].type = GSS_IOV_BUFFER_TYPE_HEADER | GSS_IOV_BUFFER_FLAG_ALLOCATE;
iov[1].type = GSS_IOV_BUFFER_TYPE_DATA;
iov[1].buffer.value = str;
iov[1].buffer.length = strlen(str);
```

```
iov[2].type = GSS_IOV_BUFFER_TYPE_PADDING | GSS_IOV_BUFFER_FLAG_ALLOCATE;
iov[3].type = GSS_IOV_BUFFER_TYPE_TRAILER | GSS_IOV_BUFFER_FLAG_ALLOCATE;

major = gss_wrap_iov(&minor, ctx, 1, GSS_C_QOP_DEFAULT, NULL,
                     iov, 4);
if (GSS_ERROR(major))
    handle_error(major, minor);

/* Transmit or otherwise use resulting buffers. */

(void)gss_release_iov_buffer(&minor, iov, 4);
```

If the caller does not choose to request buffer allocation by gss_wrap_iov, it should first call gss_wrap_iov_length to query the lengths of the HEADER, PADDING, and TRAILER buffers. DATA buffers must be provided in the iov list so that padding length can be computed correctly, but the output buffers need not be initialized. Here is an example of using gss_wrap_iov_length and gss_wrap_iov:

```
OM_uint32 major, minor;
gss_iov_buffer_desc iov[4];
char str[1024] = "message", *ptr;

iov[0].type = GSS_IOV_BUFFER_TYPE_HEADER;
iov[1].type = GSS_IOV_BUFFER_TYPE_DATA;
iov[1].buffer.value = str;
iov[1].buffer.length = strlen(str);

iov[2].type = GSS_IOV_BUFFER_TYPE_PADDING;
iov[3].type = GSS_IOV_BUFFER_TYPE_TRAILER;

major = gss_wrap_iov_length(&minor, ctx, 1, GSS_C_QOP_DEFAULT,
                            NULL, iov, 4);
if (GSS_ERROR(major))
    handle_error(major, minor);
if (strlen(str) + iov[0].buffer.length + iov[2].buffer.length +
    iov[3].buffer.length > sizeof(str))
    handle_out_of_space_error();
ptr = str + strlen(str);
iov[0].buffer.value = ptr;
ptr += iov[0].buffer.length;
iov[2].buffer.value = ptr;
ptr += iov[2].buffer.length;
iov[3].buffer.value = ptr;

major = gss_wrap_iov(&minor, ctx, 1, GSS_C_QOP_DEFAULT, NULL,
                     iov, 4);
if (GSS_ERROR(major))
    handle_error(major, minor);
```

If the context was established using the **GSS_C_DCE_STYLE** flag (described in RFC 4757), wrap tokens compatible with Microsoft DCE RPC can be constructed. In this case, the IOV list must include a SIGN_ONLY buffer, a DATA buffer, a second SIGN_ONLY buffer, and a HEADER buffer in that order (the order of the buffer contents remains arbitrary). The application must pad the DATA buffer to a multiple of 16 bytes as no padding or trailer buffer is used.

gss_unwrap_iov may be called with an IOV list just like one which would be provided to gss_wrap_iov. DATA buffers will be decrypted in-place if they were encrypted, and SIGN_ONLY buffers will not be modified.

Alternatively, gss_unwrap_iov may be called with a single STREAM buffer, zero or more SIGN_ONLY buffers, and a single DATA buffer. The STREAM buffer is interpreted as a complete wrap token. The STREAM buffer will be

modified in-place to decrypt its contents. The DATA buffer will be initialized to point to the decrypted data within the STREAM buffer, unless it has the **GSS_C_BUFFER_FLAG_ALLOCATE** flag set, in which case it will be initialized with a copy of the decrypted data. Here is an example (*token* and *token_len* are assumed to be a pre-existing pointer and length for a modifiable region of data):

```
OM_uint32 major, minor;
gss_iov_buffer_desc iov[2];

iov[0].type = GSS_IOV_BUFFER_TYPE_STREAM;
iov[0].buffer.value = token;
iov[0].buffer.length = token_len;
iov[1].type = GSS_IOV_BUFFER_TYPE_DATA;
major = gss_unwrap_iov(&minor, ctx, NULL, NULL, iov, 2);
if (GSS_ERROR(major))
    handle_error(major, minor);

/* Decrypted data is in iov[1].buffer, pointing to a subregion of
 * token. */
```

1.9 IOV MIC tokens

The following extensions (declared in <gssapi/gssapi_ext.h>) can be used in release 1.12 or later to construct and verify MIC tokens using an IOV list:

```
OM_uint32 gss_get_mic_iov(OM_uint32 *minor_status,
                          gss_ctx_id_t context_handle,
                          gss_qop_t qop_req,
                          gss_iov_buffer_desc *iov,
                          int iov_count);

OM_uint32 gss_get_mic_iov_length(OM_uint32 *minor_status,
                                 gss_ctx_id_t context_handle,
                                 gss_qop_t qop_req,
                                 gss_iov_buffer_desc *iov,
                                 iov_count);

OM_uint32 gss_verify_mic_iov(OM_uint32 *minor_status,
                             gss_ctx_id_t context_handle,
                             gss_qop_t *qop_state,
                             gss_iov_buffer_desc *iov,
                             int iov_count);
```

The caller of gss_get_mic_iov provides an array of gss_iov_buffer_desc structures, each containing a type and a gss_buffer_desc structure. Valid types include:

- **GSS_C_BUFFER_TYPE_DATA** and **GSS_C_BUFFER_TYPE_SIGN_ONLY**: The corresponding buffer for each of these types will be signed for the MIC token, in the order provided.
- **GSS_C_BUFFER_TYPE_MIC_TOKEN**: The GSSAPI MIC token.

The type of the MIC_TOKEN buffer may be combined with **GSS_C_BUFFER_FLAG_ALLOCATE** to request that gss_get_mic_iov allocate the buffer contents. If gss_get_mic_iov allocates the buffer, it sets the **GSS_C_BUFFER_FLAG_ALLOCATED** flag on the buffer type. gss_release_iov_buffer can be used to release all allocated buffers within an iov list and unset their allocated flags. Here is an example of how gss_get_mic_iov can be used with allocation requested (*ctx* is assumed to be a previously established gss_ctx_id_t):

```
OM_uint32 major, minor;
gss_iov_buffer_desc iov[3];

iov[0].type = GSS_IOV_BUFFER_TYPE_DATA;
iov[0].buffer.value = "sign1";
iov[0].buffer.length = 5;
iov[1].type = GSS_IOV_BUFFER_TYPE_SIGN_ONLY;
iov[1].buffer.value = "sign2";
iov[1].buffer.length = 5;
iov[2].type = GSS_IOV_BUFFER_TYPE_MIC_TOKEN | GSS_IOV_BUFFER_FLAG_ALLOCATE;

major = gss_get_mic_iov(&minor, ctx, GSS_C_QOP_DEFAULT, iov, 3);
if (GSS_ERROR(major))
    handle_error(major, minor);

/* Transmit or otherwise use iov[2].buffer. */

(void)gss_release_iov_buffer(&minor, iov, 3);
```

If the caller does not choose to request buffer allocation by gss_get_mic_iov, it should first call gss_get_mic_iov_length to query the length of the MIC_TOKEN buffer. Here is an example of using gss_get_mic_iov_length and gss_get_mic_iov:

```
OM_uint32 major, minor;
gss_iov_buffer_desc iov[2];
char data[1024];

iov[0].type = GSS_IOV_BUFFER_TYPE_MIC_TOKEN;
iov[1].type = GSS_IOV_BUFFER_TYPE_DATA;
iov[1].buffer.value = "message";
iov[1].buffer.length = 7;

major = gss_get_mic_iov_length(&minor, ctx, GSS_C_QOP_DEFAULT, iov, 2);
if (GSS_ERROR(major))
    handle_error(major, minor);
if (iov[0].buffer.length > sizeof(data))
    handle_out_of_space_error();
iov[0].buffer.value = data;

major = gss_get_mic_iov(&minor, ctx, GSS_C_QOP_DEFAULT, iov, 2);
if (GSS_ERROR(major))
    handle_error(major, minor);
```

CHAPTER

TWO

YEAR 2038 CONSIDERATIONS FOR USES OF KRB5_TIMESTAMP

POSIX time values, which measure the number of seconds since January 1 1970, will exceed the maximum value representable in a signed 32-bit integer in January 2038. This documentation describes considerations for consumers of the MIT krb5 libraries.

Applications or libraries which use libkrb5 and consume the timestamps included in credentials or other structures make use of the *krb5_timestamp* type. For historical reasons, krb5_timestamp is a signed 32-bit integer, even on platforms where a larger type is natively used to represent time values. To behave properly for time values after January 2038, calling code should cast krb5_timestamp values to uint32_t, and then to time_t:

```
(time_t)(uint32_t)timestamp
```

Used in this way, krb5_timestamp values can represent time values up until February 2106, provided that the platform uses a 64-bit or larger time_t type. This usage will also remain safe if a later version of MIT krb5 changes krb5_timestamp to an unsigned 32-bit integer.

The GSSAPI only uses representations of time intervals, not absolute times. Callers of the GSSAPI should require no changes to behave correctly after January 2038, provided that they use MIT krb5 release 1.16 or later.

CHAPTER

THREE

DIFFERENCES BETWEEN HEIMDAL AND MIT KERBEROS API

krb5_auth_con_getaddrs()	H5l: If either of the pointers to local_addr and remote_addr is not NULL, it is freed firs
krb5_auth_con_setaddrs()	H5l: If either address is NULL, the previous address remains in place
krb5_auth_con_setports()	H5l: Not implemented as of version 1.3.3
krb5_auth_con_setrecvsubkey()	H5l: If either port is NULL, the previous port remains in place
krb5_auth_con_setsendsubkey()	H5l: Not implemented as of version 1.3.3
krb5_cc_set_config()	MIT: Before version 1.10 it was assumed that the last argument *data* is ALWAYS non-z
krb5_cccol_last_change_time()	MIT: not implemented
krb5_set_default_realm()	H5l: Caches the computed default realm context field. If the second argument is NULL

CHAPTER

FOUR

INITIAL CREDENTIALS

Software that performs tasks such as logging users into a computer when they type their Kerberos password needs to get initial credentials (usually ticket granting tickets) from Kerberos. Such software shares some behavior with the kinit(1) program.

Whenever a program grants access to a resource (such as a local login session on a desktop computer) based on a user successfully getting initial Kerberos credentials, it must verify those credentials against a secure shared secret (e.g., a host keytab) to ensure that the user credentials actually originate from a legitimate KDC. Failure to perform this verification is a critical vulnerability, because a malicious user can execute the "Zanarotti attack": the user constructs a fake response that appears to come from the legitimate KDC, but whose contents come from an attacker-controlled KDC.

Some applications read a Kerberos password over the network (ideally over a secure channel), which they then verify against the KDC. While this technique may be the only practical way to integrate Kerberos into some existing legacy systems, its use is contrary to the original design goals of Kerberos.

The function *krb5_get_init_creds_password()* will get initial credentials for a client using a password. An application that needs to verify the credentials can call *krb5_verify_init_creds()*. Here is an example of code to obtain and verify TGT credentials, given strings *princname* and *password* for the client principal name and password:

```
krb5_error_code ret;
krb5_creds creds;
krb5_principal client_princ = NULL;

memset(&creds, 0, sizeof(creds));
ret = krb5_parse_name(context, princname, &client_princ);
if (ret)
    goto cleanup;
ret = krb5_get_init_creds_password(context, &creds, client_princ,
                                   password, NULL, NULL, 0, NULL, NULL);
if (ret)
    goto cleanup;
ret = krb5_verify_init_creds(context, &creds, NULL, NULL, NULL, NULL);

cleanup:
krb5_free_principal(context, client_princ);
krb5_free_cred_contents(context, &creds);
return ret;
```

4.1 Options for get_init_creds

The function *krb5_get_init_creds_password()* takes an options parameter (which can be a null pointer). Use the function *krb5_get_init_creds_opt_alloc()* to allocate an options structure, and *krb5_get_init_creds_opt_free()* to free it. For example:

```
krb5_error_code ret;
krb5_get_init_creds_opt *opt = NULL;
krb5_creds creds;

memset(&creds, 0, sizeof(creds));
ret = krb5_get_init_creds_opt_alloc(context, &opt);
if (ret)
    goto cleanup;
krb5_get_init_creds_opt_set_tkt_life(opt, 24 * 60 * 60);
ret = krb5_get_init_creds_password(context, &creds, client_princ,
                                   password, NULL, NULL, 0, NULL, opt);
if (ret)
    goto cleanup;

cleanup:
krb5_get_init_creds_opt_free(context, opt);
krb5_free_cred_contents(context, &creds);
return ret;
```

4.2 Getting anonymous credentials

As of release 1.8, it is possible to obtain fully anonymous or partially anonymous (realm-exposed) credentials, if the KDC supports it. The MIT KDC supports issuing fully anonymous credentials as of release 1.8 if configured appropriately (see anonymous_pkinit), but does not support issuing realm-exposed anonymous credentials at this time.

To obtain fully anonymous credentials, call *krb5_get_init_creds_opt_set_anonymous()* on the options structure to set the anonymous flag, and specify a client principal with the KDC's realm and a single empty data component (the principal obtained by parsing `@`*realmname*). Authentication will take place using anonymous PKINIT; if successful, the client principal of the resulting tickets will be `WELLKNOWN/ANONYMOUS@WELLKNOWN:ANONYMOUS`. Here is an example:

```
krb5_get_init_creds_opt_set_anonymous(opt, 1);
ret = krb5_build_principal(context, &client_princ, strlen(myrealm),
                           myrealm, "", (char *)NULL);
if (ret)
    goto cleanup;
ret = krb5_get_init_creds_password(context, &creds, client_princ,
                                   password, NULL, NULL, 0, NULL, opt);
if (ret)
    goto cleanup;
```

To obtain realm-exposed anonymous credentials, set the anonymous flag on the options structure as above, but specify a normal client principal in order to prove membership in the realm. Authentication will take place as it normally does; if successful, the client principal of the resulting tickets will be `WELLKNOWN/ANONYMOUS@`*realmname*.

4.3 User interaction

Authenticating a user usually requires the entry of secret information, such as a password. A password can be supplied directly to *krb5_get_init_creds_password()* via the *password* parameter, or the application can supply prompter and/or responder callbacks instead. If callbacks are used, the user can also be queried for other secret information such as a PIN, informed of impending password expiration, or prompted to change a password which has expired.

4.3.1 Prompter callback

A prompter callback can be specified via the *prompter* and *data* parameters to *krb5_get_init_creds_password()*. The prompter will be invoked each time the krb5 library has a question to ask or information to present. When the prompter callback is invoked, the *banner* argument (if not null) is intended to be displayed to the user, and the questions to be answered are specified in the *prompts* array. Each prompt contains a text question in the *prompt* field, a *hidden* bit to indicate whether the answer should be hidden from display, and a storage area for the answer in the *reply* field. The callback should fill in each question's `reply->data` with the answer, up to a maximum number of `reply->length` bytes, and then reset `reply->length` to the length of the answer.

A prompter callback can call *krb5_get_prompt_types()* to get an array of type constants corresponding to the prompts, to get programmatic information about the semantic meaning of the questions. *krb5_get_prompt_types()* may return a null pointer if no prompt type information is available.

Text-based applications can use a built-in text prompter implementation by supplying *krb5_prompter_posix()* as the *prompter* parameter and a null pointer as the *data* parameter. For example:

```
ret = krb5_get_init_creds_password(context, &creds, client_princ,
                                   NULL, krb5_prompter_posix, NULL, 0,
                                   NULL, NULL);
```

4.3.2 Responder callback

A responder callback can be specified through the init_creds options using the *krb5_get_init_creds_opt_set_responder()* function. Responder callbacks can present a more sophisticated user interface for authentication secrets. The responder callback is usually invoked only once per authentication, with a list of questions produced by all of the allowed preauthentication mechanisms.

When the responder callback is invoked, the *rctx* argument can be accessed to obtain the list of questions and to answer them. The *krb5_responder_list_questions()* function retrieves an array of question types. For each question type, the *krb5_responder_get_challenge()* function retrieves additional information about the question, if applicable, and the *krb5_responder_set_answer()* function sets the answer.

Responder question types, challenges, and answers are UTF-8 strings. The question type is a well-known string; the meaning of the challenge and answer depend on the question type. If an application does not understand a question type, it cannot interpret the challenge or provide an answer. Failing to answer a question typically results in the prompter callback being used as a fallback.

Password question

The `KRB5_RESPONDER_QUESTION_PASSWORD` (or `"password"`) question type requests the user's password. This question does not have a challenge, and the response is simply the password string.

One-time password question

The KRB5_RESPONDER_QUESTION_OTP (or "otp") question type requests a choice among one-time password tokens and the PIN and value for the chosen token. The challenge and answer are JSON-encoded strings, but an application can use convenience functions to avoid doing any JSON processing itself.

The *krb5_responder_otp_get_challenge()* function decodes the challenge into a krb5_responder_otp_challenge structure. The *krb5_responder_otp_set_answer()* function selects one of the token information elements from the challenge and supplies the value and pin for that token.

PKINIT password or PIN question

The KRB5_RESPONDER_QUESTION_PKINIT (or "pkinit") question type requests PINs for hardware devices and/or passwords for encrypted credentials which are stored on disk, potentially also supplying information about the state of the hardware devices. The challenge and answer are JSON-encoded strings, but an application can use convenience functions to avoid doing any JSON processing itself.

The *krb5_responder_pkinit_get_challenge()* function decodes the challenges into a krb5_responder_pkinit_challenge structure. The *krb5_responder_pkinit_set_answer()* function can be used to supply the PIN or password for a particular client credential, and can be called multiple times.

Example

Here is an example of using a responder callback:

```
static krb5_error_code
my_responder(krb5_context context, void *data,
             krb5_responder_context rctx)
{
    krb5_error_code ret;
    krb5_responder_otp_challenge *chl;

    if (krb5_responder_get_challenge(context, rctx,
                                     KRB5_RESPONDER_QUESTION_PASSWORD)) {
        ret = krb5_responder_set_answer(context, rctx,
                                        KRB5_RESPONDER_QUESTION_PASSWORD,
                                        "open sesame");
        if (ret)
            return ret;
    }
    ret = krb5_responder_otp_get_challenge(context, rctx, &chl);
    if (ret == 0 && chl != NULL) {
        ret = krb5_responder_otp_set_answer(context, rctx, 0, "1234",
                                            NULL);
        krb5_responder_otp_challenge_free(context, rctx, chl);
        if (ret)
            return ret;
    }
    return 0;
}

static krb5_error_code
get_creds(krb5_context context, krb5_principal client_princ)
{
    krb5_error_code ret;
    krb5_get_init_creds_opt *opt = NULL;
```

```
    krb5_creds creds;

    memset(&creds, 0, sizeof(creds));
    ret = krb5_get_init_creds_opt_alloc(context, &opt);
    if (ret)
        goto cleanup;
    ret = krb5_get_init_creds_opt_set_responder(context, opt, my_responder,
                                                NULL);
    if (ret)
        goto cleanup;
    ret = krb5_get_init_creds_password(context, &creds, client_princ,
                                       NULL, NULL, NULL, 0, NULL, opt);

cleanup:
    krb5_get_init_creds_opt_free(context, opt);
    krb5_free_cred_contents(context, &creds);
    return ret;
}
```

4.4 Verifying initial credentials

Use the function *krb5_verify_init_creds()* to verify initial credentials. It takes an options structure (which can be a null pointer). Use *krb5_verify_init_creds_opt_init()* to initialize the caller-allocated options structure, and *krb5_verify_init_creds_opt_set_ap_req_nofail()* to set the "nofail" option. For example:

```
krb5_verify_init_creds_opt vopt;

krb5_verify_init_creds_opt_init(&vopt);
krb5_verify_init_creds_opt_set_ap_req_nofail(&vopt, 1);
ret = krb5_verify_init_creds(context, &creds, NULL, NULL, NULL, &vopt);
```

The confusingly named "nofail" option, when set, means that the verification must actually succeed in order for *krb5_verify_init_creds()* to indicate success. The default state of this option (cleared) means that if there is no key material available to verify the user credentials, the verification will succeed anyway. (The default can be changed by a configuration file setting.)

This accommodates a use case where a large number of unkeyed shared desktop workstations need to allow users to log in using Kerberos. The security risks from this practice are mitigated by the absence of valuable state on the shared workstations—any valuable resources that the users would access reside on networked servers.

CHAPTER

FIVE

PRINCIPAL MANIPULATION AND PARSING

Kerberos principal structure

krb5_principal_data

krb5_principal

Create and free principal

krb5_build_principal()

krb5_build_principal_alloc_va()

krb5_build_principal_ext()

krb5_copy_principal()

krb5_free_principal()

krb5_cc_get_principal()

Comparing

krb5_principal_compare()

krb5_principal_compare_flags()

krb5_principal_compare_any_realm()

krb5_sname_match()

krb5_sname_to_principal()

Parsing:

krb5_parse_name()

krb5_parse_name_flags()

krb5_unparse_name()

krb5_unparse_name_flags()

Utilities:

krb5_is_config_principal()

krb5_kuserok()

krb5_set_password()

krb5_set_password_using_ccache()

krb5_set_principal_realm()

krb5_realm_compare()

CHAPTER

SIX

COMPLETE REFERENCE - API AND DATATYPES

6.1 krb5 API

6.1.1 Frequently used public interfaces

krb5_build_principal - Build a principal name using null-terminated strings.

krb5_error_code **krb5_build_principal** (*krb5_context* *context*, *krb5_principal* * *princ*, unsigned int *rlen*, const char * *realm*, ...)

param **[in] context** - Library context

[out] princ - Principal name

[in] rlen - Realm name length

[in] realm - Realm name

retval

- 0 Success

return

- Kerberos error codes

Call *krb5_free_principal()* to free *princ* when it is no longer needed.

Note: *krb5_build_principal()* and *krb5_build_principal_alloc_va()* perform the same task. *krb5_build_principal()* takes variadic arguments. *krb5_build_principal_alloc_va()* takes a pre-computed *varargs* pointer.

---

krb5_build_principal_alloc_va - Build a principal name, using a precomputed variable argument list.

krb5_error_code **krb5_build_principal_alloc_va** (*krb5_context* *context*, *krb5_principal* * *princ*, unsigned int *rlen*, const char * *realm*, va_list *ap*)

param **[in] context** - Library context

[out] princ - Principal structure

[in] rlen - Realm name length

[in] realm - Realm name

[in] ap - List of char * components, ending with NULL

retval

- 0 Success

return

- Kerberos error codes

Similar to *krb5_build_principal()* , this function builds a principal name, but its name components are specified as a va_list.

Use *krb5_free_principal()* to deallocate *princ* when it is no longer needed.

krb5_build_principal_ext - Build a principal name using length-counted strings.

krb5_error_code **krb5_build_principal_ext** (*krb5_context context*, *krb5_principal* * *princ*, unsigned int *rlen*, const char * *realm*, ...)

param [in] context - Library context

[out] princ - Principal name

[in] rlen - Realm name length

[in] realm - Realm name

retval

- 0 Success

return

- Kerberos error codes

This function creates a principal from a length-counted string and a variable-length list of length-counted components. The list of components ends with the first 0 length argument (so it is not possible to specify an empty component with this function). Call *krb5_free_principal()* to free allocated memory for principal when it is no longer needed.

krb5_cc_close - Close a credential cache handle.

krb5_error_code **krb5_cc_close** (*krb5_context context*, *krb5_ccache cache*)

param [in] context - Library context

[in] cache - Credential cache handle

retval

- 0 Success

return

- Kerberos error codes

This function closes a credential cache handle *cache* without affecting the contents of the cache.

krb5_cc_default - Resolve the default credential cache name.

krb5_error_code **krb5_cc_default** (*krb5_context context*, *krb5_ccache* * *ccache*)

param [in] context - Library context

[out] ccache - Pointer to credential cache name

retval

- 0 Success
- KV5M_CONTEXT Bad magic number for _krb5_context structure
- KRB5_FCC_INTERNAL The name of the default credential cache cannot be obtained

return

- Kerberos error codes

Create a handle to the default credential cache as given by *krb5_cc_default_name()* .

krb5_cc_default_name - Return the name of the default credential cache.

const char * **krb5_cc_default_name** (*krb5_context context*)

param [in] context - Library context

return

- Name of default credential cache for the current user.

Return a pointer to the default credential cache name for *context* , as determined by a prior call to *krb5_cc_set_default_name()* , by the KRB5CCNAME environment variable, by the default_ccache_name profile variable, or by the operating system or build-time default value. The returned value must not be modified or freed by the caller. The returned value becomes invalid when *context* is destroyed *krb5_free_context()* or if a subsequent call to *krb5_cc_set_default_name()* is made on *context* .

The default credential cache name is cached in *context* between calls to this function, so if the value of KRB5CCNAME changes in the process environment after the first call to this function on, that change will not be reflected in later calls with the same context. The caller can invoke *krb5_cc_set_default_name()* with a NULL value of *name* to clear the cached value and force the default name to be recomputed.

krb5_cc_destroy - Destroy a credential cache.

krb5_error_code **krb5_cc_destroy** (*krb5_context context*, *krb5_ccache cache*)

param [in] context - Library context

[in] cache - Credential cache handle

retval

- 0 Success

return

- Permission errors

This function destroys any existing contents of *cache* and closes the handle to it.

krb5_cc_dup - Duplicate ccache handle.

krb5_error_code **krb5_cc_dup** (*krb5_context context*, *krb5_ccache in*, *krb5_ccache * out*)

param [in] context - Library context

[in] in - Credential cache handle to be duplicated

[out] out - Credential cache handle

Create a new handle referring to the same cache as *in* . The new handle and *in* can be closed independently.

krb5_cc_get_name - Retrieve the name, but not type of a credential cache.

const char * **krb5_cc_get_name** (*krb5_context context*, *krb5_ccache cache*)

param **[in] context** - Library context

[in] cache - Credential cache handle

return

- On success - the name of the credential cache.

Warning: Returns the name of the credential cache. The result is an alias into *cache* and should not be freed or modified by the caller. This name does not include the cache type, so should not be used as input to *krb5_cc_resolve()* .

krb5_cc_get_principal - Get the default principal of a credential cache.

krb5_error_code **krb5_cc_get_principal** (*krb5_context context*, *krb5_ccache cache*, *krb5_principal * principal*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[out] principal - Primary principal

retval

- 0 Success

return

- Kerberos error codes

Returns the default client principal of a credential cache as set by *krb5_cc_initialize()* .

Use *krb5_free_principal()* to free *principal* when it is no longer needed.

krb5_cc_get_type - Retrieve the type of a credential cache.

const char * **krb5_cc_get_type** (*krb5_context context*, *krb5_ccache cache*)

param **[in] context** - Library context

[in] cache - Credential cache handle

return

- The type of a credential cache as an alias that must not be modified or freed by the caller.

krb5_cc_initialize - Initialize a credential cache.

krb5_error_code **krb5_cc_initialize** (*krb5_context context*, *krb5_ccache cache*, *krb5_principal principal*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[in] principal - Default principal name

retval

- 0 Success

return

- System errors; Permission errors; Kerberos error codes

Destroy any existing contents of *cache* and initialize it for the default principal *principal* .

krb5_cc_new_unique - Create a new credential cache of the specified type with a unique name.

krb5_error_code **krb5_cc_new_unique**(*krb5_context context*, const char * *type*, const char * *hint*, *krb5_ccache* * *id*)

param **[in] context** - Library context

[in] type - Credential cache type name

[in] hint - Unused

[out] id - Credential cache handle

retval

- 0 Success

return

- Kerberos error codes

krb5_cc_resolve - Resolve a credential cache name.

krb5_error_code **krb5_cc_resolve**(*krb5_context context*, const char * *name*, *krb5_ccache* * *cache*)

param **[in] context** - Library context

[in] name - Credential cache name to be resolved

[out] cache - Credential cache handle

retval

- 0 Success

return

- Kerberos error codes

Fills in *cache* with a *cache* handle that corresponds to the name in *name* . *name* should be of the form **type:residual** , and *type* must be a type known to the library. If the *name* does not contain a colon, interpret it as a file name.

krb5_change_password - Change a password for an existing Kerberos account.

krb5_error_code **krb5_change_password**(*krb5_context context*, *krb5_creds* * *creds*, const char * *newpw*, int * *result_code*, *krb5_data* * *result_code_string*, *krb5_data* * *result_string*)

param [in] context - Library context

[in] creds - Credentials for kadmin/changepw service

[in] newpw - New password

[out] result_code - Numeric error code from server

[out] result_code_string - String equivalent to *result_code*

[out] result_string - Change password response from the KDC

retval

- 0 Success; otherwise - Kerberos error codes

Change the password for the existing principal identified by *creds* .

The possible values of the output *result_code* are:

- *KRB5_KPASSWD_SUCCESS* (0) - success
- *KRB5_KPASSWD_MALFORMED* (1) - Malformed request error
- *KRB5_KPASSWD_HARDERROR* (2) - Server error
- *KRB5_KPASSWD_AUTHERROR* (3) - Authentication error
- *KRB5_KPASSWD_SOFTERROR* (4) - Password change rejected

krb5_chpw_message - Get a result message for changing or setting a password.

krb5_error_code **krb5_chpw_message**(*krb5_context context*, const *krb5_data* * *server_string*, char ** *message_out*)

param [in] context - Library context

[in] server_string - Data returned from the remote system

[out] message_out - A message displayable to the user

retval

- 0 Success

return

- Kerberos error codes

This function processes the *server_string* returned in the *result_string* parameter of *krb5_change_password()* , *krb5_set_password()* , and related functions, and returns a displayable string. If *server_string* contains Active Directory structured policy information, it will be converted into human-readable text.

Use *krb5_free_string()* to free *message_out* when it is no longer needed.

Note: New in 1.11

krb5_expand_hostname - Canonicalize a hostname, possibly using name service.

krb5_error_code **krb5_expand_hostname**(*krb5_context context*, const char * *host*, char ** *canonhost_out*)

param **[in] context** - Library context

[in] host - Input hostname

[out] canonhost_out - Canonicalized hostname

This function canonicalizes orig_hostname, possibly using name service lookups if configuration permits. Use *krb5_free_string()* to free *canonhost_out* when it is no longer needed.

Note: New in 1.15

---

krb5_free_context - Free a krb5 library context.

void **krb5_free_context** (*krb5_context context*)

param **[in] context** - Library context

This function frees a *context* that was created by *krb5_init_context()* or *krb5_init_secure_context()* .

krb5_free_error_message - Free an error message generated by krb5_get_error_message() .

void **krb5_free_error_message** (*krb5_context ctx*, const char * *msg*)

param **[in] ctx** - Library context

[in] msg - Pointer to error message

krb5_free_principal - Free the storage assigned to a principal.

void **krb5_free_principal** (*krb5_context context*, *krb5_principal val*)

param **[in] context** - Library context

[in] val - Principal to be freed

krb5_fwd_tgt_creds - Get a forwarded TGT and format a KRB-CRED message.

krb5_error_code **krb5_fwd_tgt_creds** (*krb5_context context*, *krb5_auth_context auth_context*, const char * *rhost*, *krb5_principal client*, *krb5_principal server*, *krb5_ccache cc*, int *forwardable*, *krb5_data* * *outbuf*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] rhost - Remote host

[in] client - Client principal of TGT

[in] server - Principal of server to receive TGT

[in] cc - Credential cache handle (NULL to use default)

[in] forwardable - Whether TGT should be forwardable

[out] outbuf - KRB-CRED message

retval

- 0 Success
- ENOMEM Insufficient memory
- KRB5_PRINC_NOMATCH Requested principal and ticket do not match
- KRB5_NO_TKT_SUPPLIED Request did not supply a ticket
- KRB5_CC_BADNAME Credential cache name or principal name malformed

return

- Kerberos error codes

Get a TGT for use at the remote host *rhost* and format it into a KRB-CRED message. If *rhost* is NULL and *server* is of type `KRB5_NT_SRV_HST` , the second component of *server* will be used.

krb5_get_default_realm - Retrieve the default realm.

krb5_error_code **krb5_get_default_realm**(*krb5_context context*, char ** *lrealm*)

param **[in] context** - Library context

[out] lrealm - Default realm name

retval

- 0 Success

return

- Kerberos error codes

Retrieves the default realm to be used if no user-specified realm is available.

Use `krb5_free_default_realm()` to free *lrealm* when it is no longer needed.

krb5_get_error_message - Get the (possibly extended) error message for a code.

const char * **krb5_get_error_message**(*krb5_context ctx*, *krb5_error_code code*)

param **[in] ctx** - Library context

[in] code - Error code

The behavior of `krb5_get_error_message()` is only defined the first time it is called after a failed call to a krb5 function using the same context, and only when the error code passed in is the same as that returned by the krb5 function.

This function never returns NULL, so its result may be used unconditionally as a C string.

The string returned by this function must be freed using `krb5_free_error_message()`

Note: Future versions may return the same string for the second and following calls.

krb5_get_host_realm - Get the Kerberos realm names for a host.

krb5_error_code **krb5_get_host_realm**(*krb5_context context*, const char * *host*, char *** *realmsp*)

param **[in] context** - Library context

[in] host - Host name (or NULL)

[out] realmsp - Null-terminated list of realm names

retval

- 0 Success
- ENOMEM Insufficient memory

return

- Kerberos error codes

Fill in *realmsp* with a pointer to a null-terminated list of realm names. If there are no known realms for the host, a list containing the referral (empty) realm is returned.

If *host* is NULL, the local host's realms are determined.

Use `krb5_free_host_realm()` to release *realmsp* when it is no longer needed.

krb5_get_credentials - Get an additional ticket.

krb5_error_code **krb5_get_credentials** (*krb5_context context*, *krb5_flags options*, *krb5_ccache ccache*, *krb5_creds * in_creds*, *krb5_creds ** *out_creds*)

param **[in] context** - Library context

[in] options - Options

[in] ccache - Credential cache handle

[in] in_creds - Input credentials

[out] out_creds - Output updated credentials

retval

- 0 Success

return

- Kerberos error codes

Use *ccache* or a TGS exchange to get a service ticket matching *in_creds* .

Valid values for *options* are:

- `KRB5_GC_CACHED` Search only credential cache for the ticket
- `KRB5_GC_USER_USER` Return a user to user authentication ticket

in_creds must be non-null. *in_creds->client* and *in_creds->server* must be filled in to specify the client and the server respectively. If any authorization data needs to be requested for the service ticket (such as restrictions on how the ticket can be used), specify it in *in_creds->authdata* ; otherwise set *in_creds->authdata* to NULL. The session key type is specified in *in_creds->keyblock.enctype* , if it is nonzero.

The expiration date is specified in *in_creds->times.endtime* . The KDC may return tickets with an earlier expiration date. If *in_creds->times.endtime* is set to 0, the latest possible expiration date will be requested.

Any returned ticket and intermediate ticket-granting tickets are stored in *ccache* .

Use `krb5_free_creds()` to free *out_creds* when it is no longer needed.

krb5_get_fallback_host_realm

krb5_error_code **krb5_get_fallback_host_realm**(*krb5_context context*, *krb5_data * hdata*, char *** *realmsp*)

param **[in] context** - Library context

[in] hdata - Host name (or NULL)

[out] realmsp - Null-terminated list of realm names

Fill in *realmsp* with a pointer to a null-terminated list of realm names obtained through heuristics or insecure resolution methods which have lower priority than KDC referrals.

If *host* is NULL, the local host's realms are determined.

Use *krb5_free_host_realm()* to release *realmsp* when it is no longer needed.

krb5_get_init_creds_keytab - Get initial credentials using a key table.

krb5_error_code **krb5_get_init_creds_keytab**(*krb5_context context*, *krb5_creds * creds*, *krb5_principal client*, *krb5_keytab arg_keytab*, *krb5_deltat start_time*, const char * *in_tkt_service*, *krb5_get_init_creds_opt * k5_gic_options*)

param **[in] context** - Library context

[out] creds - New credentials

[in] client - Client principal

[in] arg_keytab - Key table handle

[in] start_time - Time when ticket becomes valid (0 for now)

[in] in_tkt_service - Service name of initial credentials (or NULL)

[in] k5_gic_options - Initial credential options

retval

- 0 Success

return

- Kerberos error codes

This function requests KDC for an initial credentials for *client* using a client key stored in *arg_keytab* . If *in_tkt_service* is specified, it is parsed as a principal name (with the realm ignored) and used as the service principal for the request; otherwise the ticket-granting service is used.

krb5_get_init_creds_opt_alloc - Allocate a new initial credential options structure.

krb5_error_code **krb5_get_init_creds_opt_alloc**(*krb5_context context*, *krb5_get_init_creds_opt ** *opt*)

param **[in] context** - Library context

[out] opt - New options structure

retval

- 0 - Success; Kerberos errors otherwise.

This function is the preferred way to create an options structure for getting initial credentials, and is required to make use of certain options. Use *krb5_get_init_creds_opt_free()* to free *opt* when it is no longer needed.

krb5_get_init_creds_opt_free - Free initial credential options.

void **krb5_get_init_creds_opt_free** (*krb5_context context*, *krb5_get_init_creds_opt * opt*)

param **[in] context** - Library context

[in] opt - Options structure to free

See also:

krb5_get_init_creds_opt_alloc()

krb5_get_init_creds_opt_get_fast_flags - Retrieve FAST flags from initial credential options.

krb5_error_code **krb5_get_init_creds_opt_get_fast_flags** (*krb5_context context*, *krb5_get_init_creds_opt * opt*, *krb5_flags * out_flags*)

param **[in] context** - Library context

[in] opt - Options

[out] out_flags - FAST flags

retval

- 0 - Success; Kerberos errors otherwise.

krb5_get_init_creds_opt_set_address_list - Set address restrictions in initial credential options.

void **krb5_get_init_creds_opt_set_address_list** (*krb5_get_init_creds_opt * opt*, *krb5_address ** addresses*)

param **[in] opt** - Options structure

[in] addresses - Null-terminated array of addresses

krb5_get_init_creds_opt_set_anonymous - Set or unset the anonymous flag in initial credential options.

void **krb5_get_init_creds_opt_set_anonymous** (*krb5_get_init_creds_opt * opt*, int *anonymous*)

param **[in] opt** - Options structure

[in] anonymous - Whether to make an anonymous request

This function may be used to request anonymous credentials from the KDC by setting *anonymous* to non-zero. Note that anonymous credentials are only a request; clients must verify that credentials are anonymous if that is a requirement.

krb5_get_init_creds_opt_set_canonicalize - Set or unset the canonicalize flag in initial credential options.

void **krb5_get_init_creds_opt_set_canonicalize**(*krb5_get_init_creds_opt* * *opt*, int *canonicalize*)

param **[in] opt** - Options structure

[in] canonicalize - Whether to canonicalize client principal

krb5_get_init_creds_opt_set_change_password_prompt - Set or unset change-password-prompt flag in initial credential options.

void **krb5_get_init_creds_opt_set_change_password_prompt**(*krb5_get_init_creds_opt* * *opt*, int *prompt*)

param **[in] opt** - Options structure

[in] prompt - Whether to prompt to change password

This flag is on by default. It controls whether `krb5_get_init_creds_password()` will react to an expired-password error by prompting for a new password and attempting to change the old one.

krb5_get_init_creds_opt_set_etype_list - Set allowable encryption types in initial credential options.

void **krb5_get_init_creds_opt_set_etype_list**(*krb5_get_init_creds_opt* * *opt*, *krb5_enctype* * *etype_list*, int *etype_list_length*)

param **[in] opt** - Options structure

[in] etype_list - Array of encryption types

[in] etype_list_length - Length of *etype_list*

krb5_get_init_creds_opt_set_expire_callback - Set an expiration callback in initial credential options.

krb5_error_code **krb5_get_init_creds_opt_set_expire_callback**(*krb5_context* *context*, *krb5_get_init_creds_opt* * *opt*, *krb5_expire_callback_func* *cb*, void * *data*)

param **[in] context** - Library context

[in] opt - Options structure

[in] cb - Callback function

[in] data - Callback argument

Set a callback to receive password and account expiration times.

This option only applies to `krb5_get_init_creds_password()` . *cb* will be invoked if and only if credentials are successfully acquired. The callback will receive the *context* from the `krb5_get_init_creds_password()` call and the *data* argument supplied with this API. The remaining arguments should be interpreted as follows:

If *is_last_req* is true, then the KDC reply contained last-req entries which unambiguously indicated the password expiration, account expiration, or both. (If either value was not present, the corresponding argument will be 0.) Furthermore, a non-zero *password_expiration* should be taken as a suggestion from the KDC that a warning be displayed.

If *is_last_req* is false, then *account_expiration* will be 0 and *password_expiration* will contain the expiration time of either the password or account, or 0 if no expiration time was indicated in the KDC reply. The callback should independently decide whether to display a password expiration warning.

Note that *cb* may be invoked even if credentials are being acquired for the kadmin/changepw service in order to change the password. It is the caller's responsibility to avoid displaying a password expiry warning in this case.

Warning: Setting an expire callback with this API will cause *krb5_get_init_creds_password()* not to send password expiry warnings to the prompter, as it ordinarily may.

Note: New in 1.9

krb5_get_init_creds_opt_set_fast_ccache - Set FAST armor cache in initial credential options.

krb5_error_code **krb5_get_init_creds_opt_set_fast_ccache**(*krb5_context context*, *krb5_get_init_creds_opt * opt*, *krb5_ccache ccache*)

param [in] context - Library context

[in] opt - Options

[in] ccache - Credential cache handle

This function is similar to *krb5_get_init_creds_opt_set_fast_ccache_name()* , but uses a credential cache handle instead of a name.

Note: New in 1.9

krb5_get_init_creds_opt_set_fast_ccache_name - Set location of FAST armor ccache in initial credential options.

krb5_error_code **krb5_get_init_creds_opt_set_fast_ccache_name**(*krb5_context context*, *krb5_get_init_creds_opt * opt*, const char *fast_ccache_name*)

param [in] context - Library context

[in] opt - Options

[in] fast_ccache_name - Credential cache name

Sets the location of a credential cache containing an armor ticket to protect an initial credential exchange using the FAST protocol extension.

In version 1.7, setting an armor ccache requires that FAST be used for the exchange. In version 1.8 or later, setting the armor ccache causes FAST to be used if the KDC supports it; *krb5_get_init_creds_opt_set_fast_flags()* must be used to require that FAST be used.

krb5_get_init_creds_opt_set_fast_flags - Set FAST flags in initial credential options.

krb5_error_code **krb5_get_init_creds_opt_set_fast_flags** (*krb5_context context*, *krb5_get_init_creds_opt * opt*, *krb5_flags flags*)

param **[in] context** - Library context

[in] opt - Options

[in] flags - FAST flags

retval

- 0 - Success; Kerberos errors otherwise.

The following flag values are valid:

- *KRB5_FAST_REQUIRED* - Require FAST to be used

krb5_get_init_creds_opt_set_forwardable - Set or unset the forwardable flag in initial credential options.

void **krb5_get_init_creds_opt_set_forwardable** (*krb5_get_init_creds_opt * opt*, int *forwardable*)

param **[in] opt** - Options structure

[in] forwardable - Whether credentials should be forwardable

krb5_get_init_creds_opt_set_in_ccache - Set an input credential cache in initial credential options.

krb5_error_code **krb5_get_init_creds_opt_set_in_ccache** (*krb5_context context*, *krb5_get_init_creds_opt * opt*, *krb5_ccache ccache*)

param **[in] context** - Library context

[in] opt - Options

[in] ccache - Credential cache handle

If an input credential cache is set, then the krb5_get_init_creds family of APIs will read settings from it. Setting an input ccache is desirable when the application wishes to perform authentication in the same way (using the same preauthentication mechanisms, and making the same non-security- sensitive choices) as the previous authentication attempt, which stored information in the passed-in ccache.

Note: New in 1.11

---

krb5_get_init_creds_opt_set_out_ccache - Set an output credential cache in initial credential options.

krb5_error_code **krb5_get_init_creds_opt_set_out_ccache** (*krb5_context context*, *krb5_get_init_creds_opt * opt*, *krb5_ccache ccache*)

param **[in] context** - Library context

[in] opt - Options

[in] ccache - Credential cache handle

If an output credential cache is set, then the krb5_get_init_creds family of APIs will write credentials to it. Setting an output ccache is desirable both because it simplifies calling code and because it permits the krb5_get_init_creds APIs to write out configuration information about the realm to the ccache.

krb5_get_init_creds_opt_set_pa - Supply options for preauthentication in initial credential options.

krb5_error_code **krb5_get_init_creds_opt_set_pa** (*krb5_context context*, *krb5_get_init_creds_opt * opt*, const char * *attr*, const char * *value*)

param **[in] context** - Library context

[in] opt - Options structure

[in] attr - Preauthentication option name

[in] value - Preauthentication option value

This function allows the caller to supply options for preauthentication. The values of *attr* and *value* are supplied to each preauthentication module available within *context* .

krb5_get_init_creds_opt_set_pac_request - Ask the KDC to include or not include a PAC in the ticket.

krb5_error_code **krb5_get_init_creds_opt_set_pac_request** (*krb5_context context*, *krb5_get_init_creds_opt * opt*, *krb5_boolean req_pac*)

param **[in] context** - Library context

[in] opt - Options structure

[in] req_pac - Whether to request a PAC or not

If this option is set, the AS request will include a PAC-REQUEST pa-data item explicitly asking the KDC to either include or not include a privilege attribute certificate in the ticket authorization data. By default, no request is made; typically the KDC will default to including a PAC if it supports them.

Note: New in 1.15

---

krb5_get_init_creds_opt_set_preauth_list - Set preauthentication types in initial credential options.

void **krb5_get_init_creds_opt_set_preauth_list** (*krb5_get_init_creds_opt * opt*, *krb5_preauthtype * preauth_list*, int *preauth_list_length*)

param **[in] opt** - Options structure

[in] preauth_list - Array of preauthentication types

[in] preauth_list_length - Length of *preauth_list*

This function can be used to perform optimistic preauthentication when getting initial credentials, in combination with *krb5_get_init_creds_opt_set_salt()* and *krb5_get_init_creds_opt_set_pa()* .

krb5_get_init_creds_opt_set_proxiable - Set or unset the proxiable flag in initial credential options.

void **krb5_get_init_creds_opt_set_proxiable**(*krb5_get_init_creds_opt* * *opt*, int *proxiable*)

param **[in] opt** - Options structure

[in] proxiable - Whether credentials should be proxiable

krb5_get_init_creds_opt_set_renew_life - Set the ticket renewal lifetime in initial credential options.

void **krb5_get_init_creds_opt_set_renew_life**(*krb5_get_init_creds_opt* * *opt*, *krb5_deltat* *renew_life*)

param **[in] opt** - Pointer to *options* field

[in] renew_life - Ticket renewal lifetime

krb5_get_init_creds_opt_set_responder - Set the responder function in initial credential options.

krb5_error_code **krb5_get_init_creds_opt_set_responder**(*krb5_context* *context*, *krb5_get_init_creds_opt* * *opt*, *krb5_responder_fn* *responder*, void * *data*)

param **[in] context** - Library context

[in] opt - Options structure

[in] responder - Responder function

[in] data - Responder data argument

Note: New in 1.11

---

krb5_get_init_creds_opt_set_salt - Set salt for optimistic preauthentication in initial credential options.

void **krb5_get_init_creds_opt_set_salt**(*krb5_get_init_creds_opt* * *opt*, *krb5_data* * *salt*)

param **[in] opt** - Options structure

[in] salt - Salt data

When getting initial credentials with a password, a salt string it used to convert the password to a key. Normally this salt is obtained from the first KDC reply, but when performing optimistic preauthentication, the client may need to supply the salt string with this function.

krb5_get_init_creds_opt_set_tkt_life - Set the ticket lifetime in initial credential options.

void **krb5_get_init_creds_opt_set_tkt_life**(*krb5_get_init_creds_opt* * *opt*, *krb5_deltat* *tkt_life*)

param **[in] opt** - Options structure

[in] tkt_life - Ticket lifetime

krb5_get_init_creds_password - Get initial credentials using a password.

krb5_error_code **krb5_get_init_creds_password**(*krb5_context context*, *krb5_creds* * *creds*, *krb5_principal client*, const char * *password*, *krb5_prompter_fct prompter*, void * *data*, *krb5_deltat start_time*, const char * *in_tkt_service*, *krb5_get_init_creds_opt* * *k5_gic_options*)

param [in] context - Library context

[out] creds - New credentials

[in] client - Client principal

[in] password - Password (or NULL)

[in] prompter - Prompter function

[in] data - Prompter callback data

[in] start_time - Time when ticket becomes valid (0 for now)

[in] in_tkt_service - Service name of initial credentials (or NULL)

[in] k5_gic_options - Initial credential options

retval

- 0 Success
- EINVAL Invalid argument
- KRB5_KDC_UNREACH Cannot contact any KDC for requested realm
- KRB5_PREAUTH_FAILED Generic Pre-athentication failure
- KRB5_LIBOS_PWDINTR Password read interrupted
- KRB5_REALM_CANT_RESOLVE Cannot resolve network address for KDC in requested realm
- KRB5KDC_ERR_KEY_EXP Password has expired
- KRB5_LIBOS_BADPWDMATCH Password mismatch
- KRB5_CHPW_PWDNULL New password cannot be zero length
- KRB5_CHPW_FAIL Password change failed

return

- Kerberos error codes

This function requests KDC for an initial credentials for *client* using *password* . If *password* is NULL, a password will be prompted for using *prompter* if necessary. If *in_tkt_service* is specified, it is parsed as a principal name (with the realm ignored) and used as the service principal for the request; otherwise the ticket-granting service is used.

krb5_get_profile - Retrieve configuration profile from the context.

krb5_error_code **krb5_get_profile**(*krb5_context context*, struct _profile_t ** *profile*)

param [in] context - Library context

[out] profile - Pointer to data read from a configuration file

retval

- 0 Success

return

- Kerberos error codes

This function creates a new *profile* object that reflects profile in the supplied *context* .

The *profile* object may be freed with profile_release() function. See profile.h and profile API for more details.

krb5_get_prompt_types - Get prompt types array from a context.

krb5_prompt_type * **krb5_get_prompt_types** (*krb5_context context*)

param [in] context - Library context

return

- Pointer to an array of prompt types corresponding to the prompter's prompts arguments. Each type has one of the following values: KRB5_PROMPT_TYPE_PASSWORD KRB5_PROMPT_TYPE_NEW_PASSWORD KRB5_PROMPT_TYPE_NEW_PASSWORD_AGAIN KRB5_PROMPT_TYPE_PREAUTH

krb5_get_renewed_creds - Get renewed credential from KDC using an existing credential.

krb5_error_code **krb5_get_renewed_creds** (*krb5_context context*, *krb5_creds * creds*, *krb5_principal client*, *krb5_ccache ccache*, const char * *in_tkt_service*)

param [in] context - Library context

[out] creds - Renewed credentials

[in] client - Client principal name

[in] ccache - Credential cache

[in] in_tkt_service - Server principal string (or NULL)

retval

- 0 Success

return

- Kerberos error codes

This function gets a renewed credential using an existing one from *ccache* . If *in_tkt_service* is specified, it is parsed (with the realm part ignored) and used as the server principal of the credential; otherwise, the ticket-granting service is used.

If successful, the renewed credential is placed in *creds* .

krb5_get_validated_creds - Get validated credentials from the KDC.

krb5_error_code **krb5_get_validated_creds** (*krb5_context context*, *krb5_creds * creds*, *krb5_principal client*, *krb5_ccache ccache*, const char * *in_tkt_service*)

param **[in] context** - Library context

[out] creds - Validated credentials

[in] client - Client principal name

[in] ccache - Credential cache

[in] in_tkt_service - Server principal string (or NULL)

retval

- 0 Success
- KRB5_NO_2ND_TKT Request missing second ticket
- KRB5_NO_TKT_SUPPLIED Request did not supply a ticket
- KRB5_PRINC_NOMATCH Requested principal and ticket do not match
- KRB5_KDCREP_MODIFIED KDC reply did not match expectations
- KRB5_KDCREP_SKEW Clock skew too great in KDC reply

return

- Kerberos error codes

This function gets a validated credential using a postdated credential from *ccache* . If *in_tkt_service* is specified, it is parsed (with the realm part ignored) and used as the server principal of the credential; otherwise, the ticket-granting service is used.

If successful, the validated credential is placed in *creds* .

krb5_init_context - Create a krb5 library context.

krb5_error_code **krb5_init_context** (*krb5_context* * *context*)

param **[out] context** - Library context

retval

- 0 Success

return

- Kerberos error codes

The *context* must be released by calling `krb5_free_context()` when it is no longer needed.

Warning: Any program or module that needs the Kerberos code to not trust the environment must use `krb5_init_secure_context()` , or clean out the environment.

krb5_init_secure_context - Create a krb5 library context using only configuration files.

krb5_error_code **krb5_init_secure_context** (*krb5_context* * *context*)

param **[out] context** - Library context

retval

- 0 Success

return

- Kerberos error codes

Create a context structure, using only system configuration files. All information passed through the environment variables is ignored.

The *context* must be released by calling `krb5_free_context()` when it is no longer needed.

krb5_is_config_principal - Test whether a principal is a configuration principal.

krb5_boolean **krb5_is_config_principal** (*krb5_context context*, *krb5_const_principal principal*)

param **[in] context** - Library context

[in] principal - Principal to check

return

- TRUE if the principal is a configuration principal (generated part of krb5_cc_set_config()); FALSE otherwise.

krb5_is_thread_safe - Test whether the Kerberos library was built with multithread support.

krb5_boolean **krb5_is_thread_safe** (void *None*)

param **None**

retval

- TRUE if the library is threadsafe; FALSE otherwise

krb5_kt_close - Close a key table handle.

krb5_error_code **krb5_kt_close** (*krb5_context context*, *krb5_keytab keytab*)

param **[in] context** - Library context

[in] keytab - Key table handle

retval

- 0 None

krb5_kt_client_default - Resolve the default client key table.

krb5_error_code **krb5_kt_client_default** (*krb5_context context*, *krb5_keytab * keytab_out*)

param **[in] context** - Library context

[out] keytab_out - Key table handle

retval

- 0 Success

return

- Kerberos error codes

Fill *keytab_out* with a handle to the default client key table.

Note: New in 1.11

krb5_kt_default - Resolve the default key table.

krb5_error_code **krb5_kt_default** (*krb5_context context*, *krb5_keytab * id*)

param [in] context - Library context

[out] id - Key table handle

retval

- 0 Success

return

- Kerberos error codes

Set *id* to a handle to the default key table. The key table is not opened.

krb5_kt_default_name - Get the default key table name.

krb5_error_code **krb5_kt_default_name** (*krb5_context context*, char * *name*, int *name_size*)

param [in] context - Library context

[out] name - Default key table name

[in] name_size - Space available in *name*

retval

- 0 Success
- KRB5_CONFIG_NOTENUFSPACE Buffer is too short

return

- Kerberos error codes

Fill *name* with the name of the default key table for *context* .

krb5_kt_dup - Duplicate keytab handle.

krb5_error_code **krb5_kt_dup** (*krb5_context context*, *krb5_keytab in*, *krb5_keytab * out*)

param [in] context - Library context

[in] in - Key table handle to be duplicated

[out] out - Key table handle

Create a new handle referring to the same key table as *in* . The new handle and *in* can be closed independently.

Note: New in 1.12

krb5_kt_get_name - Get a key table name.

krb5_error_code **krb5_kt_get_name** (*krb5_context context*, *krb5_keytab keytab*, char * *name*, unsigned int *namelen*)

param **[in] context** - Library context

[in] keytab - Key table handle

[out] name - Key table name

[in] namelen - Maximum length to fill in name

retval

- 0 Success
- KRB5_KT_NAME_TOOLONG Key table name does not fit in namelen bytes

return

- Kerberos error codes

Fill *name* with the name of *keytab* including the type and delimiter.

krb5_kt_get_type - Return the type of a key table.

const char * **krb5_kt_get_type** (*krb5_context context*, *krb5_keytab keytab*)

param **[in] context** - Library context

[in] keytab - Key table handle

return

- The type of a key table as an alias that must not be modified or freed by the caller.

krb5_kt_resolve - Get a handle for a key table.

krb5_error_code **krb5_kt_resolve** (*krb5_context context*, const char * *name*, *krb5_keytab* * *ktid*)

param **[in] context** - Library context

[in] name - Name of the key table

[out] ktid - Key table handle

retval

- 0 Success

return

- Kerberos error codes

Resolve the key table name *name* and set *ktid* to a handle identifying the key table. Use `krb5_kt_close()` to free *ktid* when it is no longer needed.

name must be of the form **type:residual** , where *type* must be a type known to the library and *residual* portion should be specific to the particular keytab type. If no *type* is given, the default is **FILE** .

If *name* is of type **FILE** , the keytab file is not opened by this call.

krb5_kuserok - Determine if a principal is authorized to log in as a local user.

krb5_boolean **krb5_kuserok** (*krb5_context context*, *krb5_principal principal*, const char * *luser*)

param **[in] context** - Library context

[in] principal - Principal name

[in] luser - Local username

retval

- TRUE Principal is authorized to log in as user; FALSE otherwise.

Determine whether *principal* is authorized to log in as a local user *luser* .

krb5_parse_name - Convert a string principal name to a krb5_principal structure.

krb5_error_code **krb5_parse_name** (*krb5_context context*, const char * *name*, *krb5_principal* * *principal_out*)

param **[in] context** - Library context

[in] name - String representation of a principal name

[out] principal_out - New principal

retval

- 0 Success

return

- Kerberos error codes

Convert a string representation of a principal name to a krb5_principal structure.

A string representation of a Kerberos name consists of one or more principal name components, separated by slashes, optionally followed by the @ character and a realm name. If the realm name is not specified, the local realm is used.

To use the slash and @ symbols as part of a component (quoted) instead of using them as a component separator or as a realm prefix), put a backslash () character in front of the symbol. Similarly, newline, tab, backspace, and NULL characters can be included in a component by using **n** , **t** , **b** or **0** , respectively.

Use *krb5_free_principal()* to free *principal_out* when it is no longer needed.

Note: The realm in a Kerberos *name* cannot contain slash, colon, or NULL characters.

---

krb5_parse_name_flags - Convert a string principal name to a krb5_principal with flags.

krb5_error_code **krb5_parse_name_flags** (*krb5_context context*, const char * *name*, int *flags*, *krb5_principal* * *principal_out*)

param **[in] context** - Library context

[in] name - String representation of a principal name

[in] flags - Flag

[out] principal_out - New principal

retval

- 0 Success

return

- Kerberos error codes

Similar to *krb5_parse_name()* , this function converts a single-string representation of a principal name to a krb5_principal structure.

The following flags are valid:

- *KRB5_PRINCIPAL_PARSE_NO_REALM* - no realm must be present in *name*
- *KRB5_PRINCIPAL_PARSE_REQUIRE_REALM* - realm must be present in *name*
- *KRB5_PRINCIPAL_PARSE_ENTERPRISE* - create single-component enterprise principal
- *KRB5_PRINCIPAL_PARSE_IGNORE_REALM* - ignore realm if present in *name*

If **KRB5_PRINCIPAL_PARSE_NO_REALM** or **KRB5_PRINCIPAL_PARSE_IGNORE_REALM** is specified in *flags* , the realm of the new principal will be empty. Otherwise, the default realm for *context* will be used if *name* does not specify a realm.

Use *krb5_free_principal()* to free *principal_out* when it is no longer needed.

krb5_principal_compare - Compare two principals.

krb5_boolean **krb5_principal_compare**(*krb5_context context*, *krb5_const_principal princ1*, *krb5_const_principal princ2*)

param [in] context - Library context

[in] princ1 - First principal

[in] princ2 - Second principal

retval

- TRUE if the principals are the same; FALSE otherwise

krb5_principal_compare_any_realm - Compare two principals ignoring realm components.

krb5_boolean **krb5_principal_compare_any_realm**(*krb5_context context*, *krb5_const_principal princ1*, *krb5_const_principal princ2*)

param [in] context - Library context

[in] princ1 - First principal

[in] princ2 - Second principal

retval

- TRUE if the principals are the same; FALSE otherwise

Similar to *krb5_principal_compare()* , but do not compare the realm components of the principals.

krb5_principal_compare_flags - Compare two principals with additional flags.

krb5_boolean **krb5_principal_compare_flags**(*krb5_context context*, *krb5_const_principal princ1*, *krb5_const_principal princ2*, int *flags*)

param **[in] context** - Library context

[in] princ1 - First principal

[in] princ2 - Second principal

[in] flags - Flags

retval

- TRUE if the principal names are the same; FALSE otherwise

Valid flags are:

- *KRB5_PRINCIPAL_COMPARE_IGNORE_REALM* - ignore realm component
- *KRB5_PRINCIPAL_COMPARE_ENTERPRISE* - UPNs as real principals
- *KRB5_PRINCIPAL_COMPARE_CASEFOLD* case-insensitive
- *KRB5_PRINCIPAL_COMPARE_UTF8* - treat principals as UTF-8

See also:

krb5_principal_compare()

krb5_prompter_posix - Prompt user for password.

krb5_error_code **krb5_prompter_posix**(*krb5_context context*, void * *data*, const char * *name*, const char * *banner*, int *num_prompts*, *krb5_prompt prompts*)

param **[in] context** - Library context

data - Unused (callback argument)

[in] name - Name to output during prompt

[in] banner - Banner to output during prompt

[in] num_prompts - Number of prompts in *prompts*

[in] prompts - Array of prompts and replies

retval

- 0 Success

return

- Kerberos error codes

This function is intended to be used as a prompter callback for *krb5_get_init_creds_password()* or *krb5_init_creds_init()* .

Writes *name* and *banner* to stdout, each followed by a newline, then writes each prompt field in the *prompts* array, followed by":", and sets the reply field of the entry to a line of input read from stdin. If the hidden flag is set for a prompt, then terminal echoing is turned off when input is read.

krb5_realm_compare - Compare the realms of two principals.

krb5_boolean **krb5_realm_compare**(*krb5_context context*, *krb5_const_principal princ1*, *krb5_const_principal princ2*)

param **[in] context** - Library context

[in] princ1 - First principal

[in] princ2 - Second principal

retval

- TRUE if the realm names are the same; FALSE otherwise

krb5_responder_get_challenge - Retrieve the challenge data for a given question in the responder context.

const char * **krb5_responder_get_challenge**(*krb5_context ctx*, *krb5_responder_context rctx*, const char * *question*)

param **[in] ctx** - Library context

[in] rctx - Responder context

[in] question - Question name

Return a pointer to a C string containing the challenge for *question* within *rctx* , or NULL if the question is not present in *rctx* . The structure of the question depends on the question name, but will always be printable UTF-8 text. The returned pointer is an alias, valid only as long as the lifetime of *rctx* , and should not be modified or freed by the caller.

Note: New in 1.11

krb5_responder_list_questions - List the question names contained in the responder context.

const char *const * **krb5_responder_list_questions**(*krb5_context ctx*, *krb5_responder_context rctx*)

param **[in] ctx** - Library context

[in] rctx - Responder context

Return a pointer to a null-terminated list of question names which are present in *rctx* . The pointer is an alias, valid only as long as the lifetime of *rctx* , and should not be modified or freed by the caller. A question's challenge can be retrieved using `krb5_responder_get_challenge()` and answered using `krb5_responder_set_answer()` .

Note: New in 1.11

krb5_responder_set_answer - Answer a named question in the responder context.

krb5_error_code **krb5_responder_set_answer**(*krb5_context ctx*, *krb5_responder_context rctx*, const char * *question*, const char * *answer*)

param **[in] ctx** - Library context

[in] rctx - Responder context

[in] question - Question name

[in] answer - The string to set (MUST be printable UTF-8)

retval

- EINVAL question is not present within rctx

This function supplies an answer to *question* within *rctx* . The appropriate form of the answer depends on the question name.

Note: New in 1.11

---

krb5_responder_otp_get_challenge - Decode the KRB5_RESPONDER_QUESTION_OTP to a C struct.

krb5_error_code **krb5_responder_otp_get_challenge**(*krb5_context* *ctx*, *krb5_responder_context* *rctx*, *krb5_responder_otp_challenge* ** *chl*)

param **[in] ctx** - Library context

[in] rctx - Responder context

[out] chl - Challenge structure

A convenience function which parses the KRB5_RESPONDER_QUESTION_OTP question challenge data, making it available in native C. The main feature of this function is the ability to interact with OTP tokens without parsing the JSON.

The returned value must be passed to *krb5_responder_otp_challenge_free()* to be freed.

Note: New in 1.11

---

krb5_responder_otp_set_answer - Answer the KRB5_RESPONDER_QUESTION_OTP question.

krb5_error_code **krb5_responder_otp_set_answer**(*krb5_context* *ctx*, *krb5_responder_context* *rctx*, size_t *ti*, const char * *value*, const char * *pin*)

param **[in] ctx** - Library context

[in] rctx - Responder context

[in] ti - The index of the tokeninfo selected

[in] value - The value to set, or NULL for none

[in] pin - The pin to set, or NULL for none

Note: New in 1.11

---

krb5_responder_otp_challenge_free - Free the value returned by krb5_responder_otp_get_challenge() .

void **krb5_responder_otp_challenge_free**(*krb5_context ctx*, *krb5_responder_context rctx*, *krb5_responder_otp_challenge * chl*)

param **[in] ctx** - Library context

[in] rctx - Responder context

[in] chl - The challenge to free

Note: New in 1.11

---

krb5_responder_pkinit_get_challenge - Decode the KRB5_RESPONDER_QUESTION_PKINIT to a C struct.

krb5_error_code **krb5_responder_pkinit_get_challenge**(*krb5_context ctx*, *krb5_responder_context rctx*, *krb5_responder_pkinit_challenge ** *chl_out*)

param **[in] ctx** - Library context

[in] rctx - Responder context

[out] chl_out - Challenge structure

A convenience function which parses the KRB5_RESPONDER_QUESTION_PKINIT question challenge data, making it available in native C. The main feature of this function is the ability to read the challenge without parsing the JSON.

The returned value must be passed to *krb5_responder_pkinit_challenge_free()* to be freed.

Note: New in 1.12

---

krb5_responder_pkinit_set_answer - Answer the KRB5_RESPONDER_QUESTION_PKINIT question for one identity.

krb5_error_code **krb5_responder_pkinit_set_answer**(*krb5_context ctx*, *krb5_responder_context rctx*, const char * *identity*, const char * *pin*)

param **[in] ctx** - Library context

[in] rctx - Responder context

[in] identity - The identity for which a PIN is being supplied

[in] pin - The provided PIN, or NULL for none

Note: New in 1.12

---

krb5_responder_pkinit_challenge_free - Free the value returned by krb5_responder_pkinit_get_challenge() .

void **krb5_responder_pkinit_challenge_free**(*krb5_context ctx*, *krb5_responder_context rctx*, *krb5_responder_pkinit_challenge* * *chl*)

param **[in] ctx** - Library context

[in] rctx - Responder context

[in] chl - The challenge to free

Note: New in 1.12

---

krb5_set_default_realm - Override the default realm for the specified context.

krb5_error_code **krb5_set_default_realm**(*krb5_context context*, const char * *lrealm*)

param **[in] context** - Library context

[in] lrealm - Realm name for the default realm

retval

- 0 Success

return

- Kerberos error codes

If *lrealm* is NULL, clear the default realm setting.

krb5_set_password - Set a password for a principal using specified credentials.

krb5_error_code **krb5_set_password**(*krb5_context context*, *krb5_creds* * *creds*, const char * *newpw*, *krb5_principal change_password_for*, int * *result_code*, *krb5_data* * *result_code_string*, *krb5_data* * *result_string*)

param **[in] context** - Library context

[in] creds - Credentials for kadmin/changepw service

[in] newpw - New password

[in] change_password_for - Change the password for this principal

[out] result_code - Numeric error code from server

[out] result_code_string - String equivalent to *result_code*

[out] result_string - Data returned from the remote system

retval

- 0 Success and result_code is set to KRB5_KPASSWD_SUCCESS .

return

- Kerberos error codes.

This function uses the credentials *creds* to set the password *newpw* for the principal *change_password_for* . It implements the set password operation of RFC 3244, for interoperability with Microsoft Windows implementations.

The error code and strings are returned in *result_code* , *result_code_string* and *result_string* .

Note: If *change_password_for* is NULL, the change is performed on the current principal. If *change_password_for* is non-null, the change is performed on the principal name passed in *change_password_for* .

---

krb5_set_password_using_ccache - Set a password for a principal using cached credentials.

krb5_error_code **krb5_set_password_using_ccache**(*krb5_context context*, *krb5_ccache ccache*, const char * *newpw*, *krb5_principal change_password_for*, int * *result_code*, *krb5_data* * *result_code_string*, *krb5_data* * *result_string*)

param **[in] context** - Library context

[in] ccache - Credential cache

[in] newpw - New password

[in] change_password_for - Change the password for this principal

[out] result_code - Numeric error code from server

[out] result_code_string - String equivalent to *result_code*

[out] result_string - Data returned from the remote system

retval

- 0 Success

return

- Kerberos error codes

This function uses the cached credentials from *ccache* to set the password *newpw* for the principal *change_password_for* . It implements RFC 3244 set password operation (interoperable with MS Windows implementations) using the credential cache.

The error code and strings are returned in *result_code* , *result_code_string* and *result_string* .

Note: If *change_password_for* is set to NULL, the change is performed on the default principal in *ccache* . If *change_password_for* is non null, the change is performed on the specified principal.

---

krb5_set_principal_realm - Set the realm field of a principal.

krb5_error_code **krb5_set_principal_realm**(*krb5_context context*, *krb5_principal principal*, const char * *realm*)

param **[in] context** - Library context

[in] principal - Principal name

[in] realm - Realm name

retval

- 0 Success

return

- Kerberos error codes

Set the realm name part of *principal* to *realm* , overwriting the previous realm.

krb5_set_trace_callback - Specify a callback function for trace events.

krb5_error_code **krb5_set_trace_callback**(*krb5_context context*, *krb5_trace_callback fn*, void * *cb_data*)

param **[in] context** - Library context

[in] fn - Callback function

[in] cb_data - Callback data

return

- Returns KRB5_TRACE_NOSUPP if tracing is not supported in the library (unless fn is NULL).

Specify a callback for trace events occurring in krb5 operations performed within *context* . *fn* will be invoked with *context* as the first argument, *cb_data* as the last argument, and a pointer to a krb5_trace_info as the second argument. If the trace callback is reset via this function or *context* is destroyed, *fn* will be invoked with a NULL second argument so it can clean up *cb_data* . Supply a NULL value for *fn* to disable trace callbacks within *context* .

Note: This function overrides the information passed through the *KRB5_TRACE* environment variable.

Note: New in 1.9

krb5_set_trace_filename - Specify a file name for directing trace events.

krb5_error_code **krb5_set_trace_filename**(*krb5_context context*, const char * *filename*)

param **[in] context** - Library context

[in] filename - File name

retval

- KRB5_TRACE_NOSUPP Tracing is not supported in the library.

Open *filename* for appending (creating it, if necessary) and set up a callback to write trace events to it.

Note: This function overrides the information passed through the *KRB5_TRACE* environment variable.

Note: New in 1.9

krb5_sname_match - Test whether a principal matches a matching principal.

krb5_boolean **krb5_sname_match**(*krb5_context context*, *krb5_const_principal matching*, *krb5_const_principal princ*)

param **[in] context** - Library context

[in] matching - Matching principal

[in] princ - Principal to test

return

- TRUE if princ matches matching , FALSE otherwise.

If *matching* is NULL, return TRUE. If *matching* is not a matching principal, return the value of krb5_principal_compare(context, matching, princ).

Note: A matching principal is a host-based principal with an empty realm and/or second data component (hostname). Profile configuration may cause the hostname to be ignored even if it is present. A principal matches a matching principal if the former has the same non-empty (and non-ignored) components of the latter.

---

krb5_sname_to_principal - Generate a full principal name from a service name.

krb5_error_code **krb5_sname_to_principal**(*krb5_context context*, const char * *hostname*, const char * *sname*, *krb5_int32 type*, *krb5_principal* * *ret_princ*)

param **[in] context** - Library context

[in] hostname - Host name, or NULL to use local host

[in] sname - Service name, or NULL to use **"host"**

[in] type - Principal type

[out] ret_princ - Generated principal

retval

- 0 Success

return

- Kerberos error codes

This function converts a *hostname* and *sname* into *krb5_principal* structure *ret_princ* . The returned principal will be of the form *sname/hostname@REALM* where REALM is determined by `krb5_get_host_realm()` . In some cases this may be the referral (empty) realm.

The *type* can be one of the following:

- `KRB5_NT_SRV_HST` canonicalizes the host name before looking up the realm and generating the principal.
- `KRB5_NT_UNKNOWN` accepts the hostname as given, and does not canonicalize it.

Use krb5_free_principal to free *ret_princ* when it is no longer needed.

krb5_unparse_name - Convert a krb5_principal structure to a string representation.

krb5_error_code **krb5_unparse_name**(*krb5_context context*, *krb5_const_principal principal*, char ** *name*)

param **[in] context** - Library context

[in] principal - Principal

[out] name - String representation of principal name

retval

- 0 Success

return

- Kerberos error codes

The resulting string representation uses the format and quoting conventions described for *krb5_parse_name()* .

Use *krb5_free_unparsed_name()* to free *name* when it is no longer needed.

krb5_unparse_name_ext - Convert krb5_principal structure to string and length.

krb5_error_code **krb5_unparse_name_ext**(*krb5_context context*, *krb5_const_principal principal*, char ** *name*, unsigned int * *size*)

param **[in] context** - Library context

[in] principal - Principal

[inout] name - String representation of principal name

[inout] size - Size of unparsed name

retval

- 0 Success

return

- Kerberos error codes. On failure name is set to NULL

This function is similar to *krb5_unparse_name()* , but allows the use of an existing buffer for the result. If size is not NULL, then *name* must point to either NULL or an existing buffer of at least the size pointed to by *size* . The buffer will be allocated or resized if necessary, with the new pointer stored into *name* . Whether or not the buffer is resized, the necessary space for the result, including null terminator, will be stored into *size* .

If size is NULL, this function behaves exactly as *krb5_unparse_name()* .

krb5_unparse_name_flags - Convert krb5_principal structure to a string with flags.

krb5_error_code **krb5_unparse_name_flags**(*krb5_context context*, *krb5_const_principal principal*, int *flags*, char ** *name*)

param **[in] context** - Library context

[in] principal - Principal

[in] flags - Flags

[out] name - String representation of principal name

retval

- 0 Success

return

- Kerberos error codes. On failure name is set to NULL

Similar to *krb5_unparse_name()* , this function converts a krb5_principal structure to a string representation.

The following flags are valid:

- *KRB5_PRINCIPAL_UNPARSE_SHORT* - omit realm if it is the local realm
- *KRB5_PRINCIPAL_UNPARSE_NO_REALM* - omit realm
- *KRB5_PRINCIPAL_UNPARSE_DISPLAY* - do not quote special characters

Use *krb5_free_unparsed_name()* to free *name* when it is no longer needed.

krb5_unparse_name_flags_ext - Convert krb5_principal structure to string format with flags.

krb5_error_code **krb5_unparse_name_flags_ext** (*krb5_context context*, *krb5_const_principal principal*, int *flags*, char ** *name*, unsigned int * *size*)

param [in] context - Library context

[in] principal - Principal

[in] flags - Flags

[out] name - Single string format of principal name

[out] size - Size of unparsed name buffer

retval

- 0 Success

return

- Kerberos error codes. On failure name is set to NULL

krb5_us_timeofday - Retrieve the system time of day, in sec and ms, since the epoch.

krb5_error_code **krb5_us_timeofday** (*krb5_context context*, *krb5_timestamp * seconds*, *krb5_int32 * microseconds*)

param [in] context - Library context

[out] seconds - System timeofday, seconds portion

[out] microseconds - System timeofday, microseconds portion

retval

- 0 Success

return

- Kerberos error codes

This function retrieves the system time of day with the context specific time offset adjustment.

krb5_verify_authdata_kdc_issued - Unwrap and verify AD-KDCIssued authorization data.

krb5_error_code **krb5_verify_authdata_kdc_issued**(*krb5_context context*, const *krb5_keyblock* * *key*, const *krb5_authdata* * *ad_kdcissued*, *krb5_principal* * *issuer*, *krb5_authdata* *** *authdata*)

param [in] context - Library context

[in] key - Session key

[in] ad_kdcissued - AD-KDCIssued authorization data to be unwrapped

[out] issuer - Name of issuing principal (or NULL)

[out] authdata - Unwrapped list of authorization data

This function unwraps an AD-KDCIssued authdatum (see RFC 4120 section 5.2.6.2) and verifies its signature against *key* . The issuer field of the authdatum element is returned in *issuer* , and the unwrapped list of authdata is returned in *authdata* .

6.1.2 Rarely used public interfaces

krb5_425_conv_principal - Convert a Kerberos V4 principal to a Kerberos V5 principal.

krb5_error_code **krb5_425_conv_principal**(*krb5_context context*, const char * *name*, const char * *instance*, const char * *realm*, *krb5_principal* * *princ*)

param [in] context - Library context

[in] name - V4 name

[in] instance - V4 instance

[in] realm - Realm

[out] princ - V5 principal

retval

- 0 Success; otherwise - Kerberos error codes

This function builds a *princ* from V4 specification based on given input *name.instance@realm* .

Use `krb5_free_principal()` to free *princ* when it is no longer needed.

krb5_524_conv_principal - Convert a Kerberos V5 principal to a Kerberos V4 principal.

krb5_error_code **krb5_524_conv_principal**(*krb5_context context*, *krb5_const_principal princ*, char * *name*, char * *inst*, char * *realm*)

param [in] context - Library context

[in] princ - V5 Principal

[out] name - V4 principal's name to be filled in

[out] inst - V4 principal's instance name to be filled in

[out] realm - Principal's realm name to be filled in

retval

- 0 Success

- KRB5_INVALID_PRINCIPAL Invalid principal name
- KRB5_CONFIG_CANTOPEN Can't open or find Kerberos configuration file

return

- Kerberos error codes

This function separates a V5 principal *princ* into *name* , *instance* , and *realm* .

krb5_address_compare - Compare two Kerberos addresses.

krb5_boolean **krb5_address_compare** (*krb5_context context*, const *krb5_address* * *addr1*, const *krb5_address* * *addr2*)

param **[in] context** - Library context

[in] addr1 - First address to be compared

[in] addr2 - Second address to be compared

return

- TRUE if the addresses are the same, FALSE otherwise

krb5_address_order - Return an ordering of the specified addresses.

int **krb5_address_order** (*krb5_context context*, const *krb5_address* * *addr1*, const *krb5_address* * *addr2*)

param **[in] context** - Library context

[in] addr1 - First address

[in] addr2 - Second address

retval

- 0 The two addresses are the same
-

krb5_address_search - Search a list of addresses for a specified address.

krb5_boolean **krb5_address_search** (*krb5_context context*, const *krb5_address* * *addr*, *krb5_address* *const * *addrlist*)

param **[in] context** - Library context

[in] addr - Address to search for

[in] addrlist - Address list to be searched (or NULL)

return

- TRUE if addr is listed in addrlist , or addrlist is NULL; FALSE otherwise

Note: If *addrlist* contains only a NetBIOS addresses, it will be treated as a null list.

krb5_allow_weak_crypto - Allow the application to override the profile's allow_weak_crypto setting.

krb5_error_code **krb5_allow_weak_crypto** (*krb5_context* *context*, *krb5_boolean* *enable*)

param [in] context - Library context

[in] enable - Boolean flag

retval

- 0 (always)

This function allows an application to override the allow_weak_crypto setting. It is primarily for use by aklog.

krb5_aname_to_localname - Convert a principal name to a local name.

krb5_error_code **krb5_aname_to_localname** (*krb5_context* *context*, *krb5_const_principal* *aname*, int *lnsize_in*, char * *lname*)

param [in] context - Library context

[in] aname - Principal name

[in] lnsize_in - Space available in *lname*

[out] lname - Local name buffer to be filled in

retval

- 0 Success
- System errors

return

- Kerberos error codes

If *aname* does not correspond to any local account, KRB5_LNAME_NOTRANS is returned. If *lnsize_in* is too small for the local name, KRB5_CONFIG_NOTENUFSPACE is returned.

Local names, rather than principal names, can be used by programs that translate to an environment-specific name (for example, a user account name).

krb5_anonymous_principal - Build an anonymous principal.

krb5_const_principal **krb5_anonymous_principal** (void *None*)

param None

This function returns constant storage that must not be freed.

See also:

KRB5_ANONYMOUS_PRINCSTR

krb5_anonymous_realm - Return an anonymous realm data.

const *krb5_data* * **krb5_anonymous_realm** (void *None*)

param None

This function returns constant storage that must not be freed.

See also:

KRB5_ANONYMOUS_REALMSTR

krb5_appdefault_boolean - Retrieve a boolean value from the appdefaults section of krb5.conf.

void **krb5_appdefault_boolean**(*krb5_context context*, const char * *appname*, const *krb5_data* * *realm*, const char * *option*, int *default_value*, int * *ret_value*)

param **[in] context** - Library context

[in] appname - Application name

[in] realm - Realm name

[in] option - Option to be checked

[in] default_value - Default value to return if no match is found

[out] ret_value - Boolean value of *option*

This function gets the application defaults for *option* based on the given *appname* and/or *realm* .

See also:

krb5_appdefault_string()

krb5_appdefault_string - Retrieve a string value from the appdefaults section of krb5.conf.

void **krb5_appdefault_string**(*krb5_context context*, const char * *appname*, const *krb5_data* * *realm*, const char * *option*, const char * *default_value*, char ** *ret_value*)

param **[in] context** - Library context

[in] appname - Application name

[in] realm - Realm name

[in] option - Option to be checked

[in] default_value - Default value to return if no match is found

[out] ret_value - String value of *option*

This function gets the application defaults for *option* based on the given *appname* and/or *realm* .

See also:

krb5_appdefault_boolean()

krb5_auth_con_free - Free a krb5_auth_context structure.

krb5_error_code **krb5_auth_con_free**(*krb5_context context*, *krb5_auth_context auth_context*)

param **[in] context** - Library context

[in] auth_context - Authentication context to be freed

retval

- 0 (always)

This function frees an auth context allocated by *krb5_auth_con_init()* .

krb5_auth_con_genaddrs - Generate auth context addresses from a connected socket.

krb5_error_code **krb5_auth_con_genaddrs** (*krb5_context context*, *krb5_auth_context auth_context*, int *infd*, int *flags*)

param [in] context - Library context

[in] auth_context - Authentication context

[in] infd - Connected socket descriptor

[in] flags - Flags

retval

- 0 Success; otherwise - Kerberos error codes

This function sets the local and/or remote addresses in *auth_context* based on the local and remote endpoints of the socket *infd* . The following flags determine the operations performed:

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

krb5_auth_con_get_checksum_func - Get the checksum callback from an auth context.

krb5_error_code **krb5_auth_con_get_checksum_func** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_mk_req_checksum_func* * *func*, void ** *data*)

param [in] context - Library context

[in] auth_context - Authentication context

[out] func - Checksum callback

[out] data - Callback argument

retval

- 0 (always)

krb5_auth_con_getaddrs - Retrieve address fields from an auth context.

krb5_error_code **krb5_auth_con_getaddrs** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_address* ** *local_addr*, *krb5_address* ** *remote_addr*)

param [in] context - Library context

[in] auth_context - Authentication context

[out] local_addr - Local address (NULL if not needed)

[out] remote_addr - Remote address (NULL if not needed)

retval

- 0 Success; otherwise - Kerberos error codes

krb5_auth_con_getauthenticator - Retrieve the authenticator from an auth context.

krb5_error_code **krb5_auth_con_getauthenticator**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_authenticator ** authenticator*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[out] authenticator - Authenticator

retval

- 0 Success. Otherwise - Kerberos error codes

Use *krb5_free_authenticator()* to free *authenticator* when it is no longer needed.

krb5_auth_con_getflags - Retrieve flags from a krb5_auth_context structure.

krb5_error_code **krb5_auth_con_getflags**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_int32 * flags*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[out] flags - Flags bit mask

retval

- 0 (always)

Valid values for *flags* are:

- *KRB5_AUTH_CONTEXT_DO_TIME* Use timestamps
- *KRB5_AUTH_CONTEXT_RET_TIME* Save timestamps
- *KRB5_AUTH_CONTEXT_DO_SEQUENCE* Use sequence numbers
- *KRB5_AUTH_CONTEXT_RET_SEQUENCE* Save sequence numbers

krb5_auth_con_getkey - Retrieve the session key from an auth context as a keyblock.

krb5_error_code **krb5_auth_con_getkey**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_keyblock ** keyblock*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[out] keyblock - Session key

retval

- 0 Success. Otherwise - Kerberos error codes

This function creates a keyblock containing the session key from *auth_context* . Use *krb5_free_keyblock()* to free *keyblock* when it is no longer needed

krb5_auth_con_getkey_k - Retrieve the session key from an auth context.

krb5_error_code **krb5_auth_con_getkey_k** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_key * key*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[out] key - Session key

retval

- 0 (always)

This function sets *key* to the session key from *auth_context* . Use *krb5_k_free_key()* to release *key* when it is no longer needed.

krb5_auth_con_getlocalseqnumber - Retrieve the local sequence number from an auth context.

krb5_error_code **krb5_auth_con_getlocalseqnumber** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_int32 * seqnumber*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[out] seqnumber - Local sequence number

retval

- 0 Success; otherwise - Kerberos error codes

Retrieve the local sequence number from *auth_context* and return it in *seqnumber* . The *KRB5_AUTH_CONTEXT_DO_SEQUENCE* flag must be set in *auth_context* for this function to be useful.

krb5_auth_con_getrcache - Retrieve the replay cache from an auth context.

krb5_error_code **krb5_auth_con_getrcache** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_rcache * rcache*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[out] rcache - Replay cache handle

retval

- 0 (always)

This function fetches the replay cache from *auth_context* . The caller should not close *rcache* .

krb5_auth_con_getrecvsubkey - Retrieve the receiving subkey from an auth context as a keyblock.

krb5_error_code **krb5_auth_con_getrecvsubkey** (*krb5_context ctx*, *krb5_auth_context ac*, *krb5_keyblock ** keyblock*)

param **[in] ctx** - Library context

[in] ac - Authentication context

[out] keyblock - Receiving subkey

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a keyblock containing the receiving subkey from *auth_context* . Use `krb5_free_keyblock()` to free *keyblock* when it is no longer needed.

krb5_auth_con_getrecvsubkey_k - Retrieve the receiving subkey from an auth context as a keyblock.

krb5_error_code **krb5_auth_con_getrecvsubkey_k**(*krb5_context ctx*, *krb5_auth_context ac*, *krb5_key * key*)

param **[in] ctx** - Library context

[in] ac - Authentication context

[out] key - Receiving subkey

retval

- 0 Success; otherwise - Kerberos error codes

This function sets *key* to the receiving subkey from *auth_context* . Use `krb5_k_free_key()` to release *key* when it is no longer needed.

krb5_auth_con_getremoteseqnumber - Retrieve the remote sequence number from an auth context.

krb5_error_code **krb5_auth_con_getremoteseqnumber**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_int32 * seqnumber*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[out] seqnumber - Remote sequence number

retval

- 0 Success; otherwise - Kerberos error codes

Retrieve the remote sequence number from *auth_context* and return it in *seqnumber* . The `KRB5_AUTH_CONTEXT_DO_SEQUENCE` flag must be set in *auth_context* for this function to be useful.

krb5_auth_con_getsendsubkey - Retrieve the send subkey from an auth context as a keyblock.

krb5_error_code **krb5_auth_con_getsendsubkey**(*krb5_context ctx*, *krb5_auth_context ac*, *krb5_keyblock ** keyblock*)

param **[in] ctx** - Library context

[in] ac - Authentication context

[out] keyblock - Send subkey

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a keyblock containing the send subkey from *auth_context* . Use `krb5_free_keyblock()` to free *keyblock* when it is no longer needed.

krb5_auth_con_getsendsubkey_k - Retrieve the send subkey from an auth context.

krb5_error_code **krb5_auth_con_getsendsubkey_k** (*krb5_context ctx*, *krb5_auth_context ac*, *krb5_key* * *key*)

param [in] ctx - Library context

[in] ac - Authentication context

[out] key - Send subkey

retval

- 0 Success; otherwise - Kerberos error codes

This function sets *key* to the send subkey from *auth_context* . Use `krb5_k_free_key()` to release *key* when it is no longer needed.

krb5_auth_con_init - Create and initialize an authentication context.

krb5_error_code **krb5_auth_con_init** (*krb5_context context*, *krb5_auth_context* * *auth_context*)

param [in] context - Library context

[out] auth_context - Authentication context

retval

- 0 Success; otherwise - Kerberos error codes

This function creates an authentication context to hold configuration and state relevant to krb5 functions for authenticating principals and protecting messages once authentication has occurred.

By default, flags for the context are set to enable the use of the replay cache (`KRB5_AUTH_CONTEXT_DO_TIME`), but not sequence numbers. Use `krb5_auth_con_setflags()` to change the flags.

The allocated *auth_context* must be freed with `krb5_auth_con_free()` when it is no longer needed.

krb5_auth_con_set_checksum_func - Set a checksum callback in an auth context.

krb5_error_code **krb5_auth_con_set_checksum_func** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_mk_req_checksum_func func*, void * *data*)

param [in] context - Library context

[in] auth_context - Authentication context

[in] func - Checksum callback

[in] data - Callback argument

retval

- 0 (always)

Set a callback to obtain checksum data in *krb5_mk_req()* . The callback will be invoked after the subkey and local sequence number are stored in *auth_context* .

krb5_auth_con_set_req_cksumtype - Set checksum type in an an auth context.

krb5_error_code **krb5_auth_con_set_req_cksumtype**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_cksumtype cksumtype*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] cksumtype - Checksum type

retval

- 0 Success. Otherwise - Kerberos error codes

This function sets the checksum type in *auth_context* to be used by *krb5_mk_req()* for the authenticator checksum.

krb5_auth_con_setaddrs - Set the local and remote addresses in an auth context.

krb5_error_code **krb5_auth_con_setaddrs**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_address * local_addr*, *krb5_address * remote_addr*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] local_addr - Local address

[in] remote_addr - Remote address

retval

- 0 Success; otherwise - Kerberos error codes

This function releases the storage assigned to the contents of the local and remote addresses of *auth_context* and then sets them to *local_addr* and *remote_addr* respectively.

See also:

krb5_auth_con_genaddrs()

krb5_auth_con_setflags - Set a flags field in a krb5_auth_context structure.

krb5_error_code **krb5_auth_con_setflags**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_int32 flags*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] flags - Flags bit mask

retval

- 0 (always)

Valid values for *flags* are:

- *KRB5_AUTH_CONTEXT_DO_TIME* Use timestamps

- *KRB5_AUTH_CONTEXT_RET_TIME* Save timestamps
- *KRB5_AUTH_CONTEXT_DO_SEQUENCE* Use sequence numbers
- *KRB5_AUTH_CONTEXT_RET_SEQUENCE* Save sequence numbers

krb5_auth_con_setports - Set local and remote port fields in an auth context.

krb5_error_code **krb5_auth_con_setports**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_address * local_port*, *krb5_address * remote_port*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] local_port - Local port

[in] remote_port - Remote port

retval

- 0 Success; otherwise - Kerberos error codes

This function releases the storage assigned to the contents of the local and remote ports of *auth_context* and then sets them to *local_port* and *remote_port* respectively.

See also:

krb5_auth_con_genaddrs()

krb5_auth_con_setrcache - Set the replay cache in an auth context.

krb5_error_code **krb5_auth_con_setrcache**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_rcache rcache*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] rcache - Replay cache haddle

retval

- 0 Success; otherwise - Kerberos error codes

This function sets the replay cache in *auth_context* to *rcache* . *rcache* will be closed when *auth_context* is freed, so the caller should relinquish that responsibility.

krb5_auth_con_setrecvsubkey - Set the receiving subkey in an auth context with a keyblock.

krb5_error_code **krb5_auth_con_setrecvsubkey**(*krb5_context ctx*, *krb5_auth_context ac*, *krb5_keyblock * keyblock*)

param **[in] ctx** - Library context

[in] ac - Authentication context

[in] keyblock - Receiving subkey

retval

- 0 Success; otherwise - Kerberos error codes

This function sets the receiving subkey in *ac* to a copy of *keyblock* .

krb5_auth_con_setrecvsubkey_k - Set the receiving subkey in an auth context.

krb5_error_code **krb5_auth_con_setrecvsubkey_k**(*krb5_context ctx*, *krb5_auth_context ac*, *krb5_key key*)

param **[in] ctx** - Library context

[in] ac - Authentication context

[in] key - Receiving subkey

retval

- 0 Success; otherwise - Kerberos error codes

This function sets the receiving subkey in *ac* to *key* , incrementing its reference count.

Note: New in 1.9

krb5_auth_con_setsendsubkey - Set the send subkey in an auth context with a keyblock.

krb5_error_code **krb5_auth_con_setsendsubkey**(*krb5_context ctx*, *krb5_auth_context ac*, *krb5_keyblock * keyblock*)

param **[in] ctx** - Library context

[in] ac - Authentication context

[in] keyblock - Send subkey

retval

- 0 Success. Otherwise - Kerberos error codes

This function sets the send subkey in *ac* to a copy of *keyblock* .

krb5_auth_con_setsendsubkey_k - Set the send subkey in an auth context.

krb5_error_code **krb5_auth_con_setsendsubkey_k**(*krb5_context ctx*, *krb5_auth_context ac*, *krb5_key key*)

param **[in] ctx** - Library context

[in] ac - Authentication context

[out] key - Send subkey

retval

- 0 Success; otherwise - Kerberos error codes

This function sets the send subkey in *ac* to *key* , incrementing its reference count.

Note: New in 1.9

krb5_auth_con_setuseruserkey - Set the session key in an auth context.

krb5_error_code **krb5_auth_con_setuseruserkey**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_keyblock * keyblock*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] keyblock - User key

retval

- 0 Success; otherwise - Kerberos error codes

krb5_cc_cache_match - Find a credential cache with a specified client principal.

krb5_error_code **krb5_cc_cache_match**(*krb5_context context*, *krb5_principal client*, *krb5_ccache * cache_out*)

param **[in] context** - Library context

[in] client - Client principal

[out] cache_out - Credential cache handle

retval

- 0 Success
- KRB5_CC_NOTFOUND None

Find a cache within the collection whose default principal is *client* . Use *krb5_cc_close* to close *ccache* when it is no longer needed.

Note: New in 1.10

---

krb5_cc_copy_creds - Copy a credential cache.

krb5_error_code **krb5_cc_copy_creds**(*krb5_context context*, *krb5_ccache incc*, *krb5_ccache outcc*)

param **[in] context** - Library context

[in] incc - Credential cache to be copied

[out] outcc - Copy of credential cache to be filled in

retval

- 0 Success; otherwise - Kerberos error codes

krb5_cc_end_seq_get - Finish a series of sequential processing credential cache entries.

krb5_error_code **krb5_cc_end_seq_get**(*krb5_context context*, *krb5_ccache cache*, *krb5_cc_cursor * cursor*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[in] cursor - Cursor

retval

- 0 (always)

This function finishes processing credential cache entries and invalidates *cursor* .

See also:

krb5_cc_start_seq_get() , *krb5_cc_next_cred()*

krb5_cc_get_config - Get a configuration value from a credential cache.

krb5_error_code **krb5_cc_get_config** (*krb5_context context*, *krb5_ccache id*, *krb5_const_principal principal*, const char * *key*, *krb5_data* * *data*)

param **[in] context** - Library context

[in] id - Credential cache handle

[in] principal - Configuration for this principal; if NULL, global for the whole cache

[in] key - Name of config variable

[out] data - Data to be fetched

retval

- 0 Success

return

- Kerberos error codes

Use *krb5_free_data_contents()* to free *data* when it is no longer needed.

krb5_cc_get_flags - Retrieve flags from a credential cache structure.

krb5_error_code **krb5_cc_get_flags** (*krb5_context context*, *krb5_ccache cache*, *krb5_flags * flags*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[out] flags - Flag bit mask

retval

- 0 Success; otherwise - Kerberos error codes

Warning: For memory credential cache always returns a flag mask of 0.

krb5_cc_get_full_name - Retrieve the full name of a credential cache.

krb5_error_code **krb5_cc_get_full_name** (*krb5_context context*, *krb5_ccache cache*, char ** *full-name_out*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[out] fullname_out - Full name of cache

Use *krb5_free_string()* to free *fullname_out* when it is no longer needed.

Note: New in 1.10

---

krb5_cc_move - Move a credential cache.

krb5_error_code **krb5_cc_move** (*krb5_context context*, *krb5_ccache src*, *krb5_ccache dst*)

param **[in] context** - Library context

[in] src - The credential cache to move the content from

[in] dst - The credential cache to move the content to

retval

- 0 Success; src is closed.

return

- Kerberos error codes; src is still allocated.

This function reinitializes *dst* and populates it with the credentials and default principal of *src* ; then, if successful, destroys *src* .

krb5_cc_next_cred - Retrieve the next entry from the credential cache.

krb5_error_code **krb5_cc_next_cred** (*krb5_context context*, *krb5_ccache cache*, *krb5_cc_cursor * cursor*, *krb5_creds * creds*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[in] cursor - Cursor

[out] creds - Next credential cache entry

retval

- 0 Success; otherwise - Kerberos error codes

This function fills in *creds* with the next entry in *cache* and advances *cursor* .

Use *krb5_free_cred_contents()* to free *creds* when it is no longer needed.

See also:

krb5_cc_start_seq_get() , krb5_end_seq_get()

krb5_cc_remove_cred - Remove credentials from a credential cache.

krb5_error_code **krb5_cc_remove_cred**(*krb5_context context*, *krb5_ccache cache*, *krb5_flags flags*, *krb5_creds * creds*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[in] flags - Bitwise-ORed search flags

[in] creds - Credentials to be matched

retval

- KRB5_CC_NOSUPP Not implemented for this cache type

return

- No matches found; Data cannot be deleted; Kerberos error codes

This function accepts the same flag values as *krb5_cc_retrieve_cred()* .

Warning: This function is not implemented for some cache types.

krb5_cc_retrieve_cred - Retrieve a specified credentials from a credential cache.

krb5_error_code **krb5_cc_retrieve_cred**(*krb5_context context*, *krb5_ccache cache*, *krb5_flags flags*, *krb5_creds * mcreds*, *krb5_creds * creds*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[in] flags - Flags bit mask

[in] mcreds - Credentials to match

[out] creds - Credentials matching the requested value

retval

- 0 Success; otherwise - Kerberos error codes

This function searches a credential cache for credentials matching *mcreds* and returns it if found.

Valid values for *flags* are:

- *KRB5_TC_MATCH_TIMES* The requested lifetime must be at least as great as in *mcreds* .
- *KRB5_TC_MATCH_IS_SKEY* The *is_skey* field much match exactly.
- *KRB5_TC_MATCH_FLAGS* Flags set in *mcreds* must be set.
- *KRB5_TC_MATCH_TIMES_EXACT* The requested lifetime must match exactly.
- *KRB5_TC_MATCH_FLAGS_EXACT* Flags must match exactly.
- *KRB5_TC_MATCH_AUTHDATA* The authorization data must match.
- *KRB5_TC_MATCH_SRV_NAMEONLY* Only the name portion of the principal name must match, not the realm.
- *KRB5_TC_MATCH_2ND_TKT* The second tickets must match.
- *KRB5_TC_MATCH_KTYPE* The encryption key types must match.

- *KRB5_TC_SUPPORTED_KTYPES* Check all matching entries that have any supported encryption type and return the one with the encryption type listed earliest.

Use *krb5_free_cred_contents()* to free *creds* when it is no longer needed.

krb5_cc_select - Select a credential cache to use with a server principal.

krb5_error_code **krb5_cc_select** (*krb5_context context*, *krb5_principal server*, *krb5_ccache * cache_out*, *krb5_principal * princ_out*)

param [in] context - Library context

[in] server - Server principal

[out] cache_out - Credential cache handle

[out] princ_out - Client principal

return

- If an appropriate cache is found, 0 is returned, cache_out is set to the selected cache, and princ_out is set to the default principal of that cache.

Select a cache within the collection containing credentials most appropriate for use with *server* , according to configured rules and heuristics.

Use *krb5_cc_close()* to release *cache_out* when it is no longer needed. Use *krb5_free_principal()* to release *princ_out* when it is no longer needed. Note that *princ_out* is set in some error conditions.

If the appropriate client principal can be authoritatively determined but the cache collection contains no credentials for that principal, then KRB5_CC_NOTFOUND is returned, *cache_out* is set to NULL, and *princ_out* is set to the appropriate client principal.

If no configured mechanism can determine the appropriate cache or principal, KRB5_CC_NOTFOUND is returned and *cache_out* and *princ_out* are set to NULL.

Any other error code indicates a fatal error in the processing of a cache selection mechanism.

Note: New in 1.10

---

krb5_cc_set_config - Store a configuration value in a credential cache.

krb5_error_code **krb5_cc_set_config** (*krb5_context context*, *krb5_ccache id*, *krb5_const_principal principal*, const char * *key*, *krb5_data * data*)

param [in] context - Library context

[in] id - Credential cache handle

[in] principal - Configuration for a specific principal; if NULL, global for the whole cache

[in] key - Name of config variable

[in] data - Data to store, or NULL to remove

retval

- 0 Success

return

- Kerberos error codes

Warning: Before version 1.10 *data* was assumed to be always non-null.

Note: Existing configuration under the same key is over-written.

krb5_cc_set_default_name - Set the default credential cache name.

krb5_error_code **krb5_cc_set_default_name** (*krb5_context context*, const char * *name*)

param **[in] context** - Library context

[in] name - Default credential cache name or NULL

retval

- 0 Success
- KV5M_CONTEXT Bad magic number for _krb5_context structure

return

- Kerberos error codes

Set the default credential cache name to *name* for future operations using *context* . If *name* is NULL, clear any previous application-set default name and forget any cached value of the default name for *context* .

Calls to this function invalidate the result of any previous calls to `krb5_cc_default_name()` using *context* .

krb5_cc_set_flags - Set options flags on a credential cache.

krb5_error_code **krb5_cc_set_flags** (*krb5_context context*, *krb5_ccache cache*, *krb5_flags flags*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[in] flags - Flag bit mask

retval

- 0 Success; otherwise - Kerberos error codes

This function resets *cache* flags to *flags* .

krb5_cc_start_seq_get - Prepare to sequentially read every credential in a credential cache.

krb5_error_code **krb5_cc_start_seq_get** (*krb5_context context*, *krb5_ccache cache*, *krb5_cc_cursor * cursor*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[out] cursor - Cursor

retval

- 0 Success; otherwise - Kerberos error codes

krb5_cc_end_seq_get() must be called to complete the retrieve operation.

Note: If the cache represented by *cache* is modified between the time of the call to this function and the time of the final *krb5_cc_end_seq_get()* , these changes may not be reflected in the results of *krb5_cc_next_cred()* calls.

---

krb5_cc_store_cred - Store credentials in a credential cache.

krb5_error_code **krb5_cc_store_cred** (*krb5_context context*, *krb5_ccache cache*, *krb5_creds * creds*)

param **[in] context** - Library context

[in] cache - Credential cache handle

[in] creds - Credentials to be stored in cache

retval

- 0 Success

return

- Permission errors; storage failure errors; Kerberos error codes

This function stores *creds* into *cache* . If *creds->server* and the server in the decoded ticket *creds->ticket* differ, the credentials will be stored under both server principal names.

krb5_cc_support_switch - Determine whether a credential cache type supports switching.

krb5_boolean **krb5_cc_support_switch** (*krb5_context context*, const char * *type*)

param **[in] context** - Library context

[in] type - Credential cache type

retval

- TRUE if type supports switching
- FALSE if it does not or is not a valid credential cache type.

Note: New in 1.10

---

krb5_cc_switch - Make a credential cache the primary cache for its collection.

krb5_error_code **krb5_cc_switch** (*krb5_context context*, *krb5_ccache cache*)

param **[in] context** - Library context

[in] cache - Credential cache handle

retval

- 0 Success, or the type of cache doesn't support switching

return

- Kerberos error codes

If the type of *cache* supports it, set *cache* to be the primary credential cache for the collection it belongs to.

krb5_cccol_cursor_free - Free a credential cache collection cursor.

krb5_error_code **krb5_cccol_cursor_free** (*krb5_context context*, *krb5_cccol_cursor * cursor*)

param [in] context - Library context

[in] cursor - Cursor

retval

- 0 Success; otherwise - Kerberos error codes

See also:

krb5_cccol_cursor_new() , *krb5_cccol_cursor_next()*

krb5_cccol_cursor_new - Prepare to iterate over the collection of known credential caches.

krb5_error_code **krb5_cccol_cursor_new** (*krb5_context context*, *krb5_cccol_cursor * cursor*)

param [in] context - Library context

[out] cursor - Cursor

retval

- 0 Success; otherwise - Kerberos error codes

Get a new cache iteration *cursor* that will iterate over all known credential caches independent of type.

Use *krb5_cccol_cursor_free()* to release *cursor* when it is no longer needed.

See also:

krb5_cccol_cursor_next()

krb5_cccol_cursor_next - Get the next credential cache in the collection.

krb5_error_code **krb5_cccol_cursor_next** (*krb5_context context*, *krb5_cccol_cursor cursor*, *krb5_ccache * ccache*)

param [in] context - Library context

[in] cursor - Cursor

[out] ccache - Credential cache handle

retval

- 0 Success; otherwise - Kerberos error codes

Use *krb5_cc_close()* to close *ccache* when it is no longer needed.

See also:

krb5_cccol_cursor_new() , *krb5_cccol_cursor_free()*

Note: When all caches are iterated over and the end of the list is reached, *ccache* is set to NULL.

krb5_cccol_have_content - Check if the credential cache collection contains any credentials.

krb5_error_code **krb5_cccol_have_content** (*krb5_context context*)

param **[in] context** - Library context

retval

- 0 Credentials are available in the collection
- KRB5_CC_NOTFOUND The collection contains no credentials

Note: New in 1.11

krb5_clear_error_message - Clear the extended error message in a context.

void **krb5_clear_error_message** (*krb5_context ctx*)

param **[in] ctx** - Library context

This function unsets the extended error message in a context, to ensure that it is not mistakenly applied to another occurrence of the same error code.

krb5_check_clockskew - Check if a timestamp is within the allowed clock skew of the current time.

krb5_error_code **krb5_check_clockskew** (*krb5_context context*, *krb5_timestamp date*)

param **[in] context** - Library context

[in] date - Timestamp to check

retval

- 0 Success
- KRB5KRB_AP_ERR_SKEW date is not within allowable clock skew

This function checks if *date* is close enough to the current time according to the configured allowable clock skew.

Note: New in 1.10

krb5_copy_addresses - Copy an array of addresses.

krb5_error_code **krb5_copy_addresses** (*krb5_context context*, *krb5_address* *const * *inaddr*, *krb5_address* *** *outaddr*)

param **[in] context** - Library context

[in] inaddr - Array of addresses to be copied

[out] outaddr - Copy of array of addresses

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new address array containing a copy of *inaddr* . Use *krb5_free_addresses()* to free *outaddr* when it is no longer needed.

krb5_copy_authdata - Copy an authorization data list.

krb5_error_code **krb5_copy_authdata**(*krb5_context context*, *krb5_authdata* *const * *in_authdat*, *krb5_authdata* *** *out*)

param **[in] context** - Library context

[in] in_authdat - List of *krb5_authdata* structures

[out] out - New array of *krb5_authdata* structures

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new authorization data list containing a copy of *in_authdat* , which must be null-terminated. Use `krb5_free_authdata()` to free *out* when it is no longer needed.

Note: The last array entry in *in_authdat* must be a NULL pointer.

krb5_copy_authenticator - Copy a krb5_authenticator structure.

krb5_error_code **krb5_copy_authenticator**(*krb5_context context*, const *krb5_authenticator* * *authfrom*, *krb5_authenticator* ** *authto*)

param **[in] context** - Library context

[in] authfrom - krb5_authenticator structure to be copied

[out] authto - Copy of krb5_authenticator structure

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new krb5_authenticator structure with the content of *authfrom* . Use `krb5_free_authenticator()` to free *authto* when it is no longer needed.

krb5_copy_checksum - Copy a krb5_checksum structure.

krb5_error_code **krb5_copy_checksum**(*krb5_context context*, const *krb5_checksum* * *ckfrom*, *krb5_checksum* ** *ckto*)

param **[in] context** - Library context

[in] ckfrom - Checksum to be copied

[out] ckto - Copy of krb5_checksum structure

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new krb5_checksum structure with the contents of *ckfrom* . Use `krb5_free_checksum()` to free *ckto* when it is no longer needed.

krb5_copy_context - Copy a krb5_context structure.

krb5_error_code **krb5_copy_context** (*krb5_context ctx*, *krb5_context * nctx_out*)

param **[in] ctx** - Library context

[out] nctx_out - New context structure

retval

- 0 Success

return

- Kerberos error codes

The newly created context must be released by calling `krb5_free_context()` when it is no longer needed.

krb5_copy_creds - Copy a krb5_creds structure.

krb5_error_code **krb5_copy_creds** (*krb5_context context*, const *krb5_creds * incred*, *krb5_creds ** *outcred*)

param **[in] context** - Library context

[in] incred - Credentials structure to be copied

[out] outcred - Copy of *incred*

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new credential with the contents of *incred* . Use `krb5_free_creds()` to free *outcred* when it is no longer needed.

krb5_copy_data - Copy a krb5_data object.

krb5_error_code **krb5_copy_data** (*krb5_context context*, const *krb5_data * indata*, *krb5_data ** *outdata*)

param **[in] context** - Library context

[in] indata - Data object to be copied

[out] outdata - Copy of *indata*

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new krb5_data object with the contents of *indata* . Use `krb5_free_data()` to free *outdata* when it is no longer needed.

krb5_copy_error_message - Copy the most recent extended error message from one context to another.

void **krb5_copy_error_message** (*krb5_context dest_ctx*, *krb5_context src_ctx*)

param **[in] dest_ctx** - Library context to copy message to

[in] src_ctx - Library context with current message

krb5_copy_keyblock - Copy a keyblock.

krb5_error_code **krb5_copy_keyblock** (*krb5_context context*, const *krb5_keyblock* * *from*, *krb5_keyblock* ** *to*)

param **[in] context** - Library context

[in] from - Keyblock to be copied

[out] to - Copy of keyblock *from*

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new keyblock with the same contents as *from* . Use `krb5_free_keyblock()` to free *to* when it is no longer needed.

krb5_copy_keyblock_contents - Copy the contents of a keyblock.

krb5_error_code **krb5_copy_keyblock_contents** (*krb5_context context*, const *krb5_keyblock* * *from*, *krb5_keyblock* * *to*)

param **[in] context** - Library context

[in] from - Key to be copied

[out] to - Output key

retval

- 0 Success; otherwise - Kerberos error codes

This function copies the contents of *from* to *to* . Use `krb5_free_keyblock_contents()` to free *to* when it is no longer needed.

krb5_copy_principal - Copy a principal.

krb5_error_code **krb5_copy_principal** (*krb5_context context*, *krb5_const_principal inprinc*, *krb5_principal* * *outprinc*)

param **[in] context** - Library context

[in] inprinc - Principal to be copied

[out] outprinc - Copy of *inprinc*

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new principal structure with the contents of *inprinc* . Use `krb5_free_principal()` to free *outprinc* when it is no longer needed.

krb5_copy_ticket - Copy a krb5_ticket structure.

krb5_error_code **krb5_copy_ticket** (*krb5_context context*, const *krb5_ticket* * *from*, *krb5_ticket* ** *pto*)

param **[in] context** - Library context

[in] from - Ticket to be copied

[out] pto - Copy of ticket

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new krb5_ticket structure containing the contents of *from* . Use `krb5_free_ticket()` to free *pto* when it is no longer needed.

krb5_find_authdata - Find authorization data elements.

krb5_error_code **krb5_find_authdata** (*krb5_context context*, *krb5_authdata* *const * *ticket_authdata*, *krb5_authdata* *const * *ap_req_authdata*, *krb5_authdatatype ad_type*, *krb5_authdata* *** *results*)

param **[in] context** - Library context

[in] ticket_authdata - Authorization data list from ticket

[in] ap_req_authdata - Authorization data list from AP request

[in] ad_type - Authorization data type to find

[out] results - List of matching entries

This function searches *ticket_authdata* and *ap_req_authdata* for elements of type *ad_type* . Either input list may be NULL, in which case it will not be searched; otherwise, the input lists must be terminated by NULL entries. This function will search inside AD-IF-RELEVANT containers if found in either list. Use `krb5_free_authdata()` to free *results* when it is no longer needed.

Note: New in 1.10

---

krb5_free_addresses - Free the data stored in array of addresses.

void **krb5_free_addresses** (*krb5_context context*, *krb5_address* ** *val*)

param **[in] context** - Library context

[in] val - Array of addresses to be freed

This function frees the contents of *val* and the array itself.

Note: The last entry in the array must be a NULL pointer.

---

krb5_free_ap_rep_enc_part - Free a krb5_ap_rep_enc_part structure.

void **krb5_free_ap_rep_enc_part** (*krb5_context context*, *krb5_ap_rep_enc_part* * *val*)

param **[in] context** - Library context

[in] val - AP-REP enc part to be freed

This function frees the contents of *val* and the structure itself.

krb5_free_authdata - Free the storage assigned to array of authentication data.

void **krb5_free_authdata** (*krb5_context context*, *krb5_authdata ** val*)

param **[in] context** - Library context

[in] val - Array of authentication data to be freed

This function frees the contents of *val* and the array itself.

Note: The last entry in the array must be a NULL pointer.

---

krb5_free_authenticator - Free a krb5_authenticator structure.

void **krb5_free_authenticator** (*krb5_context context*, *krb5_authenticator * val*)

param **[in] context** - Library context

[in] val - Authenticator structure to be freed

This function frees the contents of *val* and the structure itself.

krb5_free_cred_contents - Free the contents of a krb5_creds structure.

void **krb5_free_cred_contents** (*krb5_context context*, *krb5_creds * val*)

param **[in] context** - Library context

[in] val - Credential structure to free contents of

This function frees the contents of *val* , but not the structure itself.

krb5_free_creds - Free a krb5_creds structure.

void **krb5_free_creds** (*krb5_context context*, *krb5_creds * val*)

param **[in] context** - Library context

[in] val - Credential structure to be freed.

This function frees the contents of *val* and the structure itself.

krb5_free_data - Free a krb5_data structure.

void **krb5_free_data** (*krb5_context context*, *krb5_data * val*)

param **[in] context** - Library context

[in] val - Data structure to be freed

This function frees the contents of *val* and the structure itself.

krb5_free_data_contents - Free the contents of a krb5_data structure and zero the data field.

void **krb5_free_data_contents**(*krb5_context context*, *krb5_data* * *val*)

param **[in] context** - Library context

[in] val - Data structure to free contents of

This function frees the contents of *val* , but not the structure itself.

krb5_free_default_realm - Free a default realm string returned by krb5_get_default_realm() .

void **krb5_free_default_realm**(*krb5_context context*, char * *lrealm*)

param **[in] context** - Library context

[in] lrealm - Realm to be freed

krb5_free_enctypes - Free an array of encryption types.

void **krb5_free_enctypes**(*krb5_context context*, *krb5_enctype* * *val*)

param **[in] context** - Library context

[in] val - Array of enctypes to be freed

Note: New in 1.12

---

krb5_free_error - Free an error allocated by krb5_read_error() or krb5_sendauth() .

void **krb5_free_error**(*krb5_context context*, *krb5_error* * *val*)

param **[in] context** - Library context

[in] val - Error data structure to be freed

This function frees the contents of *val* and the structure itself.

krb5_free_host_realm - Free the memory allocated by krb5_get_host_realm() .

krb5_error_code **krb5_free_host_realm**(*krb5_context context*, char *const * *realmlist*)

param **[in] context** - Library context

[in] realmlist - List of realm names to be released

retval

- 0 Success

return

- Kerberos error codes

krb5_free_keyblock - Free a krb5_keyblock structure.

void **krb5_free_keyblock** (*krb5_context context*, *krb5_keyblock * val*)

param **[in] context** - Library context

[in] val - Keyblock to be freed

This function frees the contents of *val* and the structure itself.

krb5_free_keyblock_contents - Free the contents of a krb5_keyblock structure.

void **krb5_free_keyblock_contents** (*krb5_context context*, *krb5_keyblock * key*)

param **[in] context** - Library context

[in] key - Keyblock to be freed

This function frees the contents of *key* , but not the structure itself.

krb5_free_keytab_entry_contents - Free the contents of a key table entry.

krb5_error_code **krb5_free_keytab_entry_contents** (*krb5_context context*, *krb5_keytab_entry * entry*)

param **[in] context** - Library context

[in] entry - Key table entry whose contents are to be freed

retval

- 0 Success; otherwise - Kerberos error codes

Note: The pointer is not freed.

---

krb5_free_string - Free a string allocated by a krb5 function.

void **krb5_free_string** (*krb5_context context*, char * *val*)

param **[in] context** - Library context

[in] val - String to be freed

Note: New in 1.10

---

krb5_free_ticket - Free a ticket.

void **krb5_free_ticket** (*krb5_context context*, *krb5_ticket * val*)

param **[in] context** - Library context

[in] val - Ticket to be freed

This function frees the contents of *val* and the structure itself.

krb5_free_unparsed_name - Free a string representation of a principal.

void **krb5_free_unparsed_name**(*krb5_context context*, char * *val*)

param **[in] context** - Library context

[in] val - Name string to be freed

krb5_get_etype_info - Retrieve enctype, salt and s2kparams from KDC.

krb5_error_code **krb5_get_etype_info**(*krb5_context context*, *krb5_principal principal*, *krb5_get_init_creds_opt * opt*, *krb5_enctype * enctype_out*, *krb5_data * salt_out*, *krb5_data * s2kparams_out*)

param **[in] context** - Library context

[in] principal - Principal whose information is requested

[in] opt - Initial credential options

[out] enctype_out - The enctype chosen by KDC

[out] salt_out - Salt returned from KDC

[out] s2kparams_out - String-to-key parameters returned from KDC

retval

- 0 Success

return

- A Kerberos error code

Send an initial ticket request for *principal* and extract the encryption type, salt type, and string-to-key parameters from the KDC response. If the KDC provides no etype-info, set *enctype_out* to **ENCTYPE_NULL** and set *salt_out* and *s2kparams_out* to empty. If the KDC etype-info provides no salt, compute the default salt and place it in *salt_out* . If the KDC etype-info provides no string-to-key parameters, set *s2kparams_out* to empty.

> *opt* may be used to specify options which affect the initial request, such as request encryption types or a FAST armor cache (see *krb5_get_init_creds_opt_set_etype_list()* and *krb5_get_init_creds_opt_set_fast_ccache_name()*).

Use *krb5_free_data_contents()* to free *salt_out* and *s2kparams_out* when they are no longer needed.

Note: New in 1.17

---

krb5_get_permitted_enctypes - Return a list of encryption types permitted for session keys.

krb5_error_code **krb5_get_permitted_enctypes**(*krb5_context context*, *krb5_enctype ** *ktypes*)

param **[in] context** - Library context

[out] ktypes - Zero-terminated list of encryption types

retval

- 0 Success; otherwise - Kerberos error codes

This function returns the list of encryption types permitted for session keys within *context* , as determined by configuration or by a previous call to *krb5_set_default_tgs_enctypes()* .

Use *krb5_free_enctypes()* to free *ktypes* when it is no longer needed.

krb5_get_server_rcache - Generate a replay cache object for server use and open it.

krb5_error_code **krb5_get_server_rcache** (*krb5_context context*, const *krb5_data* * *piece*, *krb5_rcache* * *rcptr*)

param **[in] context** - Library context

[in] piece - Unused (replay cache identifier)

[out] rcptr - Handle to an open rcache

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a handle to the default replay cache. Use krb5_rc_close() to close *rcptr* when it is no longer needed.

Note: Prior to release 1.18, this function creates a handle to a different replay cache for each unique value of *piece* .

---

krb5_get_time_offsets - Return the time offsets from the os context.

krb5_error_code **krb5_get_time_offsets** (*krb5_context context*, *krb5_timestamp* * *seconds*, *krb5_int32* * *microseconds*)

param **[in] context** - Library context

[out] seconds - Time offset, seconds portion

[out] microseconds - Time offset, microseconds portion

retval

- 0 Success; otherwise - Kerberos error codes

This function returns the time offsets in *context* .

krb5_init_context_profile - Create a krb5 library context using a specified profile.

krb5_error_code **krb5_init_context_profile** (struct _profile_t * *profile*, *krb5_flags* *flags*, *krb5_context* * *context*)

param **[in] profile** - Profile object (NULL to create default profile)

[in] flags - Context initialization flags

[out] context - Library context

Create a context structure, optionally using a specified profile and initialization flags. If *profile* is NULL, the default profile will be created from config files. If *profile* is non-null, a copy of it will be made for the new context; the caller should still clean up its copy. Valid flag values are:

- *KRB5_INIT_CONTEXT_SECURE* Ignore environment variables
- *KRB5_INIT_CONTEXT_KDC* Use KDC configuration if creating profile

krb5_init_creds_free - Free an initial credentials context.

void **krb5_init_creds_free** (*krb5_context context*, *krb5_init_creds_context ctx*)

param **[in] context** - Library context

[in] ctx - Initial credentials context

context must be the same as the one passed to `krb5_init_creds_init()` for this initial credentials context.

krb5_init_creds_get - Acquire credentials using an initial credentials context.

krb5_error_code **krb5_init_creds_get** (*krb5_context context*, *krb5_init_creds_context ctx*)

param **[in] context** - Library context

[in] ctx - Initial credentials context

retval

- 0 Success; otherwise - Kerberos error codes

This function synchronously obtains credentials using a context created by `krb5_init_creds_init()` . On successful return, the credentials can be retrieved with `krb5_init_creds_get_creds()` .

context must be the same as the one passed to `krb5_init_creds_init()` for this initial credentials context.

krb5_init_creds_get_creds - Retrieve acquired credentials from an initial credentials context.

krb5_error_code **krb5_init_creds_get_creds** (*krb5_context context*, *krb5_init_creds_context ctx*, *krb5_creds * creds*)

param **[in] context** - Library context

[in] ctx - Initial credentials context

[out] creds - Acquired credentials

retval

- 0 Success; otherwise - Kerberos error codes

This function copies the acquired initial credentials from *ctx* into *creds* , after the successful completion of `krb5_init_creds_get()` or `krb5_init_creds_step()` . Use `krb5_free_cred_contents()` to free *creds* when it is no longer needed.

krb5_init_creds_get_error - Get the last error from KDC from an initial credentials context.

krb5_error_code **krb5_init_creds_get_error** (*krb5_context context*, *krb5_init_creds_context ctx*, *krb5_error ** error*)

param **[in] context** - Library context

[in] ctx - Initial credentials context

[out] error - Error from KDC, or NULL if none was received

retval

- 0 Success; otherwise - Kerberos error codes

krb5_init_creds_get_times - Retrieve ticket times from an initial credentials context.

krb5_error_code **krb5_init_creds_get_times**(*krb5_context context*, *krb5_init_creds_context ctx*, *krb5_ticket_times * times*)

param **[in] context** - Library context

[in] ctx - Initial credentials context

[out] times - Ticket times for acquired credentials

retval

- 0 Success; otherwise - Kerberos error codes

The initial credentials context must have completed obtaining credentials via either *krb5_init_creds_get()* or *krb5_init_creds_step()* .

krb5_init_creds_init - Create a context for acquiring initial credentials.

krb5_error_code **krb5_init_creds_init**(*krb5_context context*, *krb5_principal client*, *krb5_prompter_fct prompter*, void * *data*, *krb5_deltat start_time*, *krb5_get_init_creds_opt * options*, *krb5_init_creds_context * ctx*)

param **[in] context** - Library context

[in] client - Client principal to get initial creds for

[in] prompter - Prompter callback

[in] data - Prompter callback argument

[in] start_time - Time when credentials become valid (0 for now)

[in] options - Options structure (NULL for default)

[out] ctx - New initial credentials context

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a new context for acquiring initial credentials. Use *krb5_init_creds_free()* to free *ctx* when it is no longer needed.

Any subsequent calls to *krb5_init_creds_step()* , *krb5_init_creds_get()* , or *krb5_init_creds_free()* for this initial credentials context must use the same *context* argument as the one passed to this function.

krb5_init_creds_set_keytab - Specify a keytab to use for acquiring initial credentials.

krb5_error_code **krb5_init_creds_set_keytab**(*krb5_context context*, *krb5_init_creds_context ctx*, *krb5_keytab keytab*)

param **[in] context** - Library context

[in] ctx - Initial credentials context

[in] keytab - Key table handle

retval

- 0 Success; otherwise - Kerberos error codes

This function supplies a keytab containing the client key for an initial credentials request.

krb5_init_creds_set_password - Set a password for acquiring initial credentials.

krb5_error_code **krb5_init_creds_set_password**(*krb5_context context*, *krb5_init_creds_context ctx*, const char * *password*)

param **[in] context** - Library context

[in] ctx - Initial credentials context

[in] password - Password

retval

- 0 Success; otherwise - Kerberos error codes

This function supplies a password to be used to construct the client key for an initial credentials request.

krb5_init_creds_set_service - Specify a service principal for acquiring initial credentials.

krb5_error_code **krb5_init_creds_set_service**(*krb5_context context*, *krb5_init_creds_context ctx*, const char * *service*)

param **[in] context** - Library context

[in] ctx - Initial credentials context

[in] service - Service principal string

retval

- 0 Success; otherwise - Kerberos error codes

This function supplies a service principal string to acquire initial credentials for instead of the default krbtgt service. *service* is parsed as a principal name; any realm part is ignored.

krb5_init_creds_step - Get the next KDC request for acquiring initial credentials.

krb5_error_code **krb5_init_creds_step**(*krb5_context context*, *krb5_init_creds_context ctx*, *krb5_data* * *in*, *krb5_data* * *out*, *krb5_data* * *realm*, unsigned int * *flags*)

param **[in] context** - Library context

[in] ctx - Initial credentials context

[in] in - KDC response (empty on the first call)

[out] out - Next KDC request

[out] realm - Realm for next KDC request

[out] flags - Output flags

retval

- 0 Success; otherwise - Kerberos error codes

This function constructs the next KDC request in an initial credential exchange, allowing the caller to control the transport of KDC requests and replies. On the first call, *in* should be set to an empty buffer; on subsequent calls, it should be set to the KDC's reply to the previous request.

If more requests are needed, *flags* will be set to KRB5_INIT_CREDS_STEP_FLAG_CONTINUE and the next request will be placed in *out* . If no more requests are needed, *flags* will not contain KRB5_INIT_CREDS_STEP_FLAG_CONTINUE and *out* will be empty.

If this function returns **KRB5KRB_ERR_RESPONSE_TOO_BIG** , the caller should transmit the next request using TCP rather than UDP. If this function returns any other error, the initial credential exchange has failed.

context must be the same as the one passed to krb5_init_creds_init() for this initial credentials context.

krb5_init_keyblock - Initialize an empty krb5_keyblock .

krb5_error_code **krb5_init_keyblock** (*krb5_context context*, *krb5_enctype enctype*, size_t *length*, *krb5_keyblock* ** *out*)

param **[in] context** - Library context

[in] enctype - Encryption type

[in] length - Length of keyblock (or 0)

[out] out - New keyblock structure

retval

- 0 Success; otherwise - Kerberos error codes

Initialize a new keyblock and allocate storage for the contents of the key. It is legal to pass in a length of 0, in which case contents are left unallocated. Use krb5_free_keyblock() to free *out* when it is no longer needed.

Note: If *length* is set to 0, contents are left unallocated.

krb5_is_referral_realm - Check for a match with KRB5_REFERRAL_REALM.

krb5_boolean **krb5_is_referral_realm** (const *krb5_data* * *r*)

param **[in] r** - Realm to check

return

- TRUE if r is zero-length, FALSE otherwise

krb5_kt_add_entry - Add a new entry to a key table.

krb5_error_code **krb5_kt_add_entry** (*krb5_context context*, *krb5_keytab id*, *krb5_keytab_entry* * *entry*)

param **[in] context** - Library context

[in] id - Key table handle

[in] entry - Entry to be added

retval

- 0 Success
- ENOMEM Insufficient memory
- KRB5_KT_NOWRITE Key table is not writeable

return

- Kerberos error codes

krb5_kt_end_seq_get - Release a keytab cursor.

krb5_error_code **krb5_kt_end_seq_get**(*krb5_context context*, *krb5_keytab keytab*, *krb5_kt_cursor * cursor*)

param **[in] context** - Library context

[in] keytab - Key table handle

[out] cursor - Cursor

retval

- 0 Success

return

- Kerberos error codes

This function should be called to release the cursor created by `krb5_kt_start_seq_get()` .

krb5_kt_get_entry - Get an entry from a key table.

krb5_error_code **krb5_kt_get_entry**(*krb5_context context*, *krb5_keytab keytab*, *krb5_const_principal principal*, *krb5_kvno vno*, *krb5_enctype enctype*, *krb5_keytab_entry * entry*)

param **[in] context** - Library context

[in] keytab - Key table handle

[in] principal - Principal name

[in] vno - Key version number (0 for highest available)

[in] enctype - Encryption type (0 zero for any enctype)

[out] entry - Returned entry from key table

retval

- 0 Success
- Kerberos error codes on failure

Retrieve an entry from a key table which matches the *keytab* , *principal* , *vno* , and *enctype* . If *vno* is zero, retrieve the highest-numbered kvno matching the other fields. If *enctype* is 0, match any enctype.

Use `krb5_free_keytab_entry_contents()` to free *entry* when it is no longer needed.

Note: If *vno* is zero, the function retrieves the highest-numbered-kvno entry that matches the specified principal.

---

krb5_kt_have_content - Check if a keytab exists and contains entries.

krb5_error_code **krb5_kt_have_content**(*krb5_context context*, *krb5_keytab keytab*)

param **[in] context** - Library context

[in] keytab - Key table handle

retval

- 0 Keytab exists and contains entries
- KRB5_KT_NOTFOUND Keytab does not contain entries

Note: New in 1.11

---

krb5_kt_next_entry - Retrieve the next entry from the key table.

krb5_error_code **krb5_kt_next_entry** (*krb5_context context*, *krb5_keytab keytab*, *krb5_keytab_entry * entry*, *krb5_kt_cursor * cursor*)

param **[in] context** - Library context

[in] keytab - Key table handle

[out] entry - Returned key table entry

[in] cursor - Key table cursor

retval

- 0 Success
- KRB5_KT_END - if the last entry was reached

return

- Kerberos error codes

Return the next sequential entry in *keytab* and advance *cursor* . Callers must release the returned entry with *krb5_kt_free_entry()* .

krb5_kt_read_service_key - Retrieve a service key from a key table.

krb5_error_code **krb5_kt_read_service_key** (*krb5_context context*, *krb5_pointer keyprocarg*, *krb5_principal principal*, *krb5_kvno vno*, *krb5_enctype enctype*, *krb5_keyblock ** key*)

param **[in] context** - Library context

[in] keyprocarg - Name of a key table (NULL to use default name)

[in] principal - Service principal

[in] vno - Key version number (0 for highest available)

[in] enctype - Encryption type (0 for any type)

[out] key - Service key from key table

retval

- 0 Success

return

- Kerberos error code if not found or keyprocarg is invalid.

Open and search the specified key table for the entry identified by *principal* , *enctype* , and *vno* . If no key is found, return an error code.

The default key table is used, unless *keyprocarg* is non-null. *keyprocarg* designates a specific key table.

Use `krb5_free_keyblock()` to free *key* when it is no longer needed.

krb5_kt_remove_entry - Remove an entry from a key table.

krb5_error_code **krb5_kt_remove_entry** (*krb5_context context*, *krb5_keytab id*, *krb5_keytab_entry * entry*)

param **[in] context** - Library context

[in] id - Key table handle

[in] entry - Entry to remove from key table

retval

- 0 Success
- KRB5_KT_NOWRITE Key table is not writable

return

- Kerberos error codes

krb5_kt_start_seq_get - Start a sequential retrieval of key table entries.

krb5_error_code **krb5_kt_start_seq_get** (*krb5_context context*, *krb5_keytab keytab*, *krb5_kt_cursor * cursor*)

param **[in] context** - Library context

[in] keytab - Key table handle

[out] cursor - Cursor

retval

- 0 Success

return

- Kerberos error codes

Prepare to read sequentially every key in the specified key table. Use `krb5_kt_end_seq_get()` to release the cursor when it is no longer needed.

krb5_make_authdata_kdc_issued - Encode and sign AD-KDCIssued authorization data.

krb5_error_code **krb5_make_authdata_kdc_issued** (*krb5_context context*, const *krb5_keyblock * key*, *krb5_const_principal issuer*, *krb5_authdata* *const * *authdata*, *krb5_authdata* *** *ad_kdcissued*)

param **[in] context** - Library context

[in] key - Session key

[in] issuer - The name of the issuing principal

[in] authdata - List of authorization data to be signed

[out] ad_kdcissued - List containing AD-KDCIssued authdata

This function wraps a list of authorization data entries *authdata* in an AD-KDCIssued container (see RFC 4120 section 5.2.6.2) signed with *key* . The result is returned in *ad_kdcissued* as a single-element list.

krb5_merge_authdata - Merge two authorization data lists into a new list.

krb5_error_code **krb5_merge_authdata**(*krb5_context context*, *krb5_authdata* *const * *inauthdat1*, *krb5_authdata* *const * *inauthdat2*, *krb5_authdata* *** *outauthdat*)

param [in] context - Library context

[in] inauthdat1 - First list of *krb5_authdata* structures

[in] inauthdat2 - Second list of *krb5_authdata* structures

[out] outauthdat - Merged list of *krb5_authdata* structures

retval

- 0 Success; otherwise - Kerberos error codes

Merge two authdata arrays, such as the array from a ticket and authenticator. Use *krb5_free_authdata()* to free *outauthdat* when it is no longer needed.

Note: The last array entry in *inauthdat1* and *inauthdat2* must be a NULL pointer.

---

krb5_mk_1cred - Format a KRB-CRED message for a single set of credentials.

krb5_error_code **krb5_mk_1cred**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_creds* * *creds*, *krb5_data* ** *der_out*, *krb5_replay_data* * *rdata_out*)

param [in] context - Library context

[in] auth_context - Authentication context

[in] creds - Pointer to credentials

[out] der_out - Encoded credentials

[out] rdata_out - Replay cache data (NULL if not needed)

retval

- 0 Success
- ENOMEM Insufficient memory
- KRB5_RC_REQUIRED Message replay detection requires rcache parameter

return

- Kerberos error codes

This is a convenience function that calls *krb5_mk_ncred()* with a single set of credentials.

krb5_mk_error - Format and encode a KRB_ERROR message.

krb5_error_code **krb5_mk_error** (*krb5_context context*, const *krb5_error* * *dec_err*, *krb5_data* * *enc_err*)

param [in] context - Library context

[in] dec_err - Error structure to be encoded

[out] enc_err - Encoded error structure

retval

- 0 Success; otherwise - Kerberos error codes

This function creates a **KRB_ERROR** message in *enc_err* . Use *krb5_free_data_contents()* to free *enc_err* when it is no longer needed.

krb5_mk_ncred - Format a KRB-CRED message for an array of credentials.

krb5_error_code **krb5_mk_ncred** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_creds* ** *creds*, *krb5_data* ** *der_out*, *krb5_replay_data* * *rdata_out*)

param [in] context - Library context

[in] auth_context - Authentication context

[in] creds - Null-terminated array of credentials

[out] der_out - Encoded credentials

[out] rdata_out - Replay cache information (NULL if not needed)

retval

- 0 Success
- ENOMEM Insufficient memory
- KRB5_RC_REQUIRED Message replay detection requires rcache parameter

return

- Kerberos error codes

This function takes an array of credentials *creds* and formats a **KRB-CRED** message *der_out* to pass to *krb5_rd_cred()* .

The local and remote addresses in *auth_context* are optional; if either is specified, they are used to form the sender and receiver addresses in the KRB-CRED message.

If the *KRB5_AUTH_CONTEXT_DO_TIME* flag is set in *auth_context* , an entry for the message is entered in an in-memory replay cache to detect if the message is reflected by an attacker. If *KRB5_AUTH_CONTEXT_DO_TIME* is not set, no replay cache is used. If *KRB5_AUTH_CONTEXT_RET_TIME* is set in *auth_context* , the timestamp used for the KRB-CRED message is stored in *rdata_out* .

If either *KRB5_AUTH_CONTEXT_DO_SEQUENCE* or *KRB5_AUTH_CONTEXT_RET_SEQUENCE* is set, the *auth_context* local sequence number is included in the KRB-CRED message and then incremented. If *KRB5_AUTH_CONTEXT_RET_SEQUENCE* is set, the sequence number used is stored in *rdata_out* .

Use *krb5_free_data_contents()* to free *der_out* when it is no longer needed.

The message will be encrypted using the send subkey of *auth_context* if it is present, or the session key otherwise. If neither key is present, the credentials will not be encrypted, and the message should only be sent over a secure channel. No replay cache entry is used in this case.

Note: The *rdata_out* argument is required if the *KRB5_AUTH_CONTEXT_RET_TIME* or *KRB5_AUTH_CONTEXT_RET_SEQUENCE* flag is set in *auth_context* .

krb5_mk_priv - Format a KRB-PRIV message.

krb5_error_code **krb5_mk_priv**(*krb5_context context*, *krb5_auth_context auth_context*, const *krb5_data * userdata*, *krb5_data * der_out*, *krb5_replay_data * rdata_out*)

param [in] context - Library context

[in] auth_context - Authentication context

[in] userdata - User data for **KRB-PRIV** message

[out] der_out - Formatted **KRB-PRIV** message

[out] rdata_out - Replay data (NULL if not needed)

retval

- 0 Success; otherwise - Kerberos error codes

This function is similar to *krb5_mk_safe()* , but the message is encrypted and integrity-protected, not just integrity-protected.

The local address in *auth_context* must be set, and is used to form the sender address used in the KRB-PRIV message. The remote address is optional; if specified, it will be used to form the receiver address used in the message.

If the *KRB5_AUTH_CONTEXT_DO_TIME* flag is set in *auth_context* , a timestamp is included in the KRB-PRIV message, and an entry for the message is entered in an in-memory replay cache to detect if the message is reflected by an attacker. If *KRB5_AUTH_CONTEXT_DO_TIME* is not set, no replay cache is used. If *KRB5_AUTH_CONTEXT_RET_TIME* is set in *auth_context* , a timestamp is included in the KRB-PRIV message and is stored in *rdata_out* .

If either *KRB5_AUTH_CONTEXT_DO_SEQUENCE* or *KRB5_AUTH_CONTEXT_RET_SEQUENCE* is set, the *auth_context* local sequence number is included in the KRB-PRIV message and then incremented. If *KRB5_AUTH_CONTEXT_RET_SEQUENCE* is set, the sequence number used is stored in *rdata_out* .

Use *krb5_free_data_contents()* to free *der_out* when it is no longer needed.

Note: The *rdata_out* argument is required if the *KRB5_AUTH_CONTEXT_RET_TIME* or *KRB5_AUTH_CONTEXT_RET_SEQUENCE* flag is set in *auth_context* .

krb5_mk_rep - Format and encrypt a KRB_AP_REP message.

krb5_error_code **krb5_mk_rep**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_data * outbuf*)

param [in] context - Library context

[in] auth_context - Authentication context

[out] outbuf - **AP-REP** message

retval

- 0 Success; otherwise - Kerberos error codes

This function fills in *outbuf* with an AP-REP message using information from *auth_context* .

If the flags in *auth_context* indicate that a sequence number should be used (either *KRB5_AUTH_CONTEXT_DO_SEQUENCE* or *KRB5_AUTH_CONTEXT_RET_SEQUENCE*) and the local sequence number in *auth_context* is 0, a new number will be generated with krb5_generate_seq_number().

Use *krb5_free_data_contents()* to free *outbuf* when it is no longer needed.

krb5_mk_rep_dce - Format and encrypt a KRB_AP_REP message for DCE RPC.

krb5_error_code **krb5_mk_rep_dce**(*krb5_context context*, *krb5_auth_context auth_context*, *krb5_data * outbuf*)

param [in] context - Library context

[in] auth_context - Authentication context

[out] outbuf - **AP-REP** message

retval

- 0 Success; otherwise - Kerberos error codes

Use *krb5_free_data_contents()* to free *outbuf* when it is no longer needed.

krb5_mk_req - Create a KRB_AP_REQ message.

krb5_error_code **krb5_mk_req**(*krb5_context context*, *krb5_auth_context * auth_context*, *krb5_flags ap_req_options*, const char * *service*, const char * *hostname*, *krb5_data * in_data*, *krb5_ccache ccache*, *krb5_data * outbuf*)

param [in] context - Library context

[inout] auth_context - Pre-existing or newly created auth context

[in] ap_req_options - `AP_OPTS` options

[in] service - Service name, or NULL to use **"host"**

[in] hostname - Host name, or NULL to use local hostname

[in] in_data - Application data to be checksummed in the authenticator, or NULL

[in] ccache - Credential cache used to obtain credentials for the desired service.

[out] outbuf - **AP-REQ** message

retval

- 0 Success; otherwise - Kerberos error codes

This function is similar to *krb5_mk_req_extended()* except that it uses a given *hostname* , *service* , and *ccache* to construct a service principal name and obtain credentials.

Use *krb5_free_data_contents()* to free *outbuf* when it is no longer needed.

krb5_mk_req_extended - Create a KRB_AP_REQ message using supplied credentials.

krb5_error_code **krb5_mk_req_extended**(*krb5_context context*, *krb5_auth_context * auth_context*, *krb5_flags ap_req_options*, *krb5_data * in_data*, *krb5_creds * in_creds*, *krb5_data * outbuf*)

param [in] context - Library context

[inout] auth_context - Pre-existing or newly created auth context

[in] ap_req_options - `AP_OPTS` options

[in] in_data - Application data to be checksummed in the authenticator, or NULL

[in] in_creds - Credentials for the service with valid ticket and key

[out] outbuf - **AP-REQ** message

retval

- 0 Success; otherwise - Kerberos error codes

Valid *ap_req_options* are:

- *AP_OPTS_USE_SESSION_KEY* - Use the session key when creating the request used for user to user authentication.
- *AP_OPTS_MUTUAL_REQUIRED* - Request a mutual authentication packet from the receiver.
- *AP_OPTS_USE_SUBKEY* - Generate a subsession key from the current session key obtained from the credentials.

This function creates a KRB_AP_REQ message using supplied credentials *in_creds* . *auth_context* may point to an existing auth context or to NULL, in which case a new one will be created. If *in_data* is non-null, a checksum of it will be included in the authenticator contained in the KRB_AP_REQ message. Use *krb5_free_data_contents()* to free *outbuf* when it is no longer needed.

On successful return, the authenticator is stored in *auth_context* with the *client* and *checksum* fields nulled out. (This is to prevent pointer-sharing problems; the caller should not need these fields anyway, since the caller supplied them.)

See also:

krb5_mk_req()

krb5_mk_safe - Format a KRB-SAFE message.

krb5_error_code **krb5_mk_safe**(*krb5_context context*, *krb5_auth_context auth_context*, const *krb5_data* * *userdata*, *krb5_data* * *der_out*, *krb5_replay_data* * *rdata_out*)

param [in] context - Library context

[in] auth_context - Authentication context

[in] userdata - User data in the message

[out] der_out - Formatted **KRB-SAFE** buffer

[out] rdata_out - Replay data. Specify NULL if not needed

retval

- 0 Success; otherwise - Kerberos error codes

This function creates an integrity protected **KRB-SAFE** message using data supplied by the application.

Fields in *auth_context* specify the checksum type, the keyblock that can be used to seed the checksum, full addresses (host and port) for the sender and receiver, and `KRB5_AUTH_CONTEXT` flags.

The local address in *auth_context* must be set, and is used to form the sender address used in the KRB-SAFE message. The remote address is optional; if specified, it will be used to form the receiver address used in the message.

If the *KRB5_AUTH_CONTEXT_DO_TIME* flag is set in *auth_context* , a timestamp is included in the KRB-SAFE message, and an entry for the message is entered in an in-memory replay cache to detect if the message is reflected by an attacker. If *KRB5_AUTH_CONTEXT_DO_TIME* is not set, no replay cache is used. If *KRB5_AUTH_CONTEXT_RET_TIME* is set in *auth_context* , a timestamp is included in the KRB-SAFE message and is stored in *rdata_out* .

If either *KRB5_AUTH_CONTEXT_DO_SEQUENCE* or *KRB5_AUTH_CONTEXT_RET_SEQUENCE* is set, the *auth_context* local sequence number is included in the KRB-SAFE message and then incremented. If *KRB5_AUTH_CONTEXT_RET_SEQUENCE* is set, the sequence number used is stored in *rdata_out* .

Use *krb5_free_data_contents()* to free *der_out* when it is no longer needed.

Note: The *rdata_out* argument is required if the *KRB5_AUTH_CONTEXT_RET_TIME* or *KRB5_AUTH_CONTEXT_RET_SEQUENCE* flag is set in *auth_context* .

---

krb5_os_localaddr - Return all interface addresses for this host.

krb5_error_code **krb5_os_localaddr**(*krb5_context context*, *krb5_address *** addr*)

param [in] context - Library context

[out] addr - Array of krb5_address pointers, ending with NULL

retval

- 0 Success; otherwise - Kerberos error codes

Use *krb5_free_addresses()* to free *addr* when it is no longer needed.

krb5_pac_add_buffer - Add a buffer to a PAC handle.

krb5_error_code **krb5_pac_add_buffer**(*krb5_context context*, *krb5_pac pac*, *krb5_ui_4 type*, const *krb5_data * data*)

param [in] context - Library context

[in] pac - PAC handle

[in] type - Buffer type

[in] data - contents

retval

- 0 Success; otherwise - Kerberos error codes

This function adds a buffer of type *type* and contents *data* to *pac* if there isn't already a buffer of this type present.

The valid values of *type* is one of the following:

- *KRB5_PAC_LOGON_INFO* - Logon information
- *KRB5_PAC_CREDENTIALS_INFO* - Credentials information
- *KRB5_PAC_SERVER_CHECKSUM* - Server checksum
- *KRB5_PAC_PRIVSVR_CHECKSUM* - KDC checksum
- *KRB5_PAC_CLIENT_INFO* - Client name and ticket information
- *KRB5_PAC_DELEGATION_INFO* - Constrained delegation information

- *KRB5_PAC_UPN_DNS_INFO* - User principal name and DNS information

krb5_pac_free - Free a PAC handle.

void **krb5_pac_free** (*krb5_context context*, *krb5_pac pac*)

param **[in] context** - Library context

[in] pac - PAC to be freed

This function frees the contents of *pac* and the structure itself.

krb5_pac_get_buffer - Retrieve a buffer value from a PAC.

krb5_error_code **krb5_pac_get_buffer** (*krb5_context context*, *krb5_pac pac*, *krb5_ui_4 type*, *krb5_data * data*)

param **[in] context** - Library context

[in] pac - PAC handle

[in] type - Type of buffer to retrieve

[out] data - Buffer value

retval

- 0 Success; otherwise - Kerberos error codes

Use *krb5_free_data_contents()* to free *data* when it is no longer needed.

krb5_pac_get_types - Return an array of buffer types in a PAC handle.

krb5_error_code **krb5_pac_get_types** (*krb5_context context*, *krb5_pac pac*, size_t * *len*, *krb5_ui_4 ** *types*)

param **[in] context** - Library context

[in] pac - PAC handle

[out] len - Number of entries in *types*

[out] types - Array of buffer types

retval

- 0 Success; otherwise - Kerberos error codes

krb5_pac_init - Create an empty Privilege Attribute Certificate (PAC) handle.

krb5_error_code **krb5_pac_init** (*krb5_context context*, *krb5_pac * pac*)

param **[in] context** - Library context

[out] pac - New PAC handle

retval

- 0 Success; otherwise - Kerberos error codes

Use *krb5_pac_free()* to free *pac* when it is no longer needed.

krb5_pac_parse - Unparse an encoded PAC into a new handle.

krb5_error_code **krb5_pac_parse** (*krb5_context context*, const void * *ptr*, size_t *len*, *krb5_pac* * *pac*)

param **[in] context** - Library context

[in] ptr - PAC buffer

[in] len - Length of *ptr*

[out] pac - PAC handle

retval

- 0 Success; otherwise - Kerberos error codes

Use `krb5_pac_free()` to free *pac* when it is no longer needed.

krb5_pac_sign - Sign a PAC.

krb5_error_code **krb5_pac_sign** (*krb5_context context*, *krb5_pac pac*, *krb5_timestamp authtime*, *krb5_const_principal principal*, const *krb5_keyblock* * *server_key*, const *krb5_keyblock* * *privsvr_key*, *krb5_data* * *data*)

param **[in] context** - Library context

[in] pac - PAC handle

[in] authtime - Expected timestamp

[in] principal - Expected principal name (or NULL)

[in] server_key - Key for server checksum

[in] privsvr_key - Key for KDC checksum

[out] data - Signed PAC encoding

This function signs *pac* using the keys *server_key* and *privsvr_key* and returns the signed encoding in *data* . *pac* is modified to include the server and KDC checksum buffers. Use `krb5_free_data_contents()` to free *data* when it is no longer needed.

Note: New in 1.10

---

krb5_pac_sign_ext - Sign a PAC, possibly with a specified realm.

krb5_error_code **krb5_pac_sign_ext** (*krb5_context context*, *krb5_pac pac*, *krb5_timestamp authtime*, *krb5_const_principal principal*, const *krb5_keyblock* * *server_key*, const *krb5_keyblock* * *privsvr_key*, *krb5_boolean with_realm*, *krb5_data* * *data*)

param **[in] context** - Library context

[in] pac - PAC handle

[in] authtime - Expected timestamp

[in] principal - Principal name (or NULL)

[in] server_key - Key for server checksum

[in] privsvr_key - Key for KDC checksum

[in] with_realm - If true, include the realm of *principal*

[out] data - Signed PAC encoding

This function is similar to `krb5_pac_sign()` , but adds a parameter *with_realm* . If *with_realm* is true, the PAC_CLIENT_INFO field of the signed PAC will include the realm of *principal* as well as the name. This flag is necessary to generate PACs for cross-realm S4U2Self referrals.

Note: New in 1.17

---

krb5_pac_verify - Verify a PAC.

krb5_error_code **krb5_pac_verify** (*krb5_context context*, const *krb5_pac pac*, *krb5_timestamp authtime*, *krb5_const_principal principal*, const *krb5_keyblock* * *server*, const *krb5_keyblock* * *privsvr*)

param [in] context - Library context

[in] pac - PAC handle

[in] authtime - Expected timestamp

[in] principal - Expected principal name (or NULL)

[in] server - Key to validate server checksum (or NULL)

[in] privsvr - Key to validate KDC checksum (or NULL)

retval

- 0 Success; otherwise - Kerberos error codes

This function validates *pac* against the supplied *server* , *privsvr* , *principal* and *authtime* . If *principal* is NULL, the principal and authtime are not verified. If *server* or *privsvr* is NULL, the corresponding checksum is not verified.

If successful, *pac* is marked as verified.

Note: A checksum mismatch can occur if the PAC was copied from a cross-realm TGT by an ignorant KDC; also macOS Server Open Directory (as of 10.6) generates PACs with no server checksum at all. One should consider not failing the whole authentication because of this reason, but, instead, treating the ticket as if it did not contain a PAC or marking the PAC information as non-verified.

---

krb5_pac_verify_ext - Verify a PAC, possibly from a specified realm.

krb5_error_code **krb5_pac_verify_ext** (*krb5_context context*, const *krb5_pac pac*, *krb5_timestamp authtime*, *krb5_const_principal principal*, const *krb5_keyblock* * *server*, const *krb5_keyblock* * *privsvr*, *krb5_boolean with_realm*)

param [in] context - Library context

[in] pac - PAC handle

[in] authtime - Expected timestamp

[in] principal - Expected principal name (or NULL)

[in] server - Key to validate server checksum (or NULL)

[in] privsvr - Key to validate KDC checksum (or NULL)

[in] with_realm - If true, expect the realm of *principal*

This function is similar to `krb5_pac_verify()` , but adds a parameter *with_realm* . If *with_realm* is true, the PAC_CLIENT_INFO field is expected to include the realm of *principal* as well as the name. This flag is necessary to verify PACs in cross-realm S4U2Self referral TGTs.

Note: New in 1.17

---

krb5_pac_get_client_info

krb5_error_code **krb5_pac_get_client_info** (*krb5_context context*, const *krb5_pac pac*, *krb5_timestamp * authtime_out*, char ** *princname_out*)

param context

pac

authtime_out

princname_out

krb5_prepend_error_message - Add a prefix to the message for an error code.

void **krb5_prepend_error_message** (*krb5_context ctx*, *krb5_error_code code*, const char * *fmt*, ...)

param [in] ctx - Library context

[in] code - Error code

[in] fmt - Format string for error message prefix

Format a message and prepend it to the current message for *code* . The prefix will be separated from the old message with a colon and space.

krb5_principal2salt - Convert a principal name into the default salt for that principal.

krb5_error_code **krb5_principal2salt** (*krb5_context context*, *krb5_const_principal pr*, *krb5_data * ret*)

param [in] context - Library context

[in] pr - Principal name

[out] ret - Default salt for *pr* to be filled in

retval

- 0 Success; otherwise - Kerberos error codes

krb5_rd_cred - Read and validate a KRB-CRED message.

krb5_error_code **krb5_rd_cred** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_data * creddata*, *krb5_creds *** *creds_out*, *krb5_replay_data * rdata_out*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] creddata - **KRB-CRED** message

[out] creds_out - Null-terminated array of forwarded credentials

[out] rdata_out - Replay data (NULL if not needed)

retval

- 0 Success; otherwise - Kerberos error codes

creddata will be decrypted using the receiving subkey if it is present in *auth_context* , or the session key if the receiving subkey is not present or fails to decrypt the message.

Use `krb5_free_tgt_creds()` to free *creds_out* when it is no longer needed.

Note: The *rdata_out* argument is required if the `KRB5_AUTH_CONTEXT_RET_TIME` or `KRB5_AUTH_CONTEXT_RET_SEQUENCE` flag is set in *auth_context* .‘

---

krb5_rd_error - Decode a KRB-ERROR message.

krb5_error_code **krb5_rd_error**(*krb5_context context*, const *krb5_data* * *enc_errbuf*, *krb5_error* ** *dec_error*)

param **[in] context** - Library context

[in] enc_errbuf - Encoded error message

[out] dec_error - Decoded error message

retval

- 0 Success; otherwise - Kerberos error codes

This function processes **KRB-ERROR** message *enc_errbuf* and returns an allocated structure *dec_error* containing the error message. Use `krb5_free_error()` to free *dec_error* when it is no longer needed.

krb5_rd_priv - Process a KRB-PRIV message.

krb5_error_code **krb5_rd_priv**(*krb5_context context*, *krb5_auth_context auth_context*, const *krb5_data* * *inbuf*, *krb5_data* * *userdata_out*, *krb5_replay_data* * *rdata_out*)

param **[in] context** - Library context

[in] auth_context - Authentication structure

[in] inbuf - **KRB-PRIV** message to be parsed

[out] userdata_out - Data parsed from **KRB-PRIV** message

[out] rdata_out - Replay data. Specify NULL if not needed

retval

- 0 Success; otherwise - Kerberos error codes

This function parses a **KRB-PRIV** message, verifies its integrity, and stores its unencrypted data into *userdata_out* .

If *auth_context* has a remote address set, the address will be used to verify the sender address in the KRB-PRIV message. If *auth_context* has a local address set, it will be used to verify the receiver address in the KRB-PRIV message if the message contains one.

If the *KRB5_AUTH_CONTEXT_DO_SEQUENCE* flag is set in *auth_context* , the sequence number of the KRB-PRIV message is checked against the remote sequence number field of *auth_context* . Otherwise, the sequence number is not used.

If the *KRB5_AUTH_CONTEXT_DO_TIME* flag is set in *auth_context* , then the timestamp in the message is verified to be within the permitted clock skew of the current time, and the message is checked against an in-memory replay cache to detect reflections or replays.

Use *krb5_free_data_contents()* to free *userdata_out* when it is no longer needed.

Note: The *rdata_out* argument is required if the *KRB5_AUTH_CONTEXT_RET_TIME* or *KRB5_AUTH_CONTEXT_RET_SEQUENCE* flag is set in *auth_context* .

---

krb5_rd_rep - Parse and decrypt a KRB_AP_REP message.

krb5_error_code **krb5_rd_rep** (*krb5_context context*, *krb5_auth_context auth_context*, const *krb5_data * inbuf*, *krb5_ap_rep_enc_part ** repl*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] inbuf - AP-REP message

[out] repl - Decrypted reply message

retval

- 0 Success; otherwise - Kerberos error codes

This function parses, decrypts and verifies a message from *inbuf* and fills in *repl* with a pointer to allocated memory containing the fields from the encrypted response.

Use *krb5_free_ap_rep_enc_part()* to free *repl* when it is no longer needed.

krb5_rd_rep_dce - Parse and decrypt a KRB_AP_REP message for DCE RPC.

krb5_error_code **krb5_rd_rep_dce** (*krb5_context context*, *krb5_auth_context auth_context*, const *krb5_data * inbuf*, *krb5_ui_4 * nonce*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] inbuf - AP-REP message

[out] nonce - Sequence number from the decrypted reply

retval

- 0 Success; otherwise - Kerberos error codes

This function parses, decrypts and verifies a message from *inbuf* and fills in *nonce* with a decrypted reply sequence number.

krb5_rd_req - Parse and decrypt a KRB_AP_REQ message.

krb5_error_code **krb5_rd_req**(*krb5_context context*, *krb5_auth_context* * *auth_context*, const *krb5_data* * *inbuf*, *krb5_const_principal server*, *krb5_keytab keytab*, *krb5_flags* * *ap_req_options*, *krb5_ticket* ** *ticket*)

param [in] context - Library context

[inout] auth_context - Pre-existing or newly created auth context

[in] inbuf - AP-REQ message to be parsed

[in] server - Matching principal for server, or NULL to allow any principal in keytab

[in] keytab - Key table, or NULL to use the default

[out] ap_req_options - If non-null, the AP-REQ flags on output

[out] ticket - If non-null, ticket from the AP-REQ message

retval

- 0 Success; otherwise - Kerberos error codes

This function parses, decrypts and verifies a AP-REQ message from *inbuf* and stores the authenticator in *auth_context* .

If a keyblock was specified in *auth_context* using `krb5_auth_con_setuseruserkey()` , that key is used to decrypt the ticket in AP-REQ message and *keytab* is ignored. In this case, *server* should be specified as a complete principal name to allow for proper transited-path checking and replay cache selection.

Otherwise, the decryption key is obtained from *keytab* , or from the default keytab if it is NULL. In this case, *server* may be a complete principal name, a matching principal (see `krb5_sname_match()`), or NULL to match any principal name. The keys tried against the encrypted part of the ticket are determined as follows:

- If *server* is a complete principal name, then its entry in *keytab* is tried.
- Otherwise, if *keytab* is iterable, then all entries in *keytab* which match *server* are tried.
- Otherwise, the server principal in the ticket must match *server* , and its entry in *keytab* is tried.

The client specified in the decrypted authenticator must match the client specified in the decrypted ticket.

If the *remote_addr* field of *auth_context* is set, the request must come from that address.

If a replay cache handle is provided in the *auth_context* , the authenticator and ticket are verified against it. If no conflict is found, the new authenticator is then stored in the replay cache of *auth_context* .

Various other checks are performed on the decoded data, including cross-realm policy, clockskew, and ticket validation times.

On success the authenticator, subkey, and remote sequence number of the request are stored in *auth_context* . If the `AP_OPTS_MUTUAL_REQUIRED` bit is set, the local sequence number is XORed with the remote sequence number in the request.

Use `krb5_free_ticket()` to free *ticket* when it is no longer needed.

krb5_rd_safe - Process KRB-SAFE message.

krb5_error_code **krb5_rd_safe**(*krb5_context context*, *krb5_auth_context auth_context*, const *krb5_data* * *inbuf*, *krb5_data* * *userdata_out*, *krb5_replay_data* * *rdata_out*)

param **[in] context** - Library context

[in] auth_context - Authentication context

[in] inbuf - **KRB-SAFE** message to be parsed

[out] userdata_out - Data parsed from **KRB-SAFE** message

[out] rdata_out - Replay data. Specify NULL if not needed

retval

- 0 Success; otherwise - Kerberos error codes

This function parses a **KRB-SAFE** message, verifies its integrity, and stores its data into *userdata_out* .

If *auth_context* has a remote address set, the address will be used to verify the sender address in the KRB-SAFE message. If *auth_context* has a local address set, it will be used to verify the receiver address in the KRB-SAFE message if the message contains one.

If the *KRB5_AUTH_CONTEXT_DO_SEQUENCE* flag is set in *auth_context* , the sequence number of the KRB-SAFE message is checked against the remote sequence number field of *auth_context* . Otherwise, the sequence number is not used.

If the *KRB5_AUTH_CONTEXT_DO_TIME* flag is set in *auth_context* , then the timestamp in the message is verified to be within the permitted clock skew of the current time, and the message is checked against an in-memory replay cache to detect reflections or replays.

Use *krb5_free_data_contents()* to free *userdata_out* when it is no longer needed.

Note: The *rdata_out* argument is required if the *KRB5_AUTH_CONTEXT_RET_TIME* or *KRB5_AUTH_CONTEXT_RET_SEQUENCE* flag is set in *auth_context* .

---

krb5_read_password - Read a password from keyboard input.

krb5_error_code **krb5_read_password**(*krb5_context context*, const char * *prompt*, const char * *prompt2*, char * *return_pwd*, unsigned int * *size_return*)

param **[in] context** - Library context

[in] prompt - First user prompt when reading password

[in] prompt2 - Second user prompt (NULL to prompt only once)

[out] return_pwd - Returned password

[inout] size_return - On input, maximum size of password; on output, size of password read

retval

- 0 Success

return

- Error in reading or verifying the password
- Kerberos error codes

This function reads a password from keyboard input and stores it in *return_pwd* . *size_return* should be set by the caller to the amount of storage space available in *return_pwd* ; on successful return, it will be set to the length of the password read.

prompt is printed to the terminal, followed by":", and then a password is read from the keyboard.

If *prompt2* is NULL, the password is read only once. Otherwise, *prompt2* is printed to the terminal and a second password is read. If the two passwords entered are not identical, KRB5_LIBOS_BADPWDMATCH is returned.

Echoing is turned off when the password is read.

krb5_salttype_to_string - Convert a salt type to a string.

krb5_error_code **krb5_salttype_to_string** (*krb5_int32 salttype*, char * *buffer*, size_t *buflen*)

param **[in] salttype** - Salttype to convert

[out] buffer - Buffer to receive the converted string

[in] buflen - Storage available in *buffer*

retval

- 0 Success; otherwise - Kerberos error codes

krb5_server_decrypt_ticket_keytab - Decrypt a ticket using the specified key table.

krb5_error_code **krb5_server_decrypt_ticket_keytab** (*krb5_context context*, const *krb5_keytab kt*, *krb5_ticket* * *ticket*)

param **[in] context** - Library context

[in] kt - Key table

[in] ticket - Ticket to be decrypted

retval

- 0 Success; otherwise - Kerberos error codes

This function takes a *ticket* as input and decrypts it using key data from *kt* . The result is placed into *ticket->enc_part2* .

krb5_set_default_tgs_enctypes - Set default TGS encryption types in a krb5_context structure.

krb5_error_code **krb5_set_default_tgs_enctypes** (*krb5_context context*, const *krb5_enctype* * *etypes*)

param **[in] context** - Library context

[in] etypes - Encryption type(s) to set

retval

- 0 Success
- KRB5_PROG_ETYPE_NOSUPP Program lacks support for encryption type

return

- Kerberos error codes

This function sets the default enctype list for TGS requests made using *context* to *etypes* .

Note: This overrides the default list (from config file or built-in).

krb5_set_error_message - Set an extended error message for an error code.

void **krb5_set_error_message** (*krb5_context ctx*, *krb5_error_code code*, const char * *fmt*, ...)

param **[in] ctx** - Library context

[in] code - Error code

[in] fmt - Error string for the error code

krb5_set_kdc_recv_hook - Set a KDC post-receive hook function.

void **krb5_set_kdc_recv_hook** (*krb5_context context*, *krb5_post_recv_fn recv_hook*, void * *data*)

param **[in] context** - The library context.

[in] recv_hook - Hook function (or NULL to disable the hook)

[in] data - Callback data to be passed to *recv_hook*

recv_hook will be called after a reply is received from a KDC during a call to a library function such as `krb5_get_credentials()` . The hook function may inspect or override the reply. This hook will not be executed if the pre-send hook returns a synthetic reply.

Note: New in 1.15

krb5_set_kdc_send_hook - Set a KDC pre-send hook function.

void **krb5_set_kdc_send_hook** (*krb5_context context*, *krb5_pre_send_fn send_hook*, void * *data*)

param **[in] context** - Library context

[in] send_hook - Hook function (or NULL to disable the hook)

[in] data - Callback data to be passed to *send_hook*

send_hook will be called before messages are sent to KDCs by library functions such as `krb5_get_credentials()` . The hook function may inspect, override, or synthesize its own reply to the message.

Note: New in 1.15

krb5_set_real_time - Set time offset field in a krb5_context structure.

krb5_error_code **krb5_set_real_time** (*krb5_context context*, *krb5_timestamp seconds*, *krb5_int32 microseconds*)

param **[in] context** - Library context

[in] seconds - Real time, seconds portion

[in] microseconds - Real time, microseconds portion

retval

- 0 Success; otherwise - Kerberos error codes

This function sets the time offset in *context* to the difference between the system time and the real time as determined by *seconds* and *microseconds* .

krb5_string_to_cksumtype - Convert a string to a checksum type.

krb5_error_code **krb5_string_to_cksumtype** (char * *string*, *krb5_cksumtype* * *cksumtypep*)

param [in] string - String to be converted

[out] cksumtypep - Checksum type to be filled in

retval

- 0 Success; otherwise - EINVAL

krb5_string_to_deltat - Convert a string to a delta time value.

krb5_error_code **krb5_string_to_deltat** (char * *string*, *krb5_deltat* * *deltatp*)

param [in] string - String to be converted

[out] deltatp - Delta time to be filled in

retval

- 0 Success; otherwise - KRB5_DELTAT_BADFORMAT

krb5_string_to_enctype - Convert a string to an encryption type.

krb5_error_code **krb5_string_to_enctype** (char * *string*, *krb5_enctype* * *enctypep*)

param [in] string - String to convert to an encryption type

[out] enctypep - Encryption type

retval

- 0 Success; otherwise - EINVAL

krb5_string_to_salttype - Convert a string to a salt type.

krb5_error_code **krb5_string_to_salttype** (char * *string*, *krb5_int32* * *salttypep*)

param [in] string - String to convert to an encryption type

[out] salttypep - Salt type to be filled in

retval

- 0 Success; otherwise - EINVAL

krb5_string_to_timestamp - Convert a string to a timestamp.

krb5_error_code **krb5_string_to_timestamp** (char * *string*, *krb5_timestamp* * *timestampp*)

param [in] string - String to be converted

[out] timestampp - Pointer to timestamp

retval

- 0 Success; otherwise - EINVAL

krb5_timeofday - Retrieve the current time with context specific time offset adjustment.

krb5_error_code **krb5_timeofday** (*krb5_context context*, *krb5_timestamp * timeret*)

param [in] context - Library context

[out] timeret - Timestamp to fill in

retval

- 0 Success

return

- Kerberos error codes

This function retrieves the system time of day with the context specific time offset adjustment.

krb5_timestamp_to_sfstring - Convert a timestamp to a string, with optional output padding.

krb5_error_code **krb5_timestamp_to_sfstring** (*krb5_timestamp timestamp*, char * *buffer*, size_t *buflen*, char * *pad*)

param [in] timestamp - Timestamp to convert

[out] buffer - Buffer to hold the converted timestamp

[in] buflen - Length of buffer

[in] pad - Optional value to pad *buffer* if converted timestamp does not fill it

retval

- 0 Success; otherwise - Kerberos error codes

If *pad* is not NULL, *buffer* is padded out to *buflen* - 1 characters with the value of * *pad* .

krb5_timestamp_to_string - Convert a timestamp to a string.

krb5_error_code **krb5_timestamp_to_string** (*krb5_timestamp timestamp*, char * *buffer*, size_t *buflen*)

param [in] timestamp - Timestamp to convert

[out] buffer - Buffer to hold converted timestamp

[in] buflen - Storage available in *buffer*

retval

- 0 Success; otherwise - Kerberos error codes

The string is returned in the locale's appropriate date and time representation.

krb5_tkt_creds_free - Free a TGS request context.

void **krb5_tkt_creds_free** (*krb5_context context*, *krb5_tkt_creds_context ctx*)

param [in] context - Library context

[in] ctx - TGS request context

Note: New in 1.9

krb5_tkt_creds_get - Synchronously obtain credentials using a TGS request context.

krb5_error_code **krb5_tkt_creds_get** (*krb5_context context*, *krb5_tkt_creds_context ctx*)

param [in] context - Library context

[in] ctx - TGS request context

retval

- 0 Success; otherwise - Kerberos error codes

This function synchronously obtains credentials using a context created by `krb5_tkt_creds_init()` . On successful return, the credentials can be retrieved with `krb5_tkt_creds_get_creds()` .

Note: New in 1.9

krb5_tkt_creds_get_creds - Retrieve acquired credentials from a TGS request context.

krb5_error_code **krb5_tkt_creds_get_creds** (*krb5_context context*, *krb5_tkt_creds_context ctx*, *krb5_creds * creds*)

param [in] context - Library context

[in] ctx - TGS request context

[out] creds - Acquired credentials

retval

- 0 Success; otherwise - Kerberos error codes

This function copies the acquired initial credentials from *ctx* into *creds* , after the successful completion of `krb5_tkt_creds_get()` or `krb5_tkt_creds_step()` . Use `krb5_free_cred_contents()` to free *creds* when it is no longer needed.

Note: New in 1.9

krb5_tkt_creds_get_times - Retrieve ticket times from a TGS request context.

krb5_error_code **krb5_tkt_creds_get_times** (*krb5_context context*, *krb5_tkt_creds_context ctx*, *krb5_ticket_times * times*)

param [in] context - Library context

[in] ctx - TGS request context

[out] times - Ticket times for acquired credentials

retval

- 0 Success; otherwise - Kerberos error codes

The TGS request context must have completed obtaining credentials via either *krb5_tkt_creds_get()* or *krb5_tkt_creds_step()* .

Note: New in 1.9

krb5_tkt_creds_init - Create a context to get credentials from a KDC's Ticket Granting Service.

krb5_error_code **krb5_tkt_creds_init** (*krb5_context context*, *krb5_ccache ccache*, *krb5_creds * creds*, *krb5_flags options*, *krb5_tkt_creds_context * ctx*)

param [in] context - Library context

[in] ccache - Credential cache handle

[in] creds - Input credentials

[in] options - KRB5_GC options for this request.

[out] ctx - New TGS request context

retval

- 0 Success; otherwise - Kerberos error codes

This function prepares to obtain credentials matching *creds* , either by retrieving them from *ccache* or by making requests to ticket-granting services beginning with a ticket-granting ticket for the client principal's realm.

The resulting TGS acquisition context can be used asynchronously with *krb5_tkt_creds_step()* or synchronously with *krb5_tkt_creds_get()* . See also *krb5_get_credentials()* for synchronous use.

Use *krb5_tkt_creds_free()* to free *ctx* when it is no longer needed.

Note: New in 1.9

krb5_tkt_creds_step - Get the next KDC request in a TGS exchange.

krb5_error_code **krb5_tkt_creds_step** (*krb5_context context*, *krb5_tkt_creds_context ctx*, *krb5_data * in*, *krb5_data * out*, *krb5_data * realm*, unsigned int * *flags*)

param [in] context - Library context

[in] ctx - TGS request context

[in] in - KDC response (empty on the first call)

[out] out - Next KDC request

[out] realm - Realm for next KDC request

[out] flags - Output flags

retval

- 0 Success; otherwise - Kerberos error codes

This function constructs the next KDC request for a TGS exchange, allowing the caller to control the transport of KDC requests and replies. On the first call, *in* should be set to an empty buffer; on subsequent calls, it should be set to the KDC's reply to the previous request.

If more requests are needed, *flags* will be set to *KRB5_TKT_CREDS_STEP_FLAG_CONTINUE* and the next request will be placed in *out* . If no more requests are needed, *flags* will not contain *KRB5_TKT_CREDS_STEP_FLAG_CONTINUE* and *out* will be empty.

If this function returns **KRB5KRB_ERR_RESPONSE_TOO_BIG** , the caller should transmit the next request using TCP rather than UDP. If this function returns any other error, the TGS exchange has failed.

Note: New in 1.9

krb5_verify_init_creds - Verify initial credentials against a keytab.

krb5_error_code **krb5_verify_init_creds**(*krb5_context context*, *krb5_creds * creds*, *krb5_principal server*, *krb5_keytab keytab*, *krb5_ccache * ccache*, *krb5_verify_init_creds_opt * options*)

param **[in] context** - Library context

[in] creds - Initial credentials to be verified

[in] server - Server principal (or NULL)

[in] keytab - Key table (NULL to use default keytab)

[in] ccache - Credential cache for fetched creds (or NULL)

[in] options - Verification options (NULL for default options)

retval

- 0 Success; otherwise - Kerberos error codes

This function attempts to verify that *creds* were obtained from a KDC with knowledge of a key in *keytab* , or the default keytab if *keytab* is NULL. If *server* is provided, the highest-kvno key entry for that principal name is used to verify the credentials; otherwise, all unique"host"service principals in the keytab are tried.

If the specified keytab does not exist, or is empty, or cannot be read, or does not contain an entry for *server* , then credential verification may be skipped unless configuration demands that it succeed. The caller can control this behavior by providing a verification options structure; see *krb5_verify_init_creds_opt_init()* and *krb5_verify_init_creds_opt_set_ap_req_nofail()* .

If *ccache* is NULL, any additional credentials fetched during the verification process will be destroyed. If *ccache* points to NULL, a memory ccache will be created for the additional credentials and returned in *ccache* . If *ccache* points to a valid credential cache handle, the additional credentials will be stored in that cache.

krb5_verify_init_creds_opt_init - Initialize a credential verification options structure.

void **krb5_verify_init_creds_opt_init**(*krb5_verify_init_creds_opt * k5_vic_options*)

param **[in] k5_vic_options** - Verification options structure

krb5_verify_init_creds_opt_set_ap_req_nofail - Set whether credential verification is required.

void **krb5_verify_init_creds_opt_set_ap_req_nofail**(*krb5_verify_init_creds_opt * k5_vic_options*, int *ap_req_nofail*)

param **[in] k5_vic_options** - Verification options structure

[in] ap_req_nofail - Whether to require successful verification

This function determines how *krb5_verify_init_creds()* behaves if no keytab information is available. If *ap_req_nofail* is **FALSE** , verification will be skipped in this case and *krb5_verify_init_creds()* will return successfully. If *ap_req_nofail* is **TRUE** , *krb5_verify_init_creds()* will not return successfully unless verification can be performed.

If this function is not used, the behavior of *krb5_verify_init_creds()* is determined through configuration.

krb5_vprepend_error_message - Add a prefix to the message for an error code using a va_list.

void **krb5_vprepend_error_message**(*krb5_context ctx*, *krb5_error_code code*, const char * *fmt*, va_list *args*)

param **[in] ctx** - Library context

[in] code - Error code

[in] fmt - Format string for error message prefix

[in] args - List of vprintf(3) style arguments

This function is similar to *krb5_prepend_error_message()* , but uses a va_list instead of variadic arguments.

krb5_vset_error_message - Set an extended error message for an error code using a va_list.

void **krb5_vset_error_message**(*krb5_context ctx*, *krb5_error_code code*, const char * *fmt*, va_list *args*)

param **[in] ctx** - Library context

[in] code - Error code

[in] fmt - Error string for the error code

[in] args - List of vprintf(3) style arguments

krb5_vwrap_error_message - Add a prefix to a different error code's message using a va_list.

void **krb5_vwrap_error_message**(*krb5_context ctx*, *krb5_error_code old_code*, *krb5_error_code code*, const char * *fmt*, va_list *args*)

param **[in] ctx** - Library context

[in] old_code - Previous error code

[in] code - Error code

[in] fmt - Format string for error message prefix

[in] args - List of vprintf(3) style arguments

This function is similar to *krb5_wrap_error_message()* , but uses a va_list instead of variadic arguments.

krb5_wrap_error_message - Add a prefix to a different error code's message.

void **krb5_wrap_error_message**(*krb5_context ctx*, *krb5_error_code old_code*, *krb5_error_code code*, const char * *fmt*, ...)

param **[in] ctx** - Library context

[in] old_code - Previous error code

[in] code - Error code

[in] fmt - Format string for error message prefix

Format a message and prepend it to the message for *old_code* . The prefix will be separated from the old message with a colon and space. Set the resulting message as the extended error message for *code* .

6.1.3 Public interfaces that should not be called directly

krb5_c_block_size - Return cipher block size.

krb5_error_code **krb5_c_block_size** (*krb5_context context*, *krb5_enctype enctype*, size_t * *blocksize*)

param **[in] context** - Library context

[in] enctype - Encryption type

[out] blocksize - Block size for *enctype*

retval

- 0 Success; otherwise - Kerberos error codes

krb5_c_checksum_length - Return the length of checksums for a checksum type.

krb5_error_code **krb5_c_checksum_length** (*krb5_context context*, *krb5_cksumtype cksumtype*, size_t * *length*)

param **[in] context** - Library context

[in] cksumtype - Checksum type

[out] length - Checksum length

retval

- 0 Success; otherwise - Kerberos error codes

krb5_c_crypto_length - Return a length of a message field specific to the encryption type.

krb5_error_code **krb5_c_crypto_length** (*krb5_context context*, *krb5_enctype enctype*, *krb5_cryptotype type*, unsigned int * *size*)

param **[in] context** - Library context

[in] enctype - Encryption type

[in] type - Type field (See `KRB5_CRYPTO_TYPE` types)

[out] size - Length of the *type* specific to *enctype*

retval

- 0 Success; otherwise - Kerberos error codes

krb5_c_crypto_length_iov - Fill in lengths for header, trailer and padding in a IOV array.

krb5_error_code **krb5_c_crypto_length_iov**(*krb5_context context*, *krb5_enctype enctype*, *krb5_crypto_iov* * *data*, size_t *num_data*)

param **[in] context** - Library context

[in] enctype - Encryption type

[inout] data - IOV array

[in] num_data - Size of *data*

retval

- 0 Success; otherwise - Kerberos error codes

Padding is set to the actual padding required based on the provided *data* buffers. Typically this API is used after setting up the data buffers and *KRB5_CRYPTO_TYPE_SIGN_ONLY* buffers, but before actually allocating header, trailer and padding.

krb5_c_decrypt - Decrypt data using a key (operates on keyblock).

krb5_error_code **krb5_c_decrypt**(*krb5_context context*, const *krb5_keyblock* * *key*, *krb5_keyusage usage*, const *krb5_data* * *cipher_state*, const *krb5_enc_data* * *input*, *krb5_data* * *output*)

param **[in] context** - Library context

[in] key - Encryption key

[in] usage - Key usage (see KRB5_KEYUSAGE types)

[inout] cipher_state - Cipher state; specify NULL if not needed

[in] input - Encrypted data

[out] output - Decrypted data

retval

- 0 Success; otherwise - Kerberos error codes

This function decrypts the data block *input* and stores the output into *output* . The actual decryption key will be derived from *key* and *usage* if key derivation is specified for the encryption type. If non-null, *cipher_state* specifies the beginning state for the decryption operation, and is updated with the state to be passed as input to the next operation.

Note: The caller must initialize *output* and allocate at least enough space for the result. The usual practice is to allocate an output buffer as long as the ciphertext, and let *krb5_c_decrypt()* trim *output->length* . For some enctypes, the resulting *output->length* may include padding bytes.

---

krb5_c_decrypt_iov - Decrypt data in place supporting AEAD (operates on keyblock).

krb5_error_code **krb5_c_decrypt_iov**(*krb5_context context*, const *krb5_keyblock* * *keyblock*, *krb5_keyusage usage*, const *krb5_data* * *cipher_state*, *krb5_crypto_iov* * *data*, size_t *num_data*)

param **[in] context** - Library context

[in] keyblock - Encryption key

[in] usage - Key usage (see KRB5_KEYUSAGE types)

[in] cipher_state - Cipher state; specify NULL if not needed

[inout] data - IOV array. Modified in-place.

[in] num_data - Size of *data*

retval

- 0 Success; otherwise - Kerberos error codes

This function decrypts the data block *data* and stores the output in-place. The actual decryption key will be derived from *keyblock* and *usage* if key derivation is specified for the encryption type. If non-null, *cipher_state* specifies the beginning state for the decryption operation, and is updated with the state to be passed as input to the next operation. The caller must allocate the right number of krb5_crypto_iov structures before calling into this API.

See also:

krb5_c_decrypt_iov()

Note: On return from a *krb5_c_decrypt_iov()* call, the *data->length* in the iov structure are adjusted to reflect actual lengths of the ciphertext used. For example, if the padding length is too large, the length will be reduced. Lengths are never increased.

---

krb5_c_derive_prfplus - Derive a key using some input data (via RFC 6113 PRF+).

krb5_error_code **krb5_c_derive_prfplus** (*krb5_context context*, const *krb5_keyblock * k*, const *krb5_data * input*, *krb5_enctype enctype*, *krb5_keyblock ** *out*)

param **[in] context** - Library context

[in] k - KDC contribution key

[in] input - Input string

[in] enctype - Output key enctype (or **ENCTYPE_NULL**)

[out] out - Derived keyblock

This function uses PRF+ as defined in RFC 6113 to derive a key from another key and an input string. If *enctype* is **ENCTYPE_NULL** , the output key will have the same enctype as the input key.

krb5_c_encrypt - Encrypt data using a key (operates on keyblock).

krb5_error_code **krb5_c_encrypt** (*krb5_context context*, const *krb5_keyblock * key*, *krb5_keyusage usage*, const *krb5_data * cipher_state*, const *krb5_data * input*, *krb5_enc_data * output*)

param **[in] context** - Library context

[in] key - Encryption key

[in] usage - Key usage (see KRB5_KEYUSAGE types)

[inout] cipher_state - Cipher state; specify NULL if not needed

[in] input - Data to be encrypted

[out] output - Encrypted data

retval

- 0 Success; otherwise - Kerberos error codes

This function encrypts the data block *input* and stores the output into *output* . The actual encryption key will be derived from *key* and *usage* if key derivation is specified for the encryption type. If non-null, *cipher_state* specifies the beginning state for the encryption operation, and is updated with the state to be passed as input to the next operation.

Note: The caller must initialize *output* and allocate at least enough space for the result (using *krb5_c_encrypt_length()* to determine the amount of space needed). *output->length* will be set to the actual length of the ciphertext.

---

krb5_c_encrypt_iov - Encrypt data in place supporting AEAD (operates on keyblock).

krb5_error_code **krb5_c_encrypt_iov**(*krb5_context context*, const *krb5_keyblock* * *keyblock*, *krb5_keyusage usage*, const *krb5_data* * *cipher_state*, *krb5_crypto_iov* * *data*, size_t *num_data*)

param [in] context - Library context

[in] keyblock - Encryption key

[in] usage - Key usage (see KRB5_KEYUSAGE types)

[in] cipher_state - Cipher state; specify NULL if not needed

[inout] data - IOV array. Modified in-place.

[in] num_data - Size of *data*

retval

- 0 Success; otherwise - Kerberos error codes

This function encrypts the data block *data* and stores the output in-place. The actual encryption key will be derived from *keyblock* and *usage* if key derivation is specified for the encryption type. If non-null, *cipher_state* specifies the beginning state for the encryption operation, and is updated with the state to be passed as input to the next operation. The caller must allocate the right number of krb5_crypto_iov structures before calling into this API.

See also:

krb5_c_decrypt_iov()

Note: On return from a *krb5_c_encrypt_iov()* call, the *data->length* in the iov structure are adjusted to reflect actual lengths of the ciphertext used. For example, if the padding length is too large, the length will be reduced. Lengths are never increased.

---

krb5_c_encrypt_length - Compute encrypted data length.

krb5_error_code **krb5_c_encrypt_length**(*krb5_context context*, *krb5_enctype enctype*, size_t *inputlen*, size_t * *length*)

param **[in] context** - Library context

[in] enctype - Encryption type

[in] inputlen - Length of the data to be encrypted

[out] length - Length of the encrypted data

retval

- 0 Success; otherwise - Kerberos error codes

This function computes the length of the ciphertext produced by encrypting *inputlen* bytes including padding, confounder, and checksum.

krb5_c_enctype_compare - Compare two encryption types.

krb5_error_code **krb5_c_enctype_compare**(*krb5_context context*, *krb5_enctype e1*, *krb5_enctype e2*, *krb5_boolean * similar*)

param **[in] context** - Library context

[in] e1 - First encryption type

[in] e2 - Second encryption type

[out] similar - **TRUE** if types are similar, **FALSE** if not

retval

- 0 Success; otherwise - Kerberos error codes

This function determines whether two encryption types use the same kind of keys.

krb5_c_free_state - Free a cipher state previously allocated by krb5_c_init_state() .

krb5_error_code **krb5_c_free_state**(*krb5_context context*, const *krb5_keyblock * key*, *krb5_data * state*)

param **[in] context** - Library context

[in] key - Key

[in] state - Cipher state to be freed

retval

- 0 Success; otherwise - Kerberos error codes

krb5_c_fx_cf2_simple - Compute the KRB-FX-CF2 combination of two keys and pepper strings.

krb5_error_code **krb5_c_fx_cf2_simple**(*krb5_context context*, const *krb5_keyblock * k1*, const char * *pepper1*, const *krb5_keyblock * k2*, const char * *pepper2*, *krb5_keyblock ** *out*)

param **[in] context** - Library context

[in] k1 - KDC contribution key

[in] pepper1 - String”PKINIT”

[in] k2 - Reply key

[in] pepper2 - String”KeyExchange”

[out] out - Output key

retval

- 0 Success; otherwise - Kerberos error codes

This function computes the KRB-FX-CF2 function over its inputs and places the results in a newly allocated keyblock. This function is simple in that it assumes that *pepper1* and *pepper2* are C strings with no internal nulls and that the enctype of the result will be the same as that of *k1* . *k1* and *k2* may be of different enctypes.

krb5_c_init_state - Initialize a new cipher state.

krb5_error_code **krb5_c_init_state**(*krb5_context context*, const *krb5_keyblock* * *key*, *krb5_keyusage usage*, *krb5_data* * *new_state*)

param **[in] context** - Library context

[in] key - Key

[in] usage - Key usage (see `KRB5_KEYUSAGE` types)

[out] new_state - New cipher state

retval

- 0 Success; otherwise - Kerberos error codes

krb5_c_is_coll_proof_cksum - Test whether a checksum type is collision-proof.

krb5_boolean **krb5_c_is_coll_proof_cksum**(*krb5_cksumtype ctype*)

param **[in] ctype** - Checksum type

return

- TRUE if ctype is collision-proof, FALSE if it is not collision-proof or not a valid checksum type.

krb5_c_is_keyed_cksum - Test whether a checksum type is keyed.

krb5_boolean **krb5_c_is_keyed_cksum**(*krb5_cksumtype ctype*)

param **[in] ctype** - Checksum type

return

- TRUE if ctype is a keyed checksum type, FALSE otherwise.

krb5_c_keyed_checksum_types - Return a list of keyed checksum types usable with an encryption type.

krb5_error_code **krb5_c_keyed_checksum_types**(*krb5_context context*, *krb5_enctype enctype*, unsigned int * *count*, *krb5_cksumtype* ** *cksumtypes*)

param **[in] context** - Library context

[in] enctype - Encryption type

[out] count - Count of allowable checksum types

[out] cksumtypes - Array of allowable checksum types

retval

- 0 Success; otherwise - Kerberos error codes

Use *krb5_free_cksumtypes()* to free *cksumtypes* when it is no longer needed.

krb5_c_keylengths - Return length of the specified key in bytes.

krb5_error_code **krb5_c_keylengths**(*krb5_context context*, *krb5_enctype enctype*, size_t * *keybytes*, size_t * *keylength*)

param [in] context - Library context

[in] enctype - Encryption type

[out] keybytes - Number of bytes required to make a key

[out] keylength - Length of final key

retval

- 0 Success; otherwise - Kerberos error codes

krb5_c_make_checksum - Compute a checksum (operates on keyblock).

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*)

param [in] context - Library context

[in] cksumtype - Checksum type (0 for mandatory type)

[in] key - Encryption key for a keyed checksum

[in] usage - Key usage (see KRB5_KEYUSAGE types)

[in] input - Input data

[out] cksum - Generated checksum

retval

- 0 Success; otherwise - Kerberos error codes

This function computes a checksum of type *cksumtype* over *input* , using *key* if the checksum type is a keyed checksum. If *cksumtype* is 0 and *key* is non-null, the checksum type will be the mandatory-to-implement checksum type for the key's encryption type. The actual checksum key will be derived from *key* and *usage* if key derivation is specified for the checksum type. The newly created *cksum* must be released by calling *krb5_free_checksum_contents()* when it is no longer needed.

See also:

krb5_c_verify_checksum()

Note: This function is similar to *krb5_k_make_checksum()* , but operates on keyblock *key* .

---

krb5_c_make_checksum_iov - Fill in a checksum element in IOV array (operates on keyblock)

krb5_error_code **krb5_c_make_checksum_iov**(*krb5_context context*, *krb5_cksumtype cksumtype*, const *krb5_keyblock* * *key*, *krb5_keyusage usage*, *krb5_crypto_iov* * *data*, size_t *num_data*)

param **[in] context** - Library context

[in] cksumtype - Checksum type (0 for mandatory type)

[in] key - Encryption key for a keyed checksum

[in] usage - Key usage (see KRB5_KEYUSAGE types)

[inout] data - IOV array

[in] num_data - Size of *data*

retval

- 0 Success; otherwise - Kerberos error codes

Create a checksum in the *KRB5_CRYPTO_TYPE_CHECKSUM* element over *KRB5_CRYPTO_TYPE_DATA* and *KRB5_CRYPTO_TYPE_SIGN_ONLY* chunks in *data* . Only the *KRB5_CRYPTO_TYPE_CHECKSUM* region is modified.

See also:

krb5_c_verify_checksum_iov()

Note: This function is similar to *krb5_k_make_checksum_iov()* , but operates on keyblock *key* .

---

krb5_c_make_random_key - Generate an enctype-specific random encryption key.

krb5_error_code **krb5_c_make_random_key**(*krb5_context context*, *krb5_enctype enctype*, *krb5_keyblock* * *k5_random_key*)

param **[in] context** - Library context

[in] enctype - Encryption type of the generated key

[out] k5_random_key - An allocated and initialized keyblock

retval

- 0 Success; otherwise - Kerberos error codes

Use *krb5_free_keyblock_contents()* to free *k5_random_key* when no longer needed.

krb5_c_padding_length - Return a number of padding octets.

krb5_error_code **krb5_c_padding_length**(*krb5_context context*, *krb5_enctype enctype*, size_t *data_length*, unsigned int * *size*)

param **[in] context** - Library context

[in] enctype - Encryption type

[in] data_length - Length of the plaintext to pad

[out] size - Number of padding octets

retval

- 0 Success; otherwise - KRB5_BAD_ENCTYPE

This function returns the number of the padding octets required to pad *data_length* octets of plaintext.

krb5_c_prf - Generate enctype-specific pseudo-random bytes.

krb5_error_code **krb5_c_prf** (*krb5_context context*, const *krb5_keyblock* * *keyblock*, *krb5_data* * *input*, *krb5_data* * *output*)

param **[in] context** - Library context

[in] keyblock - Key

[in] input - Input data

[out] output - Output data

retval

- 0 Success; otherwise - Kerberos error codes

This function selects a pseudo-random function based on *keyblock* and computes its value over *input* , placing the result into *output* . The caller must preinitialize *output* and allocate space for the result, using `krb5_c_prf_length()` to determine the required length.

krb5_c_prfplus - Generate pseudo-random bytes using RFC 6113 PRF+.

krb5_error_code **krb5_c_prfplus** (*krb5_context context*, const *krb5_keyblock* * *k*, const *krb5_data* * *input*, *krb5_data* * *output*)

param **[in] context** - Library context

[in] k - KDC contribution key

[in] input - Input data

[out] output - Pseudo-random output buffer

return

- 0 on success, E2BIG if output->length is too large for PRF+ to generate, ENOMEM on allocation failure, or an error code from krb5_c_prf()

This function fills *output* with PRF+(k, input) as defined in RFC 6113 section 5.1. The caller must preinitialize *output* and allocate the desired amount of space. The length of the pseudo-random output will match the length of *output* .

Note: RFC 4402 defines a different PRF+ operation. This function does not implement that operation.

---

krb5_c_prf_length - Get the output length of pseudo-random functions for an encryption type.

krb5_error_code **krb5_c_prf_length** (*krb5_context context*, *krb5_enctype enctype*, size_t * *len*)

param **[in] context** - Library context

[in] enctype - Encryption type

[out] len - Length of PRF output

retval

- 0 Success; otherwise - Kerberos error codes

krb5_c_random_add_entropy - Add entropy to the pseudo-random number generator.

krb5_error_code **krb5_c_random_add_entropy** (*krb5_context context*, unsigned int *randsource*, const *krb5_data* * *data*)

param [in] context - Library context

[in] randsource - Entropy source (see KRB5_RANDSOURCE types)

[in] data - Data

retval

- 0 Success; otherwise - Kerberos error codes

Contribute entropy to the PRNG used by krb5 crypto operations. This may or may not affect the output of the next crypto operation requiring random data.

krb5_c_random_make_octets - Generate pseudo-random bytes.

krb5_error_code **krb5_c_random_make_octets** (*krb5_context context*, *krb5_data* * *data*)

param [in] context - Library context

[out] data - Random data

retval

- 0 Success; otherwise - Kerberos error codes

Fills in *data* with bytes from the PRNG used by krb5 crypto operations. The caller must preinitialize *data* and allocate the desired amount of space.

krb5_c_random_os_entropy - Collect entropy from the OS if possible.

krb5_error_code **krb5_c_random_os_entropy** (*krb5_context context*, int *strong*, int * *success*)

param [in] context - Library context

[in] strong - Strongest available source of entropy

[out] success - 1 if OS provides entropy, 0 otherwise

retval

- 0 Success; otherwise - Kerberos error codes

If *strong* is non-zero, this function attempts to use the strongest available source of entropy. Setting this flag may cause the function to block on some operating systems. Good uses include seeding the PRNG for kadmind and realm setup.

krb5_c_random_to_key - Generate an enctype-specific key from random data.

krb5_error_code **krb5_c_random_to_key** (*krb5_context context*, *krb5_enctype enctype*, *krb5_data* * *random_data*, *krb5_keyblock* * *k5_random_key*)

param **[in] context** - Library context

[in] enctype - Encryption type

[in] random_data - Random input data

[out] k5_random_key - Resulting key

retval

- 0 Success; otherwise - Kerberos error codes

This function takes random input data *random_data* and produces a valid key *k5_random_key* for a given *enctype* .

See also:

krb5_c_keylengths()

Note: It is assumed that *k5_random_key* has already been initialized and *k5_random_key->contents* has been allocated with the correct length.

---

krb5_c_string_to_key - Convert a string (such a password) to a key.

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*)

param **[in] context** - Library context

[in] enctype - Encryption type

[in] string - String to be converted

[in] salt - Salt value

[out] key - Generated key

retval

- 0 Success; otherwise - Kerberos error codes

This function converts *string* to a *key* of encryption type *enctype* , using the specified *salt* . The newly created *key* must be released by calling *krb5_free_keyblock_contents()* when it is no longer needed.

krb5_c_string_to_key_with_params - Convert a string (such as a password) to a key with additional parameters.

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*)

param **[in] context** - Library context

[in] enctype - Encryption type

[in] string - String to be converted

[in] salt - Salt value

[in] params - Parameters

[out] key - Generated key

retval

- 0 Success; otherwise - Kerberos error codes

This function is similar to *krb5_c_string_to_key()* , but also takes parameters which may affect the algorithm in an enctype-dependent way. The newly created *key* must be released by calling *krb5_free_keyblock_contents()* when it is no longer needed.

krb5_c_valid_cksumtype - Verify that specified checksum type is a valid Kerberos checksum type.

krb5_boolean **krb5_c_valid_cksumtype**(*krb5_cksumtype ctype*)

param [in] ctype - Checksum type

return

- TRUE if ctype is valid, FALSE if not

krb5_c_valid_enctype - Verify that a specified encryption type is a valid Kerberos encryption type.

krb5_boolean **krb5_c_valid_enctype**(*krb5_enctype ktype*)

param [in] ktype - Encryption type

return

- TRUE if ktype is valid, FALSE if not

krb5_c_verify_checksum - Verify a checksum (operates on keyblock).

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*)

param [in] context - Library context

[in] key - Encryption key for a keyed checksum

[in] usage - *key* usage

[in] data - Data to be used to compute a new checksum using *key* to compare *cksum* against

[in] cksum - Checksum to be verified

[out] valid - Non-zero for success, zero for failure

retval

- 0 Success; otherwise - Kerberos error codes

This function verifies that *cksum* is a valid checksum for *data* . If the checksum type of *cksum* is a keyed checksum, *key* is used to verify the checksum. If the checksum type in *cksum* is 0 and *key* is not NULL, the mandatory checksum type for *key* will be used. The actual checksum key will be derived from *key* and *usage* if key derivation is specified for the checksum type.

Note: This function is similar to *krb5_k_verify_checksum()* , but operates on keyblock *key* .

krb5_c_verify_checksum_iov - Validate a checksum element in IOV array (operates on keyblock).

krb5_error_code **krb5_c_verify_checksum_iov**(*krb5_context context*, *krb5_cksumtype cksumtype*, const *krb5_keyblock* * *key*, *krb5_keyusage usage*, const *krb5_crypto_iov* * *data*, size_t *num_data*, *krb5_boolean* * *valid*)

param **[in] context** - Library context

[in] cksumtype - Checksum type (0 for mandatory type)

[in] key - Encryption key for a keyed checksum

[in] usage - Key usage (see `KRB5_KEYUSAGE` types)

[in] data - IOV array

[in] num_data - Size of *data*

[out] valid - Non-zero for success, zero for failure

retval

- 0 Success; otherwise - Kerberos error codes

Confirm that the checksum in the *KRB5_CRYPTO_TYPE_CHECKSUM* element is a valid checksum of the *KRB5_CRYPTO_TYPE_DATA* and *KRB5_CRYPTO_TYPE_SIGN_ONLY* regions in the iov.

See also:

krb5_c_make_checksum_iov()

Note: This function is similar to *krb5_k_verify_checksum_iov()* , but operates on keyblock *key* .

---

krb5_cksumtype_to_string - Convert a checksum type to a string.

krb5_error_code **krb5_cksumtype_to_string**(*krb5_cksumtype cksumtype*, char * *buffer*, size_t *buflen*)

param **[in] cksumtype** - Checksum type

[out] buffer - Buffer to hold converted checksum type

[in] buflen - Storage available in *buffer*

retval

- 0 Success; otherwise - Kerberos error codes

krb5_decode_authdata_container - Unwrap authorization data.

krb5_error_code **krb5_decode_authdata_container**(*krb5_context context*, *krb5_authdatatype type*, const *krb5_authdata* * *container*, *krb5_authdata* *** *authdata*)

param **[in] context** - Library context

[in] type - `KRB5_AUTHDATA` type of *container*

[in] container - Authorization data to be decoded

[out] authdata - List of decoded authorization data

retval

- 0 Success; otherwise - Kerberos error codes

See also:

krb5_encode_authdata_container()

krb5_decode_ticket - Decode an ASN.1-formatted ticket.

krb5_error_code **krb5_decode_ticket** (const *krb5_data* * *code*, *krb5_ticket* ** *rep*)

param [in] code - ASN.1-formatted ticket

[out] rep - Decoded ticket information

retval

- 0 Success; otherwise - Kerberos error codes

krb5_deltat_to_string - Convert a relative time value to a string.

krb5_error_code **krb5_deltat_to_string** (*krb5_deltat deltat*, char * *buffer*, size_t *buflen*)

param [in] deltat - Relative time value to convert

[out] buffer - Buffer to hold time string

[in] buflen - Storage available in *buffer*

retval

- 0 Success; otherwise - Kerberos error codes

krb5_encode_authdata_container - Wrap authorization data in a container.

krb5_error_code **krb5_encode_authdata_container** (*krb5_context context*, *krb5_authdatatype type*, *krb5_authdata* *const * *authdata*, *krb5_authdata* *** *container*)

param [in] context - Library context

[in] type - KRB5_AUTHDATA type of *container*

[in] authdata - List of authorization data to be encoded

[out] container - List of encoded authorization data

retval

- 0 Success; otherwise - Kerberos error codes

The result is returned in *container* as a single-element list.

See also:

krb5_decode_authdata_container()

krb5_enctype_to_name - Convert an encryption type to a name or alias.

krb5_error_code **krb5_enctype_to_name** (*krb5_enctype enctype*, *krb5_boolean shortest*, char * *buffer*, size_t *buflen*)

param **[in] enctype** - Encryption type

[in] shortest - Flag

[out] buffer - Buffer to hold encryption type string

[in] buflen - Storage available in *buffer*

retval

- 0 Success; otherwise - Kerberos error codes

If *shortest* is FALSE, this function returns the enctype's canonical name (like"aes128-cts-hmac-sha1-96"). If *shortest* is TRUE, it return the enctype's shortest alias (like"aes128-cts").

Note: New in 1.9

krb5_enctype_to_string - Convert an encryption type to a string.

krb5_error_code **krb5_enctype_to_string** (*krb5_enctype enctype*, char * *buffer*, size_t *buflen*)

param **[in] enctype** - Encryption type

[out] buffer - Buffer to hold encryption type string

[in] buflen - Storage available in *buffer*

retval

- 0 Success; otherwise - Kerberos error codes

krb5_free_checksum - Free a krb5_checksum structure.

void **krb5_free_checksum** (*krb5_context context*, *krb5_checksum* * *val*)

param **[in] context** - Library context

[in] val - Checksum structure to be freed

This function frees the contents of *val* and the structure itself.

krb5_free_checksum_contents - Free the contents of a krb5_checksum structure.

void **krb5_free_checksum_contents** (*krb5_context context*, *krb5_checksum* * *val*)

param **[in] context** - Library context

[in] val - Checksum structure to free contents of

This function frees the contents of *val* , but not the structure itself.

krb5_free_cksumtypes - Free an array of checksum types.

void **krb5_free_cksumtypes** (*krb5_context context*, *krb5_cksumtype * val*)

param **[in] context** - Library context

[in] val - Array of checksum types to be freed

krb5_free_tgt_creds - Free an array of credential structures.

void **krb5_free_tgt_creds** (*krb5_context context*, *krb5_creds ** tgts*)

param **[in] context** - Library context

[in] tgts - Null-terminated array of credentials to free

Note: The last entry in the array *tgts* must be a NULL pointer.

---

krb5_k_create_key - Create a krb5_key from the enctype and key data in a keyblock.

krb5_error_code **krb5_k_create_key** (*krb5_context context*, const *krb5_keyblock * key_data*, *krb5_key * out*)

param **[in] context** - Library context

[in] key_data - Keyblock

[out] out - Opaque key

retval

- 0 Success; otherwise - KRB5_BAD_ENCTYPE

The reference count on a key *out* is set to 1. Use `krb5_k_free_key()` to free *out* when it is no longer needed.

krb5_k_decrypt - Decrypt data using a key (operates on opaque key).

krb5_error_code **krb5_k_decrypt** (*krb5_context context*, *krb5_key key*, *krb5_keyusage usage*, const *krb5_data * cipher_state*, const *krb5_enc_data * input*, *krb5_data * output*)

param **[in] context** - Library context

[in] key - Encryption key

[in] usage - Key usage (see `KRB5_KEYUSAGE` types)

[inout] cipher_state - Cipher state; specify NULL if not needed

[in] input - Encrypted data

[out] output - Decrypted data

retval

- 0 Success; otherwise - Kerberos error codes

This function decrypts the data block *input* and stores the output into *output* . The actual decryption key will be derived from *key* and *usage* if key derivation is specified for the encryption type. If non-null, *cipher_state* specifies the beginning state for the decryption operation, and is updated with the state to be passed as input to the next operation.

Note: The caller must initialize *output* and allocate at least enough space for the result. The usual practice is to allocate an output buffer as long as the ciphertext, and let `krb5_c_decrypt()` trim *output->length* . For some enctypes, the resulting *output->length* may include padding bytes.

---

krb5_k_decrypt_iov - Decrypt data in place supporting AEAD (operates on opaque key).

krb5_error_code **krb5_k_decrypt_iov**(*krb5_context context*, *krb5_key key*, *krb5_keyusage usage*, const *krb5_data* * *cipher_state*, *krb5_crypto_iov* * *data*, size_t *num_data*)

param **[in] context** - Library context

[in] key - Encryption key

[in] usage - Key usage (see `KRB5_KEYUSAGE` types)

[in] cipher_state - Cipher state; specify NULL if not needed

[inout] data - IOV array. Modified in-place.

[in] num_data - Size of *data*

retval

- 0 Success; otherwise - Kerberos error codes

This function decrypts the data block *data* and stores the output in-place. The actual decryption key will be derived from *key* and *usage* if key derivation is specified for the encryption type. If non-null, *cipher_state* specifies the beginning state for the decryption operation, and is updated with the state to be passed as input to the next operation. The caller must allocate the right number of krb5_crypto_iov structures before calling into this API.

See also:

`krb5_k_encrypt_iov()`

Note: On return from a `krb5_c_decrypt_iov()` call, the *data->length* in the iov structure are adjusted to reflect actual lengths of the ciphertext used. For example, if the padding length is too large, the length will be reduced. Lengths are never increased.

---

krb5_k_encrypt - Encrypt data using a key (operates on opaque key).

krb5_error_code **krb5_k_encrypt**(*krb5_context context*, *krb5_key key*, *krb5_keyusage usage*, const *krb5_data* * *cipher_state*, const *krb5_data* * *input*, *krb5_enc_data* * *output*)

param **[in] context** - Library context

[in] key - Encryption key

[in] usage - Key usage (see `KRB5_KEYUSAGE` types)

[inout] cipher_state - Cipher state; specify NULL if not needed

[in] input - Data to be encrypted

[out] output - Encrypted data

retval

- 0 Success; otherwise - Kerberos error codes

This function encrypts the data block *input* and stores the output into *output* . The actual encryption key will be derived from *key* and *usage* if key derivation is specified for the encryption type. If non-null, *cipher_state* specifies the beginning state for the encryption operation, and is updated with the state to be passed as input to the next operation.

Note: The caller must initialize *output* and allocate at least enough space for the result (using `krb5_c_encrypt_length()` to determine the amount of space needed). *output->length* will be set to the actual length of the ciphertext.

---

krb5_k_encrypt_iov - Encrypt data in place supporting AEAD (operates on opaque key).

krb5_error_code **krb5_k_encrypt_iov**(*krb5_context context*, *krb5_key key*, *krb5_keyusage usage*, const *krb5_data* * *cipher_state*, *krb5_crypto_iov* * *data*, size_t *num_data*)

param [in] context - Library context

[in] key - Encryption key

[in] usage - Key usage (see `KRB5_KEYUSAGE` types)

[in] cipher_state - Cipher state; specify NULL if not needed

[inout] data - IOV array. Modified in-place.

[in] num_data - Size of *data*

retval

- 0 Success; otherwise - Kerberos error codes

This function encrypts the data block *data* and stores the output in-place. The actual encryption key will be derived from *key* and *usage* if key derivation is specified for the encryption type. If non-null, *cipher_state* specifies the beginning state for the encryption operation, and is updated with the state to be passed as input to the next operation. The caller must allocate the right number of krb5_crypto_iov structures before calling into this API.

See also:

`krb5_k_decrypt_iov()`

Note: On return from a `krb5_c_encrypt_iov()` call, the *data->length* in the iov structure are adjusted to reflect actual lengths of the ciphertext used. For example, if the padding length is too large, the length will be reduced. Lengths are never increased.

---

krb5_k_free_key - Decrement the reference count on a key and free it if it hits zero.

void **krb5_k_free_key**(*krb5_context context*, *krb5_key key*)

param context

key

krb5_k_key_enctype - Retrieve the enctype of a krb5_key structure.

krb5_enctype **krb5_k_key_enctype**(*krb5_context context*, *krb5_key key*)

param context

key

krb5_k_key_keyblock - Retrieve a copy of the keyblock from a krb5_key structure.

krb5_error_code **krb5_k_key_keyblock**(*krb5_context context*, *krb5_key key*, *krb5_keyblock* ** *key_data*)

param context

key

key_data

krb5_k_make_checksum - Compute a checksum (operates on opaque key).

krb5_error_code **krb5_k_make_checksum**(*krb5_context context*, *krb5_cksumtype cksumtype*, *krb5_key key*, *krb5_keyusage usage*, const *krb5_data* * *input*, *krb5_checksum* * *cksum*)

param [in] context - Library context

[in] cksumtype - Checksum type (0 for mandatory type)

[in] key - Encryption key for a keyed checksum

[in] usage - Key usage (see `KRB5_KEYUSAGE` types)

[in] input - Input data

[out] cksum - Generated checksum

retval

- 0 Success; otherwise - Kerberos error codes

This function computes a checksum of type *cksumtype* over *input* , using *key* if the checksum type is a keyed checksum. If *cksumtype* is 0 and *key* is non-null, the checksum type will be the mandatory-to-implement checksum type for the key's encryption type. The actual checksum key will be derived from *key* and *usage* if key derivation is specified for the checksum type. The newly created *cksum* must be released by calling *krb5_free_checksum_contents()* when it is no longer needed.

See also:

krb5_c_verify_checksum()

Note: This function is similar to *krb5_c_make_checksum()* , but operates on opaque *key* .

---

krb5_k_make_checksum_iov - Fill in a checksum element in IOV array (operates on opaque key)

krb5_error_code **krb5_k_make_checksum_iov**(*krb5_context context*, *krb5_cksumtype cksumtype*, *krb5_key key*, *krb5_keyusage usage*, *krb5_crypto_iov* * *data*, size_t *num_data*)

param **[in] context** - Library context

[in] cksumtype - Checksum type (0 for mandatory type)

[in] key - Encryption key for a keyed checksum

[in] usage - Key usage (see KRB5_KEYUSAGE types)

[inout] data - IOV array

[in] num_data - Size of *data*

retval

- 0 Success; otherwise - Kerberos error codes

Create a checksum in the *KRB5_CRYPTO_TYPE_CHECKSUM* element over *KRB5_CRYPTO_TYPE_DATA* and *KRB5_CRYPTO_TYPE_SIGN_ONLY* chunks in *data* . Only the *KRB5_CRYPTO_TYPE_CHECKSUM* region is modified.

See also:

krb5_k_verify_checksum_iov()

Note: This function is similar to *krb5_c_make_checksum_iov()* , but operates on opaque *key* .

---

krb5_k_prf - Generate enctype-specific pseudo-random bytes (operates on opaque key).

krb5_error_code **krb5_k_prf** (*krb5_context context*, *krb5_key key*, *krb5_data * input*, *krb5_data * output*)

param **[in] context** - Library context

[in] key - Key

[in] input - Input data

[out] output - Output data

retval

- 0 Success; otherwise - Kerberos error codes

This function selects a pseudo-random function based on *key* and computes its value over *input* , placing the result into *output* . The caller must preinitialize *output* and allocate space for the result.

Note: This function is similar to *krb5_c_prf()* , but operates on opaque *key* .

---

krb5_k_reference_key - Increment the reference count on a key.

void **krb5_k_reference_key** (*krb5_context context*, *krb5_key key*)

param **context**

key

krb5_k_verify_checksum - Verify a checksum (operates on opaque key).

krb5_error_code **krb5_k_verify_checksum**(*krb5_context context*, *krb5_key key*, *krb5_keyusage usage*, const *krb5_data* * *data*, const *krb5_checksum* * *cksum*, *krb5_boolean* * *valid*)

param **[in] context** - Library context

[in] key - Encryption key for a keyed checksum

[in] usage - *key* usage

[in] data - Data to be used to compute a new checksum using *key* to compare *cksum* against

[in] cksum - Checksum to be verified

[out] valid - Non-zero for success, zero for failure

retval

- 0 Success; otherwise - Kerberos error codes

This function verifies that *cksum* is a valid checksum for *data* . If the checksum type of *cksum* is a keyed checksum, *key* is used to verify the checksum. If the checksum type in *cksum* is 0 and *key* is not NULL, the mandatory checksum type for *key* will be used. The actual checksum key will be derived from *key* and *usage* if key derivation is specified for the checksum type.

Note: This function is similar to *krb5_c_verify_checksum()* , but operates on opaque *key* .

---

krb5_k_verify_checksum_iov - Validate a checksum element in IOV array (operates on opaque key).

krb5_error_code **krb5_k_verify_checksum_iov**(*krb5_context context*, *krb5_cksumtype cksumtype*, *krb5_key key*, *krb5_keyusage usage*, const *krb5_crypto_iov* * *data*, size_t *num_data*, *krb5_boolean* * *valid*)

param **[in] context** - Library context

[in] cksumtype - Checksum type (0 for mandatory type)

[in] key - Encryption key for a keyed checksum

[in] usage - Key usage (see KRB5_KEYUSAGE types)

[in] data - IOV array

[in] num_data - Size of *data*

[out] valid - Non-zero for success, zero for failure

retval

- 0 Success; otherwise - Kerberos error codes

Confirm that the checksum in the *KRB5_CRYPTO_TYPE_CHECKSUM* element is a valid checksum of the *KRB5_CRYPTO_TYPE_DATA* and *KRB5_CRYPTO_TYPE_SIGN_ONLY* regions in the iov.

See also:

krb5_k_make_checksum_iov()

Note: This function is similar to *krb5_c_verify_checksum_iov()* , but operates on opaque *key* .

6.1.4 Legacy convenience interfaces

krb5_recvauth - Server function for sendauth protocol.

krb5_error_code **krb5_recvauth**(*krb5_context context*, *krb5_auth_context * auth_context*, *krb5_pointer fd*, char * *appl_version*, *krb5_principal server*, *krb5_int32 flags*, *krb5_keytab keytab*, *krb5_ticket ** *ticket*)

param [in] context - Library context

[inout] auth_context - Pre-existing or newly created auth context

[in] fd - File descriptor

[in] appl_version - Application protocol version to be matched against the client's application version

[in] server - Server principal (NULL for any in *keytab*)

[in] flags - Additional specifications

[in] keytab - Key table containing service keys

[out] ticket - Ticket (NULL if not needed)

retval

- 0 Success; otherwise - Kerberos error codes

This function performs the server side of a sendauth/recvauth exchange by sending and receiving messages over *fd* .

Use *krb5_free_ticket()* to free *ticket* when it is no longer needed.

See also:

krb5_sendauth()

krb5_recvauth_version - Server function for sendauth protocol with version parameter.

krb5_error_code **krb5_recvauth_version**(*krb5_context context*, *krb5_auth_context * auth_context*, *krb5_pointer fd*, *krb5_principal server*, *krb5_int32 flags*, *krb5_keytab keytab*, *krb5_ticket ** *ticket*, *krb5_data * version*)

param [in] context - Library context

[inout] auth_context - Pre-existing or newly created auth context

[in] fd - File descriptor

[in] server - Server principal (NULL for any in *keytab*)

[in] flags - Additional specifications

[in] keytab - Decryption key

[out] ticket - Ticket (NULL if not needed)

[out] version - sendauth protocol version (NULL if not needed)

retval

- 0 Success; otherwise - Kerberos error codes

This function is similar to *krb5_recvauth()* with the additional output information place into *version* .

krb5_sendauth - Client function for sendauth protocol.

krb5_error_code **krb5_sendauth**(*krb5_context context*, *krb5_auth_context * auth_context*, *krb5_pointer fd*, char * *appl_version*, *krb5_principal client*, *krb5_principal server*, *krb5_flags ap_req_options*, *krb5_data * in_data*, *krb5_creds * in_creds*, *krb5_ccache ccache*, *krb5_error ** *error*, *krb5_ap_rep_enc_part ** *rep_result*, *krb5_creds ** *out_creds*)

param [in] context - Library context

[inout] auth_context - Pre-existing or newly created auth context

[in] fd - File descriptor that describes network socket

[in] appl_version - Application protocol version to be matched with the receiver's application version

[in] client - Client principal

[in] server - Server principal

[in] ap_req_options - AP_OPTS options

[in] in_data - Data to be sent to the server

[in] in_creds - Input credentials, or NULL to use *ccache*

[in] ccache - Credential cache

[out] error - If non-null, contains KRB_ERROR message returned from server

[out] rep_result - If non-null and *ap_req_options* is *AP_OPTS_MUTUAL_REQUIRED* , contains the result of mutual authentication exchange

[out] out_creds - If non-null, the retrieved credentials

retval

- 0 Success; otherwise - Kerberos error codes

This function performs the client side of a sendauth/recvauth exchange by sending and receiving messages over *fd* .

Credentials may be specified in three ways:

- If *in_creds* is NULL, credentials are obtained with *krb5_get_credentials()* using the principals *client* and *server* . *server* must be non-null; *client* may NULL to use the default principal of *ccache* .
- If *in_creds* is non-null, but does not contain a ticket, credentials for the exchange are obtained with *krb5_get_credentials()* using *in_creds* . In this case, the values of *client* and *server* are unused.
- If *in_creds* is a complete credentials structure, it used directly. In this case, the values of *client* , *server* , and *ccache* are unused.

If the server is using a different application protocol than that specified in *appl_version* , an error will be returned.

Use *krb5_free_creds()* to free *out_creds* , *krb5_free_ap_rep_enc_part()* to free *rep_result* , and *krb5_free_error()* to free *error* when they are no longer needed.

See also:

krb5_recvauth()

6.1.5 Deprecated public interfaces

krb5_524_convert_creds - Convert a Kerberos V5 credentials to a Kerberos V4 credentials.

int **krb5_524_convert_creds** (*krb5_context context*, *krb5_creds * v5creds*, struct credentials * *v4creds*)

param context

v5creds

v4creds

retval

- KRB524_KRB4_DISABLED (always)

Note: Not implemented

---

krb5_auth_con_getlocalsubkey

krb5_error_code **krb5_auth_con_getlocalsubkey** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_keyblock ** keyblock*)

param context

auth_context

keyblock

DEPRECATED Replaced by krb5_auth_con_getsendsubkey() .

krb5_auth_con_getremotesubkey

krb5_error_code **krb5_auth_con_getremotesubkey** (*krb5_context context*, *krb5_auth_context auth_context*, *krb5_keyblock ** keyblock*)

param context

auth_context

keyblock

DEPRECATED Replaced by krb5_auth_con_getrecvsubkey() .

krb5_auth_con_initivector - Cause an auth context to use cipher state.

krb5_error_code **krb5_auth_con_initivector**(*krb5_context context*, *krb5_auth_context auth_context*)

param [in] context - Library context

[in] auth_context - Authentication context

retval

- 0 Success; otherwise - Kerberos error codes

Prepare *auth_context* to use cipher state when `krb5_mk_priv()` or `krb5_rd_priv()` encrypt or decrypt data.

krb5_build_principal_va

krb5_error_code **krb5_build_principal_va**(*krb5_context context*, *krb5_principal princ*, unsigned int *rlen*, const char * *realm*, va_list *ap*)

param context

princ

rlen

realm

ap

DEPRECATED Replaced by krb5_build_principal_alloc_va() .

krb5_c_random_seed

krb5_error_code **krb5_c_random_seed**(*krb5_context context*, *krb5_data* * *data*)

param context

data

DEPRECATED Replaced by krb5_c_* API family.

krb5_calculate_checksum

krb5_error_code **krb5_calculate_checksum**(*krb5_context context*, *krb5_cksumtype ctype*, *krb5_const_pointer in*, size_t *in_length*, *krb5_const_pointer seed*, size_t *seed_length*, *krb5_checksum* * *outcksum*)

param context

ctype

in

in_length

seed

seed_length

outcksum

DEPRECATED See krb5_c_make_checksum()

krb5_checksum_size

size_t **krb5_checksum_size** (*krb5_context context*, *krb5_cksumtype ctype*)

param context

ctype

DEPRECATED See krb5_c_checksum_length()

krb5_encrypt

krb5_error_code **krb5_encrypt** (*krb5_context context*, *krb5_const_pointer inptr*, *krb5_pointer outptr*, size_t *size*, *krb5_encrypt_block* * *eblock*, *krb5_pointer ivec*)

param context

inptr

outptr

size

eblock

ivec

DEPRECATED Replaced by krb5_c_* API family.

krb5_decrypt

krb5_error_code **krb5_decrypt** (*krb5_context context*, *krb5_const_pointer inptr*, *krb5_pointer outptr*, size_t *size*, *krb5_encrypt_block* * *eblock*, *krb5_pointer ivec*)

param context

inptr

outptr

size

eblock

ivec

DEPRECATED Replaced by krb5_c_* API family.

krb5_eblock_enctype

krb5_enctype **krb5_eblock_enctype** (*krb5_context context*, const *krb5_encrypt_block* * *eblock*)

param context

eblock

DEPRECATED Replaced by krb5_c_* API family.

krb5_encrypt_size

size_t **krb5_encrypt_size**(size_t *length*, *krb5_enctype crypto*)

param length

crypto

DEPRECATED Replaced by krb5_c_* API family.

krb5_finish_key

krb5_error_code **krb5_finish_key**(*krb5_context context*, *krb5_encrypt_block* * *eblock*)

param context

eblock

DEPRECATED Replaced by krb5_c_* API family.

krb5_finish_random_key

krb5_error_code **krb5_finish_random_key**(*krb5_context context*, const *krb5_encrypt_block* * *eblock*, *krb5_pointer* * *ptr*)

param context

eblock

ptr

DEPRECATED Replaced by krb5_c_* API family.

krb5_cc_gen_new

krb5_error_code **krb5_cc_gen_new**(*krb5_context context*, *krb5_ccache* * *cache*)

param context

cache

krb5_get_credentials_renew

krb5_error_code **krb5_get_credentials_renew**(*krb5_context context*, *krb5_flags options*, *krb5_ccache ccache*, *krb5_creds* * *in_creds*, *krb5_creds* ** *out_creds*)

param context

options

ccache

in_creds

out_creds

DEPRECATED Replaced by krb5_get_renewed_creds.

krb5_get_credentials_validate

krb5_error_code **krb5_get_credentials_validate**(*krb5_context context*, *krb5_flags options*, *krb5_ccache ccache*, *krb5_creds * in_creds*, *krb5_creds ** *out_creds*)

param context

options

ccache

in_creds

out_creds

DEPRECATED Replaced by krb5_get_validated_creds.

krb5_get_in_tkt_with_password

krb5_error_code **krb5_get_in_tkt_with_password**(*krb5_context context*, *krb5_flags options*, *krb5_address* *const * *addrs*, *krb5_enctype * ktypes*, *krb5_preauthtype * pre_auth_types*, const char * *password*, *krb5_ccache ccache*, *krb5_creds * creds*, *krb5_kdc_rep ** *ret_as_reply*)

param context

options

addrs

ktypes

pre_auth_types

password

ccache

creds

ret_as_reply

DEPRECATED Replaced by krb5_get_init_creds_password() .

krb5_get_in_tkt_with_skey

krb5_error_code **krb5_get_in_tkt_with_skey**(*krb5_context context*, *krb5_flags options*, *krb5_address* *const * *addrs*, *krb5_enctype * ktypes*, *krb5_preauthtype * pre_auth_types*, const *krb5_keyblock * key*, *krb5_ccache ccache*, *krb5_creds * creds*, *krb5_kdc_rep ** *ret_as_reply*)

param context

options

addrs

ktypes

pre_auth_types

key

ccache

creds

ret_as_reply

DEPRECATED Replaced by krb5_get_init_creds().

krb5_get_in_tkt_with_keytab

krb5_error_code **krb5_get_in_tkt_with_keytab** (*krb5_context context*, *krb5_flags options*, *krb5_address* *const * *addrs*, *krb5_enctype* * *ktypes*, *krb5_preauthtype* * *pre_auth_types*, *krb5_keytab arg_keytab*, *krb5_ccache ccache*, *krb5_creds* * *creds*, *krb5_kdc_rep* ** *ret_as_reply*)

param context

options

addrs

ktypes

pre_auth_types

arg_keytab

ccache

creds

ret_as_reply

DEPRECATED Replaced by krb5_get_init_creds_keytab() .

krb5_get_init_creds_opt_init

void **krb5_get_init_creds_opt_init** (*krb5_get_init_creds_opt* * *opt*)

param opt

DEPRECATED Use krb5_get_init_creds_opt_alloc() instead.

krb5_init_random_key

krb5_error_code **krb5_init_random_key** (*krb5_context context*, const *krb5_encrypt_block* * *eblock*, const *krb5_keyblock* * *keyblock*, *krb5_pointer* * *ptr*)

param context

eblock

keyblock

ptr

DEPRECATED Replaced by krb5_c_* API family.

krb5_kt_free_entry

krb5_error_code **krb5_kt_free_entry**(*krb5_context context*, *krb5_keytab_entry * entry*)

param context

entry

DEPRECATED Use krb5_free_keytab_entry_contents instead.

krb5_random_key

krb5_error_code **krb5_random_key**(*krb5_context context*, const *krb5_encrypt_block * eblock*, *krb5_pointer ptr*, *krb5_keyblock ** *keyblock*)

param context

eblock

ptr

keyblock

DEPRECATED Replaced by krb5_c_* API family.

krb5_process_key

krb5_error_code **krb5_process_key**(*krb5_context context*, *krb5_encrypt_block * eblock*, const *krb5_keyblock * key*)

param context

eblock

key

DEPRECATED Replaced by krb5_c_* API family.

krb5_string_to_key

krb5_error_code **krb5_string_to_key**(*krb5_context context*, const *krb5_encrypt_block * eblock*, *krb5_keyblock * keyblock*, const *krb5_data * data*, const *krb5_data * salt*)

param context

eblock

keyblock

data

salt

DEPRECATED See krb5_c_string_to_key()

krb5_use_enctype

krb5_error_code **krb5_use_enctype**(*krb5_context context*, *krb5_encrypt_block * eblock*, *krb5_enctype enctype*)

param context

eblock

enctype

DEPRECATED Replaced by krb5_c_* API family.

krb5_verify_checksum

krb5_error_code **krb5_verify_checksum**(*krb5_context context*, *krb5_cksumtype ctype*, const *krb5_checksum * cksum*, *krb5_const_pointer in*, size_t *in_length*, *krb5_const_pointer seed*, size_t *seed_length*)

param context

ctype

cksum

in

in_length

seed

seed_length

DEPRECATED See krb5_c_verify_checksum()

6.2 krb5 types and structures

6.2.1 Public

krb5_address

krb5_address

Structure for address.

Declaration

typedef struct _krb5_address krb5_address

Members

krb5_magic **krb5_address.magic**

krb5_addrtype **krb5_address.addrtype**

unsigned int **krb5_address.length**

krb5_octet * **krb5_address.contents**

krb5_addrtype

krb5_addrtype

Declaration

typedef krb5_int32 krb5_addrtype

krb5_ap_req

krb5_ap_req

Authentication header.

Declaration

typedef struct _krb5_ap_req krb5_ap_req

Members

krb5_magic **krb5_ap_req.magic**

krb5_flags **krb5_ap_req.ap_options**
Requested options.

krb5_ticket * **krb5_ap_req.ticket**
Ticket.

krb5_enc_data **krb5_ap_req.authenticator**
Encrypted authenticator.

krb5_ap_rep

krb5_ap_rep

C representaton of AP-REP message.

The server's response to a client's request for mutual authentication.

Declaration

typedef struct _krb5_ap_rep krb5_ap_rep

Members

krb5_magic **krb5_ap_rep.magic**

krb5_enc_data **krb5_ap_rep.enc_part**
Ciphertext of ApRepEncPart.

krb5_ap_rep_enc_part

krb5_ap_rep_enc_part

Cleartext that is encrypted and put into `_krb5_ap_rep` .

Declaration

typedef struct _krb5_ap_rep_enc_part krb5_ap_rep_enc_part

Members

krb5_magic **krb5_ap_rep_enc_part.magic**

krb5_timestamp **krb5_ap_rep_enc_part.ctime**
Client time, seconds portion.

krb5_int32 **krb5_ap_rep_enc_part.cusec**
Client time, microseconds portion.

krb5_keyblock * **krb5_ap_rep_enc_part.subkey**
Subkey (optional)

krb5_ui_4 **krb5_ap_rep_enc_part.seq_number**
Sequence number.

krb5_authdata

krb5_authdata

Structure for auth data.

Declaration

typedef struct _krb5_authdata krb5_authdata

Members

krb5_magic **krb5_authdata.magic**

krb5_authdatatype **krb5_authdata.ad_type**
ADTYPE.

unsigned int **krb5_authdata.length**
Length of data.

krb5_octet * **krb5_authdata.contents**
Data.

krb5_authdatatype

krb5_authdatatype

Declaration

typedef krb5_int32 krb5_authdatatype

krb5_authenticator

krb5_authenticator

Ticket authenticator.

The C representation of an unencrypted authenticator.

Declaration

typedef struct _krb5_authenticator krb5_authenticator

Members

krb5_magic **krb5_authenticator.magic**

krb5_principal **krb5_authenticator.client**
client name/realm

krb5_checksum * **krb5_authenticator.checksum**
checksum, includes type, optional

krb5_int32 **krb5_authenticator.cusec**
client usec portion

krb5_timestamp **krb5_authenticator.ctime**
client sec portion

krb5_keyblock * **krb5_authenticator.subkey**
true session key, optional

krb5_ui_4 **krb5_authenticator.seq_number**
sequence #, optional

krb5_authdata ** **krb5_authenticator.authorization_data**
authoriazation data

krb5_boolean

krb5_boolean

Declaration

typedef unsigned int krb5_boolean

krb5_checksum

krb5_checksum

Declaration

typedef struct _krb5_checksum krb5_checksum

Members

krb5_magic **krb5_checksum.magic**

krb5_cksumtype **krb5_checksum.checksum_type**

unsigned int **krb5_checksum.length**

krb5_octet * **krb5_checksum.contents**

krb5_const_pointer

krb5_const_pointer

Declaration

typedef void const* krb5_const_pointer

krb5_const_principal

krb5_const_principal

Constant version of *krb5_principal_data* .

Declaration

typedef const krb5_principal_data* krb5_const_principal

Members

krb5_magic **krb5_const_principal.magic**

krb5_data **krb5_const_principal.realm**

krb5_data * **krb5_const_principal.data**
An array of strings.

krb5_int32 **krb5_const_principal.length**

krb5_int32 **krb5_const_principal.type**

krb5_cred

krb5_cred

Credentials data structure.

Declaration

typedef struct _krb5_cred krb5_cred

Members

krb5_magic **krb5_cred.magic**

krb5_ticket ** **krb5_cred.tickets**
Tickets.

krb5_enc_data **krb5_cred.enc_part**
Encrypted part.

krb5_cred_enc_part * **krb5_cred.enc_part2**
Unencrypted version, if available.

krb5_cred_enc_part

krb5_cred_enc_part

Cleartext credentials information.

Declaration

typedef struct _krb5_cred_enc_part krb5_cred_enc_part

Members

krb5_magic **krb5_cred_enc_part.magic**

krb5_int32 **krb5_cred_enc_part.nonce**
Nonce (optional)

krb5_timestamp **krb5_cred_enc_part.timestamp**
Generation time, seconds portion.

krb5_int32 **krb5_cred_enc_part.usec**
Generation time, microseconds portion.

krb5_address * **krb5_cred_enc_part.s_address**
Sender address (optional)

krb5_address * **krb5_cred_enc_part.r_address**
Recipient address (optional)

krb5_cred_info ** **krb5_cred_enc_part.ticket_info**

krb5_cred_info

krb5_cred_info

Credentials information inserted into *EncKrbCredPart* .

Declaration

typedef struct _krb5_cred_info krb5_cred_info

Members

krb5_magic **krb5_cred_info.magic**

krb5_keyblock * **krb5_cred_info.session**
Session key used to encrypt ticket.

krb5_principal **krb5_cred_info.client**
Client principal and realm.

krb5_principal **krb5_cred_info.server**
Server principal and realm.

krb5_flags **krb5_cred_info.flags**
Ticket flags.

krb5_ticket_times **krb5_cred_info.times**
Auth, start, end, renew_till.

krb5_address ** **krb5_cred_info.caddrs**
Array of pointers to addrs (optional)

krb5_creds

krb5_creds

Credentials structure including ticket, session key, and lifetime info.

Declaration

typedef struct _krb5_creds krb5_creds

Members

krb5_magic **krb5_creds.magic**

krb5_principal **krb5_creds.client**
client's principal identifier

krb5_principal **krb5_creds.server**
server's principal identifier

krb5_keyblock **krb5_creds.keyblock**
session encryption key info

krb5_ticket_times **krb5_creds.times**
lifetime info

krb5_boolean **krb5_creds.is_skey**
true if ticket is encrypted in another ticket's skey

krb5_flags **krb5_creds.ticket_flags**
flags in ticket

krb5_address ** **krb5_creds.addresses**
addrs in ticket

krb5_data **krb5_creds.ticket**
ticket string itself

krb5_data **krb5_creds.second_ticket**
second ticket, if related to ticket (via DUPLICATE-SKEY or ENC-TKT-IN-SKEY)

krb5_authdata ** **krb5_creds.authdata**
authorization data

krb5_crypto_iov

krb5_crypto_iov

Structure to describe a region of text to be encrypted or decrypted.

The *flags* member describes the type of the iov. The *data* member points to the memory that will be manipulated. All iov APIs take a pointer to the first element of an array of krb5_crypto_iov's along with the size of that array. Buffer contents are manipulated in-place; data is overwritten. Callers must allocate the right number of krb5_crypto_iov structures before calling into an iov API.

Declaration

typedef struct _krb5_crypto_iov krb5_crypto_iov

Members

krb5_cryptotype **krb5_crypto_iov.flags**
`KRB5_CRYPTO_TYPE` type of the iov

krb5_data **krb5_crypto_iov.data**

krb5_cryptotype

krb5_cryptotype

Declaration

typedef krb5_int32 krb5_cryptotype

krb5_data

krb5_data

Declaration

typedef struct _krb5_data krb5_data

Members

krb5_magic **krb5_data.magic**

unsigned int **krb5_data.length**

char * **krb5_data.data**

krb5_deltat

krb5_deltat

Declaration

typedef krb5_int32 krb5_deltat

krb5_enc_data

krb5_enc_data

Declaration

typedef struct _krb5_enc_data krb5_enc_data

Members

krb5_magic **krb5_enc_data.magic**

krb5_enctype **krb5_enc_data.enctype**

krb5_kvno **krb5_enc_data.kvno**

krb5_data **krb5_enc_data.ciphertext**

krb5_enc_kdc_rep_part

krb5_enc_kdc_rep_part

C representation of *EncKDCRepPart* protocol message.

This is the cleartext message that is encrypted and inserted in *KDC-REP* .

Declaration

typedef struct _krb5_enc_kdc_rep_part krb5_enc_kdc_rep_part

Members

krb5_magic **krb5_enc_kdc_rep_part.magic**

krb5_msgtype **krb5_enc_kdc_rep_part.msg_type**
krb5 message type

krb5_keyblock * **krb5_enc_kdc_rep_part.session**
Session key.

krb5_last_req_entry ** **krb5_enc_kdc_rep_part.last_req**
Array of pointers to entries.

krb5_int32 **krb5_enc_kdc_rep_part.nonce**
Nonce from request.

krb5_timestamp **krb5_enc_kdc_rep_part.key_exp**
Expiration date.

krb5_flags **krb5_enc_kdc_rep_part.flags**
Ticket flags.

krb5_ticket_times **krb5_enc_kdc_rep_part.times**
Lifetime info.

krb5_principal **krb5_enc_kdc_rep_part.server**
Server's principal identifier.

krb5_address ** **krb5_enc_kdc_rep_part.caddrs**
Array of ptrs to addrs, optional.

krb5_pa_data ** **krb5_enc_kdc_rep_part.enc_padata**
Encrypted preauthentication data.

krb5_enc_tkt_part

krb5_enc_tkt_part

Encrypted part of ticket.

Declaration

typedef struct _krb5_enc_tkt_part krb5_enc_tkt_part

Members

krb5_magic **krb5_enc_tkt_part.magic**

krb5_flags **krb5_enc_tkt_part.flags**
flags

krb5_keyblock * **krb5_enc_tkt_part.session**
session key: includes enctype

krb5_principal **krb5_enc_tkt_part.client**
client name/realm

krb5_transited **krb5_enc_tkt_part.transited**
list of transited realms

krb5_ticket_times **krb5_enc_tkt_part.times**
auth, start, end, renew_till

krb5_address ** **krb5_enc_tkt_part.caddrs**
array of ptrs to addresses

krb5_authdata ** **krb5_enc_tkt_part.authorization_data**
auth data

krb5_encrypt_block

krb5_encrypt_block

Declaration

typedef struct _krb5_encrypt_block krb5_encrypt_block

Members

krb5_magic **krb5_encrypt_block.magic**

krb5_enctype **krb5_encrypt_block.crypto_entry**

krb5_keyblock * **krb5_encrypt_block.key**

krb5_enctype

krb5_enctype

Declaration

typedef krb5_int32 krb5_enctype

krb5_error

krb5_error

Error message structure.

Declaration

typedef struct _krb5_error krb5_error

Members

krb5_magic **krb5_error.magic**

krb5_timestamp **krb5_error.ctime**
Client sec portion; optional.

krb5_int32 **krb5_error.cusec**
Client usec portion; optional.

krb5_int32 **krb5_error.susec**
Server usec portion.

krb5_timestamp **krb5_error.stime**
Server sec portion.

krb5_ui_4 **krb5_error.error**
Error code (protocol error #'s)

krb5_principal **krb5_error.client**
Client principal and realm.

krb5_principal **krb5_error.server**
Server principal and realm.

krb5_data **krb5_error.text**
Descriptive text.

krb5_data **krb5_error.e_data**
Additional error-describing data.

krb5_error_code

krb5_error_code

Used to convey an operation status.

The value 0 indicates success; any other values are com_err codes. Use *krb5_get_error_message()* to obtain a string describing the error.

Declaration

typedef krb5_int32 krb5_error_code

krb5_expire_callback_func

krb5_expire_callback_func

Declaration

typedef void(* krb5_expire_callback_func) (krb5_context context, void *data, krb5_timestamp password_expiration, krb5_timestamp account_expiration, krb5_boolean is_last_req)

krb5_flags

krb5_flags

Declaration

typedef krb5_int32 krb5_flags

krb5_get_init_creds_opt

krb5_get_init_creds_opt

Store options for *_krb5_get_init_creds* .

Declaration

typedef struct _krb5_get_init_creds_opt krb5_get_init_creds_opt

Members

krb5_flags **krb5_get_init_creds_opt.flags**

krb5_deltat **krb5_get_init_creds_opt.tkt_life**

krb5_deltat **krb5_get_init_creds_opt.renew_life**

int **krb5_get_init_creds_opt.forwardable**

int **krb5_get_init_creds_opt.proxiable**

krb5_enctype * **krb5_get_init_creds_opt.etype_list**

int **krb5_get_init_creds_opt.etype_list_length**

krb5_address ** **krb5_get_init_creds_opt.address_list**

krb5_preauthtype * **krb5_get_init_creds_opt.preauth_list**

int **krb5_get_init_creds_opt.preauth_list_length**

krb5_data * **krb5_get_init_creds_opt.salt**

krb5_gic_opt_pa_data

krb5_gic_opt_pa_data

Generic preauth option attribute/value pairs.

Declaration

typedef struct _krb5_gic_opt_pa_data krb5_gic_opt_pa_data

Members

char * **krb5_gic_opt_pa_data.attr**

char * **krb5_gic_opt_pa_data.value**

krb5_int16

krb5_int16

Declaration

typedef int16_t krb5_int16

krb5_int32

krb5_int32

Declaration

typedef int32_t krb5_int32

krb5_kdc_rep

krb5_kdc_rep

Representation of the *KDC-REP* protocol message.

Declaration

typedef struct _krb5_kdc_rep krb5_kdc_rep

Members

krb5_magic **krb5_kdc_rep.magic**

krb5_msgtype **krb5_kdc_rep.msg_type**
KRB5_AS_REP or KRB5_KDC_REP.

krb5_pa_data ** **krb5_kdc_rep.padata**
Preauthentication data from KDC.

krb5_principal **krb5_kdc_rep.client**
Client principal and realm.

krb5_ticket * **krb5_kdc_rep.ticket**
Ticket.

krb5_enc_data **krb5_kdc_rep.enc_part**
Encrypted part of reply.

krb5_enc_kdc_rep_part * **krb5_kdc_rep.enc_part2**
Unencrypted version, if available.

krb5_kdc_req

krb5_kdc_req

C representation of KDC-REQ protocol message, including KDC-REQ-BODY.

Declaration

typedef struct _krb5_kdc_req krb5_kdc_req

Members

krb5_magic **krb5_kdc_req.magic**

krb5_msgtype **krb5_kdc_req.msg_type**
KRB5_AS_REQ or KRB5_TGS_REQ.

krb5_pa_data ** **krb5_kdc_req.padata**
Preauthentication data.

krb5_flags **krb5_kdc_req.kdc_options**
Requested options.

krb5_principal **krb5_kdc_req.client**
Client principal and realm.

krb5_principal **krb5_kdc_req.server**
Server principal and realm.

krb5_timestamp **krb5_kdc_req.from**
Requested start time.

krb5_timestamp **krb5_kdc_req.till**
Requested end time.

krb5_timestamp **krb5_kdc_req.rtime**
Requested renewable end time.

krb5_int32 **krb5_kdc_req.nonce**
Nonce to match request and response.

int **krb5_kdc_req.nktypes**
Number of enctypes.

krb5_enctype * **krb5_kdc_req.ktype**
Requested enctypes.

krb5_address ** **krb5_kdc_req.addresses**
Requested addresses (optional)

krb5_enc_data **krb5_kdc_req.authorization_data**
Encrypted authz data (optional)

krb5_authdata ** **krb5_kdc_req.unenc_authdata**
Unencrypted authz data.

krb5_ticket ** **krb5_kdc_req.second_ticket**
Second ticket array (optional)

krb5_keyblock

krb5_keyblock

Exposed contents of a key.

Declaration

typedef struct _krb5_keyblock krb5_keyblock

Members

krb5_magic **krb5_keyblock.magic**

krb5_enctype **krb5_keyblock.enctype**

unsigned int **krb5_keyblock.length**

krb5_octet * **krb5_keyblock.contents**

krb5_keytab_entry

krb5_keytab_entry

A key table entry.

Declaration

typedef struct krb5_keytab_entry_st krb5_keytab_entry

Members

krb5_magic **krb5_keytab_entry.magic**

krb5_principal **krb5_keytab_entry.principal**
Principal of this key.

krb5_timestamp **krb5_keytab_entry.timestamp**
Time entry written to keytable.

krb5_kvno **krb5_keytab_entry.vno**
Key version number.

krb5_keyblock **krb5_keytab_entry.key**
The secret key.

krb5_keyusage

krb5_keyusage

Declaration

typedef krb5_int32 krb5_keyusage

krb5_kt_cursor

krb5_kt_cursor

Declaration

typedef krb5_pointer krb5_kt_cursor

krb5_kvno

krb5_kvno

Declaration

typedef unsigned int krb5_kvno

krb5_last_req_entry

krb5_last_req_entry

Last request entry.

Declaration

typedef struct _krb5_last_req_entry krb5_last_req_entry

Members

krb5_magic **krb5_last_req_entry.magic**

krb5_int32 **krb5_last_req_entry.lr_type**
LR type.

krb5_timestamp **krb5_last_req_entry.value**
Timestamp.

krb5_magic

krb5_magic

Declaration

typedef krb5_error_code krb5_magic

krb5_mk_req_checksum_func

krb5_mk_req_checksum_func

Type of function used as a callback to generate checksum data for mk_req.

Declaration

typedef krb5_error_code(* krb5_mk_req_checksum_func) (krb5_context, krb5_auth_context, void *, krb5_data **)

krb5_msgtype

krb5_msgtype

Declaration

typedef unsigned int krb5_msgtype

krb5_octet

krb5_octet

Declaration

typedef uint8_t krb5_octet

krb5_pa_pac_req

krb5_pa_pac_req

Declaration

typedef struct _krb5_pa_pac_req krb5_pa_pac_req

Members

krb5_boolean **krb5_pa_pac_req.include_pac**
TRUE if a PAC should be included in TGS-REP.

krb5_pa_server_referral_data

krb5_pa_server_referral_data

Declaration

typedef struct _krb5_pa_server_referral_data krb5_pa_server_referral_data

Members

krb5_data * **krb5_pa_server_referral_data.referred_realm**

krb5_principal **krb5_pa_server_referral_data.true_principal_name**

krb5_principal **krb5_pa_server_referral_data.requested_principal_name**

krb5_timestamp **krb5_pa_server_referral_data.referral_valid_until**

krb5_checksum **krb5_pa_server_referral_data.rep_cksum**

krb5_pa_svr_referral_data

krb5_pa_svr_referral_data

Declaration

typedef struct _krb5_pa_svr_referral_data krb5_pa_svr_referral_data

Members

krb5_principal **krb5_pa_svr_referral_data.principal**
Referred name, only realm is required.

krb5_pa_data

krb5_pa_data

Pre-authentication data.

Declaration

typedef struct _krb5_pa_data krb5_pa_data

Members

krb5_magic **krb5_pa_data.magic**

krb5_preauthtype **krb5_pa_data.pa_type**
Preauthentication data type.

unsigned int **krb5_pa_data.length**
Length of data.

krb5_octet * **krb5_pa_data.contents**
Data.

krb5_pointer

krb5_pointer

Declaration

typedef void* krb5_pointer

krb5_post_recv_fn

krb5_post_recv_fn

Hook function for inspecting or overriding KDC replies.

If *code* is non-zero, KDC communication failed and *reply* should be ignored. The hook function may return *code* or a different error code, or may synthesize a reply by setting *new_reply_out* and return successfully. The hook function should use *krb5_copy_data()* to construct the value for *new_reply_out* , to ensure that it can be freed correctly by the library.

Declaration

typedef krb5_error_code(* krb5_post_recv_fn) (krb5_context context, void *data, krb5_error_code code, const krb5_data *realm, const krb5_data *message, const krb5_data *reply, krb5_data **new_reply_out)

krb5_pre_send_fn

krb5_pre_send_fn

Hook function for inspecting or modifying messages sent to KDCs.

If the hook function sets *reply_out* , *message* will not be sent to the KDC, and the given reply will used instead. If the hook function sets *new_message_out* , the given message will be sent to the KDC in place of *message* . If the hook function returns successfully without setting either output, *message* will be sent to the KDC normally. The hook function should use *krb5_copy_data()* to construct the value for *new_message_out* or *reply_out* , to ensure that it can be freed correctly by the library.

Declaration

typedef krb5_error_code(* krb5_pre_send_fn) (krb5_context context, void *data, const krb5_data *realm, const krb5_data *message, krb5_data **new_message_out, krb5_data **new_reply_out)

krb5_preauthtype

krb5_preauthtype

Declaration

typedef krb5_int32 krb5_preauthtype

krb5_principal

krb5_principal

Declaration

typedef krb5_principal_data* krb5_principal

Members

krb5_magic **krb5_principal.magic**

krb5_data **krb5_principal.realm**

krb5_data * **krb5_principal.data**
An array of strings.

krb5_int32 **krb5_principal.length**

krb5_int32 **krb5_principal.type**

krb5_principal_data

krb5_principal_data

Declaration

typedef struct krb5_principal_data krb5_principal_data

Members

krb5_magic **krb5_principal_data.magic**

krb5_data **krb5_principal_data.realm**

krb5_data * **krb5_principal_data.data**
An array of strings.

krb5_int32 **krb5_principal_data.length**

krb5_int32 **krb5_principal_data.type**

krb5_prompt

krb5_prompt

Text for prompt used in prompter callback function.

Declaration

typedef struct _krb5_prompt krb5_prompt

Members

char * **krb5_prompt.prompt**
The prompt to show to the user.

int **krb5_prompt.hidden**
Boolean; informative prompt or hidden (e.g. PIN)

krb5_data * **krb5_prompt.reply**
Must be allocated before call to prompt routine.

krb5_prompt_type

krb5_prompt_type

Declaration

typedef krb5_int32 krb5_prompt_type

krb5_prompter_fct

krb5_prompter_fct

Pointer to a prompter callback function.

Declaration

typedef krb5_error_code(* krb5_prompter_fct) (krb5_context context, void *data, const char *name, const char *banner, int num_prompts, krb5_prompt prompts[])

krb5_pwd_data

krb5_pwd_data

Declaration

typedef struct _krb5_pwd_data krb5_pwd_data

Members

krb5_magic **krb5_pwd_data.magic**

int **krb5_pwd_data.sequence_count**

passwd_phrase_element ** **krb5_pwd_data.element**

krb5_responder_context

krb5_responder_context

A container for a set of preauthentication questions and answers.

A responder context is supplied by the krb5 authentication system to a *krb5_responder_fn* callback. It contains a list of questions and can receive answers. Questions contained in a responder context can be listed using *krb5_responder_list_questions()* , retrieved using *krb5_responder_get_challenge()* , or answered using *krb5_responder_set_answer()* . The form of a question's challenge and answer depend on the question name.

Declaration

typedef struct krb5_responder_context_st* krb5_responder_context

krb5_responder_fn

krb5_responder_fn

Responder function for an initial credential exchange.

If a required question is unanswered, the prompter may be called.

Declaration

typedef krb5_error_code(* krb5_responder_fn) (krb5_context ctx, void *data, krb5_responder_context rctx)

krb5_responder_otp_challenge

krb5_responder_otp_challenge

Declaration

typedef struct _krb5_responder_otp_challenge krb5_responder_otp_challenge

Members

char * **krb5_responder_otp_challenge.service**

krb5_responder_otp_tokeninfo ** **krb5_responder_otp_challenge.tokeninfo**

krb5_responder_otp_tokeninfo

krb5_responder_otp_tokeninfo

Declaration

typedef struct _krb5_responder_otp_tokeninfo krb5_responder_otp_tokeninfo

Members

krb5_flags **krb5_responder_otp_tokeninfo.flags**

krb5_int32 **krb5_responder_otp_tokeninfo.format**

krb5_int32 **krb5_responder_otp_tokeninfo.length**

char * **krb5_responder_otp_tokeninfo.vendor**

char * **krb5_responder_otp_tokeninfo.challenge**

char * **krb5_responder_otp_tokeninfo.token_id**

char * **krb5_responder_otp_tokeninfo.alg_id**

krb5_responder_pkinit_challenge

krb5_responder_pkinit_challenge

Declaration

typedef struct _krb5_responder_pkinit_challenge krb5_responder_pkinit_challenge

Members

krb5_responder_pkinit_identity ** **krb5_responder_pkinit_challenge.identities**

krb5_responder_pkinit_identity

krb5_responder_pkinit_identity

Declaration

typedef struct _krb5_responder_pkinit_identity krb5_responder_pkinit_identity

Members

char * **krb5_responder_pkinit_identity.identity**

krb5_int32 **krb5_responder_pkinit_identity.token_flags**

krb5_response

krb5_response

Declaration

typedef struct _krb5_response krb5_response

Members

krb5_magic **krb5_response.magic**

krb5_octet **krb5_response.message_type**

krb5_data **krb5_response.response**

krb5_int32 **krb5_response.expected_nonce**

krb5_timestamp **krb5_response.request_time**

krb5_replay_data

krb5_replay_data

Replay data.

Sequence number and timestamp information output by *krb5_rd_priv()* and *krb5_rd_safe()* .

Declaration

typedef struct krb5_replay_data krb5_replay_data

Members

krb5_timestamp **krb5_replay_data.timestamp**
Timestamp, seconds portion.

krb5_int32 **krb5_replay_data.usec**
Timestamp, microseconds portion.

krb5_ui_4 **krb5_replay_data.seq**
Sequence number.

krb5_ticket

krb5_ticket

Ticket structure.

The C representation of the ticket message, with a pointer to the C representation of the encrypted part.

Declaration

typedef struct _krb5_ticket krb5_ticket

Members

krb5_magic **krb5_ticket.magic**

krb5_principal **krb5_ticket.server**
server name/realm

krb5_enc_data **krb5_ticket.enc_part**
encryption type, kvno, encrypted encoding

krb5_enc_tkt_part * **krb5_ticket.enc_part2**
ptr to decrypted version, if available

krb5_ticket_times

krb5_ticket_times

Ticket start time, end time, and renewal duration.

Declaration

typedef struct _krb5_ticket_times krb5_ticket_times

Members

krb5_timestamp **krb5_ticket_times.authtime**
Time at which KDC issued the initial ticket that corresponds to this ticket.

krb5_timestamp **krb5_ticket_times.starttime**
optional in ticket, if not present, use *authtime*

krb5_timestamp **krb5_ticket_times.endtime**
Ticket expiration time.

krb5_timestamp **krb5_ticket_times.renew_till**
Latest time at which renewal of ticket can be valid.

krb5_timestamp

krb5_timestamp

Represents a timestamp in seconds since the POSIX epoch.

This legacy type is used frequently in the ABI, but cannot represent timestamps after 2038 as a positive number. Code which uses this type should cast values of it to uint32_t so that negative values are treated as timestamps between 2038 and 2106 on platforms with 64-bit time_t.

Declaration

typedef krb5_int32 krb5_timestamp

krb5_tkt_authent

krb5_tkt_authent

Ticket authentication data.

Declaration

typedef struct _krb5_tkt_authent krb5_tkt_authent

Members

krb5_magic **krb5_tkt_authent.magic**

krb5_ticket * **krb5_tkt_authent.ticket**

krb5_authenticator * **krb5_tkt_authent.authenticator**

krb5_flags **krb5_tkt_authent.ap_options**

krb5_trace_callback

krb5_trace_callback

Declaration

typedef void(* krb5_trace_callback) (krb5_context context, const krb5_trace_info *info, void *cb_data)

krb5_trace_info

krb5_trace_info

A wrapper for passing information to a *krb5_trace_callback* .

Currently, it only contains the formatted message as determined the the format string and arguments of the tracing macro, but it may be extended to contain more fields in the future.

Declaration

typedef struct _krb5_trace_info krb5_trace_info

Members

const char * **krb5_trace_info.message**

krb5_transited

krb5_transited

Structure for transited encoding.

Declaration

typedef struct _krb5_transited krb5_transited

Members

krb5_magic **krb5_transited.magic**

krb5_octet **krb5_transited.tr_type**
Transited encoding type.

krb5_data **krb5_transited.tr_contents**
Contents.

krb5_typed_data

krb5_typed_data

Declaration

typedef struct _krb5_typed_data krb5_typed_data

Members

krb5_magic **krb5_typed_data.magic**

krb5_int32 **krb5_typed_data.type**

unsigned int **krb5_typed_data.length**

krb5_octet * **krb5_typed_data.data**

krb5_ui_2

krb5_ui_2

Declaration

typedef uint16_t krb5_ui_2

krb5_ui_4

krb5_ui_4

Declaration

typedef uint32_t krb5_ui_4

krb5_verify_init_creds_opt

krb5_verify_init_creds_opt

Declaration

typedef struct _krb5_verify_init_creds_opt krb5_verify_init_creds_opt

Members

krb5_flags **krb5_verify_init_creds_opt.flags**

int **krb5_verify_init_creds_opt.ap_req_nofail**
boolean

passwd_phrase_element

passwd_phrase_element

Declaration

typedef struct _passwd_phrase_element passwd_phrase_element

Members

krb5_magic **passwd_phrase_element.magic**

krb5_data * **passwd_phrase_element.passwd**

krb5_data * **passwd_phrase_element.phrase**

6.2.2 Internal

krb5_auth_context

krb5_auth_context

Declaration

typedef struct _krb5_auth_context* krb5_auth_context

krb5_cksumtype

krb5_cksumtype

Declaration

typedef krb5_int32 krb5_cksumtype

krb5_context

krb5_context

Declaration

typedef struct _krb5_context* krb5_context

krb5_cc_cursor

krb5_cc_cursor

Cursor for sequential lookup.

Declaration

typedef krb5_pointer krb5_cc_cursor

krb5_ccache

krb5_ccache

Declaration

typedef struct _krb5_ccache* krb5_ccache

krb5_cccol_cursor

krb5_cccol_cursor

Cursor for iterating over all ccaches.

Declaration

typedef struct _krb5_cccol_cursor* krb5_cccol_cursor

krb5_init_creds_context

krb5_init_creds_context

Declaration

typedef struct _krb5_init_creds_context* krb5_init_creds_context

krb5_key

krb5_key

Opaque identifier for a key.

Use with the krb5_k APIs for better performance for repeated operations with the same key and usage. Key identifiers must not be used simultaneously within multiple threads, as they may contain mutable internal state and are not mutex-protected.

Declaration

typedef struct krb5_key_st* krb5_key

krb5_keytab

krb5_keytab

Declaration

typedef struct _krb5_kt* krb5_keytab

krb5_pac

krb5_pac

PAC data structure to convey authorization information.

Declaration

typedef struct krb5_pac_data* krb5_pac

krb5_rcache

krb5_rcache

Declaration

typedef struct krb5_rc_st* krb5_rcache

krb5_tkt_creds_context

krb5_tkt_creds_context

Declaration

typedef struct _krb5_tkt_creds_context* krb5_tkt_creds_context

6.3 krb5 simple macros

6.3.1 Public

ADDRTYPE_ADDRPORT

ADDRTYPE_ADDRPORT

ADDRTYPE_ADDRPORT	0x0100

ADDRTYPE_CHAOS

ADDRTYPE_CHAOS

ADDRTYPE_CHAOS	0x0005

ADDRTYPE_DDP

ADDRTYPE_DDP

ADDRTYPE_DDP	0x0010

ADDRTYPE_INET

ADDRTYPE_INET

ADDRTYPE_INET	0x0002

ADDRTYPE_INET6

ADDRTYPE_INET6

ADDRTYPE_INET6	0x0018

ADDRTYPE_IPPORT

ADDRTYPE_IPPORT

ADDRTYPE_IPPORT	0x0101

ADDRTYPE_ISO

ADDRTYPE_ISO

ADDRTYPE_ISO	0x0007

ADDRTYPE_IS_LOCAL

ADDRTYPE_IS_LOCAL

ADDRTYPE_IS_LOCAL (addrtype)	(addrtype & 0x8000)

ADDRTYPE_NETBIOS

ADDRTYPE_NETBIOS

ADDRTYPE_NETBIOS	0x0014

ADDRTYPE_XNS

ADDRTYPE_XNS

ADDRTYPE_XNS	0x0006

AD_TYPE_EXTERNAL

AD_TYPE_EXTERNAL

AD_TYPE_EXTERNAL	0x4000

AD_TYPE_FIELD_TYPE_MASK

AD_TYPE_FIELD_TYPE_MASK

AD_TYPE_FIELD_TYPE_MASK	0x1fff

AD_TYPE_REGISTERED

AD_TYPE_REGISTERED

AD_TYPE_REGISTERED	0x2000

AD_TYPE_RESERVED

AD_TYPE_RESERVED

AD_TYPE_RESERVED	0x8000

AP_OPTS_ETYPE_NEGOTIATION

AP_OPTS_ETYPE_NEGOTIATION

AP_OPTS_ETYPE_NEGOTIATION	0x00000002

AP_OPTS_MUTUAL_REQUIRED

AP_OPTS_MUTUAL_REQUIRED

Perform a mutual authentication exchange.

AP_OPTS_MUTUAL_REQUIRED	0x20000000

AP_OPTS_RESERVED

AP_OPTS_RESERVED

AP_OPTS_RESERVED	0x80000000

AP_OPTS_USE_SESSION_KEY

AP_OPTS_USE_SESSION_KEY

Use session key.

AP_OPTS_USE_SESSION_KEY	0x40000000

AP_OPTS_USE_SUBKEY

AP_OPTS_USE_SUBKEY

Generate a subsession key from the current session key obtained from the credentials.

AP_OPTS_USE_SUBKEY	0x00000001

AP_OPTS_WIRE_MASK

AP_OPTS_WIRE_MASK

AP_OPTS_WIRE_MASK	0xfffffff0

CKSUMTYPE_CMAC_CAMELLIA128

CKSUMTYPE_CMAC_CAMELLIA128

RFC 6803.

CKSUMTYPE_CMAC_CAMELLIA128	0x0011

CKSUMTYPE_CMAC_CAMELLIA256

CKSUMTYPE_CMAC_CAMELLIA256

RFC 6803.

CKSUMTYPE_CMAC_CAMELLIA256	0x0012

CKSUMTYPE_CRC32

CKSUMTYPE_CRC32

CKSUMTYPE_CRC32	0x0001

CKSUMTYPE_DESCBC

CKSUMTYPE_DESCBC

CKSUMTYPE_DESCBC	0x0004

CKSUMTYPE_HMAC_MD5_ARCFOUR

CKSUMTYPE_HMAC_MD5_ARCFOUR

RFC 4757.

CKSUMTYPE_HMAC_MD5_ARCFOUR	-138

CKSUMTYPE_HMAC_SHA1_96_AES128

CKSUMTYPE_HMAC_SHA1_96_AES128

RFC 3962.

Used with ENCTYPE_AES128_CTS_HMAC_SHA1_96

CKSUMTYPE_HMAC_SHA1_96_AES128	0x000f

CKSUMTYPE_HMAC_SHA1_96_AES256

CKSUMTYPE_HMAC_SHA1_96_AES256

RFC 3962.

Used with ENCTYPE_AES256_CTS_HMAC_SHA1_96

CKSUMTYPE_HMAC_SHA1_96_AES256	0x0010

CKSUMTYPE_HMAC_SHA256_128_AES128

CKSUMTYPE_HMAC_SHA256_128_AES128

RFC 8009.

CKSUMTYPE_HMAC_SHA256_128_AES128	0x0013

CKSUMTYPE_HMAC_SHA384_192_AES256

CKSUMTYPE_HMAC_SHA384_192_AES256

RFC 8009.

CKSUMTYPE_HMAC_SHA384_192_AES256	0x0014

CKSUMTYPE_HMAC_SHA1_DES3

CKSUMTYPE_HMAC_SHA1_DES3

CKSUMTYPE_HMAC_SHA1_DES3	0x000c

CKSUMTYPE_MD5_HMAC_ARCFOUR

CKSUMTYPE_MD5_HMAC_ARCFOUR

CKSUMTYPE_MD5_HMAC_ARCFOUR	-137 /* Microsoft netlogon */

CKSUMTYPE_NIST_SHA

CKSUMTYPE_NIST_SHA

CKSUMTYPE_NIST_SHA	0x0009

CKSUMTYPE_RSA_MD4

CKSUMTYPE_RSA_MD4

CKSUMTYPE_RSA_MD4	0x0002

CKSUMTYPE_RSA_MD4_DES

CKSUMTYPE_RSA_MD4_DES

CKSUMTYPE_RSA_MD4_DES	0x0003

CKSUMTYPE_RSA_MD5

CKSUMTYPE_RSA_MD5

CKSUMTYPE_RSA_MD5	0x0007

CKSUMTYPE_RSA_MD5_DES

CKSUMTYPE_RSA_MD5_DES

CKSUMTYPE_RSA_MD5_DES	0x0008

ENCTYPE_AES128_CTS_HMAC_SHA1_96

ENCTYPE_AES128_CTS_HMAC_SHA1_96

RFC 3962.

ENCTYPE_AES128_CTS_HMAC_SHA1_96	0x0011

ENCTYPE_AES128_CTS_HMAC_SHA256_128

ENCTYPE_AES128_CTS_HMAC_SHA256_128

RFC 8009.

ENCTYPE_AES128_CTS_HMAC_SHA256_128	0x0013

ENCTYPE_AES256_CTS_HMAC_SHA1_96

ENCTYPE_AES256_CTS_HMAC_SHA1_96

RFC 3962.

ENCTYPE_AES256_CTS_HMAC_SHA1_96	0x0012

ENCTYPE_AES256_CTS_HMAC_SHA384_192

ENCTYPE_AES256_CTS_HMAC_SHA384_192

RFC 8009.

ENCTYPE_AES256_CTS_HMAC_SHA384_192	0x0014

ENCTYPE_ARCFOUR_HMAC

ENCTYPE_ARCFOUR_HMAC

RFC 4757.

ENCTYPE_ARCFOUR_HMAC	0x0017

ENCTYPE_ARCFOUR_HMAC_EXP

ENCTYPE_ARCFOUR_HMAC_EXP

RFC 4757.

ENCTYPE_ARCFOUR_HMAC_EXP	0x0018

ENCTYPE_CAMELLIA128_CTS_CMAC

ENCTYPE_CAMELLIA128_CTS_CMAC

RFC 6803.

ENCTYPE_CAMELLIA128_CTS_CMAC	0x0019

ENCTYPE_CAMELLIA256_CTS_CMAC

ENCTYPE_CAMELLIA256_CTS_CMAC

RFC 6803.

ENCTYPE_CAMELLIA256_CTS_CMAC	0x001a

ENCTYPE_DES3_CBC_ENV

ENCTYPE_DES3_CBC_ENV

DES-3 cbc mode, CMS enveloped data.

ENCTYPE_DES3_CBC_ENV	0x000f

ENCTYPE_DES3_CBC_RAW

ENCTYPE_DES3_CBC_RAW

ENCTYPE_DES3_CBC_RAW	0x0006

ENCTYPE_DES3_CBC_SHA

ENCTYPE_DES3_CBC_SHA

ENCTYPE_DES3_CBC_SHA	0x0005

ENCTYPE_DES3_CBC_SHA1

ENCTYPE_DES3_CBC_SHA1

ENCTYPE_DES3_CBC_SHA1	0x0010

ENCTYPE_DES_CBC_CRC

ENCTYPE_DES_CBC_CRC

ENCTYPE_DES_CBC_CRC	0x0001

ENCTYPE_DES_CBC_MD4

ENCTYPE_DES_CBC_MD4

ENCTYPE_DES_CBC_MD4	0x0002

ENCTYPE_DES_CBC_MD5

ENCTYPE_DES_CBC_MD5

ENCTYPE_DES_CBC_MD5	0x0003

ENCTYPE_DES_CBC_RAW

ENCTYPE_DES_CBC_RAW

ENCTYPE_DES_CBC_RAW	0x0004

ENCTYPE_DES_HMAC_SHA1

ENCTYPE_DES_HMAC_SHA1

ENCTYPE_DES_HMAC_SHA1	0x0008

ENCTYPE_DSA_SHA1_CMS

ENCTYPE_DSA_SHA1_CMS

DSA with SHA1, CMS signature.

ENCTYPE_DSA_SHA1_CMS	0x0009

ENCTYPE_MD5_RSA_CMS

ENCTYPE_MD5_RSA_CMS

MD5 with RSA, CMS signature.

ENCTYPE_MD5_RSA_CMS	0x000a

ENCTYPE_NULL

ENCTYPE_NULL

ENCTYPE_NULL	0x0000

ENCTYPE_RC2_CBC_ENV

ENCTYPE_RC2_CBC_ENV

RC2 cbc mode, CMS enveloped data.

ENCTYPE_RC2_CBC_ENV	0x000c

ENCTYPE_RSA_ENV

ENCTYPE_RSA_ENV

RSA encryption, CMS enveloped data.

ENCTYPE_RSA_ENV	0x000d

ENCTYPE_RSA_ES_OAEP_ENV

ENCTYPE_RSA_ES_OAEP_ENV

RSA w/OEAP encryption, CMS enveloped data.

ENCTYPE_RSA_ES_OAEP_ENV	0x000e

ENCTYPE_SHA1_RSA_CMS

ENCTYPE_SHA1_RSA_CMS

SHA1 with RSA, CMS signature.

ENCTYPE_SHA1_RSA_CMS	0x000b

ENCTYPE_UNKNOWN

ENCTYPE_UNKNOWN

ENCTYPE_UNKNOWN	0x01ff

KDC_OPT_ALLOW_POSTDATE

KDC_OPT_ALLOW_POSTDATE

KDC_OPT_ALLOW_POSTDATE	0x04000000

KDC_OPT_CANONICALIZE

KDC_OPT_CANONICALIZE

KDC_OPT_CANONICALIZE	0x00010000

KDC_OPT_CNAME_IN_ADDL_TKT

KDC_OPT_CNAME_IN_ADDL_TKT

KDC_OPT_CNAME_IN_ADDL_TKT	0x00020000

KDC_OPT_DISABLE_TRANSITED_CHECK

KDC_OPT_DISABLE_TRANSITED_CHECK

KDC_OPT_DISABLE_TRANSITED_CHECK	0x00000020

KDC_OPT_ENC_TKT_IN_SKEY

KDC_OPT_ENC_TKT_IN_SKEY

KDC_OPT_ENC_TKT_IN_SKEY	0x00000008

KDC_OPT_FORWARDABLE

KDC_OPT_FORWARDABLE

KDC_OPT_FORWARDABLE	0x40000000

KDC_OPT_FORWARDED

KDC_OPT_FORWARDED

KDC_OPT_FORWARDED	0x20000000

KDC_OPT_POSTDATED

KDC_OPT_POSTDATED

KDC_OPT_POSTDATED	0x02000000

KDC_OPT_PROXIABLE

KDC_OPT_PROXIABLE

KDC_OPT_PROXIABLE	0x10000000

KDC_OPT_PROXY

KDC_OPT_PROXY

KDC_OPT_PROXY	0x08000000

KDC_OPT_RENEW

KDC_OPT_RENEW

KDC_OPT_RENEW	0x00000002

KDC_OPT_RENEWABLE

KDC_OPT_RENEWABLE

KDC_OPT_RENEWABLE	0x00800000

KDC_OPT_RENEWABLE_OK

KDC_OPT_RENEWABLE_OK

KDC_OPT_RENEWABLE_OK	0x00000010

KDC_OPT_REQUEST_ANONYMOUS

KDC_OPT_REQUEST_ANONYMOUS

KDC_OPT_REQUEST_ANONYMOUS	0x00008000

KDC_OPT_VALIDATE

KDC_OPT_VALIDATE

KDC_OPT_VALIDATE	0x00000001

KDC_TKT_COMMON_MASK

KDC_TKT_COMMON_MASK

KDC_TKT_COMMON_MASK	0x54800000

KRB5_ALTAUTH_ATT_CHALLENGE_RESPONSE

KRB5_ALTAUTH_ATT_CHALLENGE_RESPONSE

alternate authentication types

KRB5_ALTAUTH_ATT_CHALLENGE_RESPONSE	64

KRB5_ANONYMOUS_PRINCSTR

KRB5_ANONYMOUS_PRINCSTR

Anonymous principal name.

KRB5_ANONYMOUS_PRINCSTR	"ANONYMOUS"

KRB5_ANONYMOUS_REALMSTR

KRB5_ANONYMOUS_REALMSTR

Anonymous realm.

KRB5_ANONYMOUS_REALMSTR	"WELLKNOWN:ANONYMOUS"

KRB5_AP_REP

KRB5_AP_REP

Response to mutual AP request.

KRB5_AP_REP	((krb5_msgtype)15)

KRB5_AP_REQ

KRB5_AP_REQ

Auth req to application server.

KRB5_AP_REQ	((krb5_msgtype)14)

KRB5_AS_REP

KRB5_AS_REP

Response to AS request.

KRB5_AS_REP	((krb5_msgtype)11)

KRB5_AS_REQ

KRB5_AS_REQ

Initial authentication request.

KRB5_AS_REQ	((krb5_msgtype)10)

KRB5_AUTHDATA_AND_OR

KRB5_AUTHDATA_AND_OR

KRB5_AUTHDATA_AND_OR	5

KRB5_AUTHDATA_AUTH_INDICATOR

KRB5_AUTHDATA_AUTH_INDICATOR

KRB5_AUTHDATA_AUTH_INDICATOR	97

KRB5_AUTHDATA_CAMMAC

KRB5_AUTHDATA_CAMMAC

KRB5_AUTHDATA_CAMMAC	96

KRB5_AUTHDATA_ETYPE_NEGOTIATION

KRB5_AUTHDATA_ETYPE_NEGOTIATION

RFC 4537.

KRB5_AUTHDATA_ETYPE_NEGOTIATION	129

KRB5_AUTHDATA_FX_ARMOR

KRB5_AUTHDATA_FX_ARMOR

KRB5_AUTHDATA_FX_ARMOR	71

KRB5_AUTHDATA_IF_RELEVANT

KRB5_AUTHDATA_IF_RELEVANT

KRB5_AUTHDATA_IF_RELEVANT	1

KRB5_AUTHDATA_INITIAL_VERIFIED_CAS

KRB5_AUTHDATA_INITIAL_VERIFIED_CAS

KRB5_AUTHDATA_INITIAL_VERIFIED_CAS	9

KRB5_AUTHDATA_KDC_ISSUED

KRB5_AUTHDATA_KDC_ISSUED

KRB5_AUTHDATA_KDC_ISSUED	4

KRB5_AUTHDATA_MANDATORY_FOR_KDC

KRB5_AUTHDATA_MANDATORY_FOR_KDC

KRB5_AUTHDATA_MANDATORY_FOR_KDC	8

KRB5_AUTHDATA_OSF_DCE

KRB5_AUTHDATA_OSF_DCE

KRB5_AUTHDATA_OSF_DCE	64

KRB5_AUTHDATA_SESAME

KRB5_AUTHDATA_SESAME

KRB5_AUTHDATA_SESAME	65

KRB5_AUTHDATA_SIGNTICKET

KRB5_AUTHDATA_SIGNTICKET

formerly 142 in krb5 1.8

KRB5_AUTHDATA_SIGNTICKET	512

KRB5_AUTHDATA_WIN2K_PAC

KRB5_AUTHDATA_WIN2K_PAC

KRB5_AUTHDATA_WIN2K_PAC	128

KRB5_AUTH_CONTEXT_DO_SEQUENCE

KRB5_AUTH_CONTEXT_DO_SEQUENCE

Prevent replays with sequence numbers.

KRB5_AUTH_CONTEXT_DO_SEQUENCE	0x00000004

KRB5_AUTH_CONTEXT_DO_TIME

KRB5_AUTH_CONTEXT_DO_TIME

Prevent replays with timestamps and replay cache.

KRB5_AUTH_CONTEXT_DO_TIME	0x00000001

KRB5_AUTH_CONTEXT_GENERATE_LOCAL_ADDR

KRB5_AUTH_CONTEXT_GENERATE_LOCAL_ADDR

Generate the local network address.

KRB5_AUTH_CONTEXT_GENERATE_LOCAL_ADDR	0x00000001

KRB5_AUTH_CONTEXT_GENERATE_LOCAL_FULL_ADDR

KRB5_AUTH_CONTEXT_GENERATE_LOCAL_FULL_ADDR

Generate the local network address and the local port.

KRB5_AUTH_CONTEXT_GENERATE_LOCAL_FULL_ADDR	0x00000004

KRB5_AUTH_CONTEXT_GENERATE_REMOTE_ADDR

KRB5_AUTH_CONTEXT_GENERATE_REMOTE_ADDR

Generate the remote network address.

KRB5_AUTH_CONTEXT_GENERATE_REMOTE_ADDR	0x00000002

KRB5_AUTH_CONTEXT_GENERATE_REMOTE_FULL_ADDR

KRB5_AUTH_CONTEXT_GENERATE_REMOTE_FULL_ADDR

Generate the remote network address and the remote port.

KRB5_AUTH_CONTEXT_GENERATE_REMOTE_FULL_ADDR	0x00000008

KRB5_AUTH_CONTEXT_PERMIT_ALL

KRB5_AUTH_CONTEXT_PERMIT_ALL

KRB5_AUTH_CONTEXT_PERMIT_ALL	0x00000010

KRB5_AUTH_CONTEXT_RET_SEQUENCE

KRB5_AUTH_CONTEXT_RET_SEQUENCE

Save sequence numbers for application.

KRB5_AUTH_CONTEXT_RET_SEQUENCE	0x00000008

KRB5_AUTH_CONTEXT_RET_TIME

KRB5_AUTH_CONTEXT_RET_TIME

Save timestamps for application.

KRB5_AUTH_CONTEXT_RET_TIME	0x00000002

KRB5_AUTH_CONTEXT_USE_SUBKEY

KRB5_AUTH_CONTEXT_USE_SUBKEY

KRB5_AUTH_CONTEXT_USE_SUBKEY	0x00000020

KRB5_CRED

KRB5_CRED

Cred forwarding message.

KRB5_CRED	((krb5_msgtype)22)

KRB5_CRYPTO_TYPE_CHECKSUM

KRB5_CRYPTO_TYPE_CHECKSUM

[out] checksum for MIC

KRB5_CRYPTO_TYPE_CHECKSUM	6

KRB5_CRYPTO_TYPE_DATA

KRB5_CRYPTO_TYPE_DATA

[in, out] plaintext

KRB5_CRYPTO_TYPE_DATA	2

KRB5_CRYPTO_TYPE_EMPTY

KRB5_CRYPTO_TYPE_EMPTY

[in] ignored

KRB5_CRYPTO_TYPE_EMPTY	0

KRB5_CRYPTO_TYPE_HEADER

KRB5_CRYPTO_TYPE_HEADER

[out] header

KRB5_CRYPTO_TYPE_HEADER	1

KRB5_CRYPTO_TYPE_PADDING

KRB5_CRYPTO_TYPE_PADDING

[out] padding

KRB5_CRYPTO_TYPE_PADDING	4

KRB5_CRYPTO_TYPE_SIGN_ONLY

KRB5_CRYPTO_TYPE_SIGN_ONLY

[in] associated data

KRB5_CRYPTO_TYPE_SIGN_ONLY	3

KRB5_CRYPTO_TYPE_STREAM

KRB5_CRYPTO_TYPE_STREAM

[in] entire message without decomposing the structure into header, data and trailer buffers

KRB5_CRYPTO_TYPE_STREAM	7

KRB5_CRYPTO_TYPE_TRAILER

KRB5_CRYPTO_TYPE_TRAILER

[out] checksum for encrypt

KRB5_CRYPTO_TYPE_TRAILER	5

KRB5_CYBERSAFE_SECUREID

KRB5_CYBERSAFE_SECUREID

Cybersafe.

RFC 4120

KRB5_CYBERSAFE_SECUREID	9

KRB5_DOMAIN_X500_COMPRESS

KRB5_DOMAIN_X500_COMPRESS

Transited encoding types.

KRB5_DOMAIN_X500_COMPRESS	1

KRB5_ENCPADATA_REQ_ENC_PA_REP

KRB5_ENCPADATA_REQ_ENC_PA_REP

RFC 6806.

KRB5_ENCPADATA_REQ_ENC_PA_REP	149

KRB5_ERROR

KRB5_ERROR

Error response.

KRB5_ERROR	((krb5_msgtype)30)

KRB5_FAST_REQUIRED

KRB5_FAST_REQUIRED

Require KDC to support FAST.

KRB5_FAST_REQUIRED	0x0001

KRB5_GC_CACHED

KRB5_GC_CACHED

Want cached ticket only.

KRB5_GC_CACHED	2

KRB5_GC_CANONICALIZE

KRB5_GC_CANONICALIZE

Set canonicalize KDC option.

KRB5_GC_CANONICALIZE	4

KRB5_GC_CONSTRAINED_DELEGATION

KRB5_GC_CONSTRAINED_DELEGATION

Constrained delegation.

KRB5_GC_CONSTRAINED_DELEGATION	64

KRB5_GC_FORWARDABLE

KRB5_GC_FORWARDABLE

Acquire forwardable tickets.

KRB5_GC_FORWARDABLE	16

KRB5_GC_NO_STORE

KRB5_GC_NO_STORE

Do not store in credential cache.

KRB5_GC_NO_STORE	8

KRB5_GC_NO_TRANSIT_CHECK

KRB5_GC_NO_TRANSIT_CHECK

Disable transited check.

KRB5_GC_NO_TRANSIT_CHECK	32

KRB5_GC_USER_USER

KRB5_GC_USER_USER

Want user-user ticket.

KRB5_GC_USER_USER	1

KRB5_GET_INIT_CREDS_OPT_ADDRESS_LIST

KRB5_GET_INIT_CREDS_OPT_ADDRESS_LIST

KRB5_GET_INIT_CREDS_OPT_ADDRESS_LIST	0x0020

KRB5_GET_INIT_CREDS_OPT_ANONYMOUS

KRB5_GET_INIT_CREDS_OPT_ANONYMOUS

KRB5_GET_INIT_CREDS_OPT_ANONYMOUS	0x0400

KRB5_GET_INIT_CREDS_OPT_CANONICALIZE

KRB5_GET_INIT_CREDS_OPT_CANONICALIZE

KRB5_GET_INIT_CREDS_OPT_CANONICALIZE	0x0200

KRB5_GET_INIT_CREDS_OPT_CHG_PWD_PRMPT

KRB5_GET_INIT_CREDS_OPT_CHG_PWD_PRMPT

KRB5_GET_INIT_CREDS_OPT_CHG_PWD_PRMPT	0x0100

KRB5_GET_INIT_CREDS_OPT_ETYPE_LIST

KRB5_GET_INIT_CREDS_OPT_ETYPE_LIST

KRB5_GET_INIT_CREDS_OPT_ETYPE_LIST	0x0010

KRB5_GET_INIT_CREDS_OPT_FORWARDABLE

KRB5_GET_INIT_CREDS_OPT_FORWARDABLE

KRB5_GET_INIT_CREDS_OPT_FORWARDABLE	0x0004

KRB5_GET_INIT_CREDS_OPT_PREAUTH_LIST

KRB5_GET_INIT_CREDS_OPT_PREAUTH_LIST

KRB5_GET_INIT_CREDS_OPT_PREAUTH_LIST	0x0040

KRB5_GET_INIT_CREDS_OPT_PROXIABLE

KRB5_GET_INIT_CREDS_OPT_PROXIABLE

KRB5_GET_INIT_CREDS_OPT_PROXIABLE	0x0008

KRB5_GET_INIT_CREDS_OPT_RENEW_LIFE

KRB5_GET_INIT_CREDS_OPT_RENEW_LIFE

KRB5_GET_INIT_CREDS_OPT_RENEW_LIFE	0x0002

KRB5_GET_INIT_CREDS_OPT_SALT

KRB5_GET_INIT_CREDS_OPT_SALT

KRB5_GET_INIT_CREDS_OPT_SALT	0x0080

KRB5_GET_INIT_CREDS_OPT_TKT_LIFE

KRB5_GET_INIT_CREDS_OPT_TKT_LIFE

KRB5_GET_INIT_CREDS_OPT_TKT_LIFE	0x0001

KRB5_INIT_CONTEXT_SECURE

KRB5_INIT_CONTEXT_SECURE

Use secure context configuration.

KRB5_INIT_CONTEXT_SECURE	0x1

KRB5_INIT_CONTEXT_KDC

KRB5_INIT_CONTEXT_KDC

Use KDC configuration if available.

KRB5_INIT_CONTEXT_KDC	0x2

KRB5_INIT_CREDS_STEP_FLAG_CONTINUE

KRB5_INIT_CREDS_STEP_FLAG_CONTINUE

More responses needed.

KRB5_INIT_CREDS_STEP_FLAG_CONTINUE	0x1

KRB5_INT16_MAX

KRB5_INT16_MAX

KRB5_INT16_MAX	65535

KRB5_INT16_MIN

KRB5_INT16_MIN

KRB5_INT16_MIN	(-KRB5_INT16_MAX-1)

KRB5_INT32_MAX

KRB5_INT32_MAX

KRB5_INT32_MAX	2147483647

KRB5_INT32_MIN

KRB5_INT32_MIN

KRB5_INT32_MIN	(-KRB5_INT32_MAX-1)

KRB5_KEYUSAGE_AD_ITE

KRB5_KEYUSAGE_AD_ITE

KRB5_KEYUSAGE_AD_ITE	21

KRB5_KEYUSAGE_AD_KDCISSUED_CKSUM

KRB5_KEYUSAGE_AD_KDCISSUED_CKSUM

KRB5_KEYUSAGE_AD_KDCISSUED_CKSUM	19

KRB5_KEYUSAGE_AD_MTE

KRB5_KEYUSAGE_AD_MTE

KRB5_KEYUSAGE_AD_MTE	20

KRB5_KEYUSAGE_AD_SIGNEDPATH

KRB5_KEYUSAGE_AD_SIGNEDPATH

KRB5_KEYUSAGE_AD_SIGNEDPATH	-21

KRB5_KEYUSAGE_APP_DATA_CKSUM

KRB5_KEYUSAGE_APP_DATA_CKSUM

KRB5_KEYUSAGE_APP_DATA_CKSUM	17

KRB5_KEYUSAGE_APP_DATA_ENCRYPT

KRB5_KEYUSAGE_APP_DATA_ENCRYPT

KRB5_KEYUSAGE_APP_DATA_ENCRYPT	16

KRB5_KEYUSAGE_AP_REP_ENCPART

KRB5_KEYUSAGE_AP_REP_ENCPART

KRB5_KEYUSAGE_AP_REP_ENCPART	12

KRB5_KEYUSAGE_AP_REQ_AUTH

KRB5_KEYUSAGE_AP_REQ_AUTH

KRB5_KEYUSAGE_AP_REQ_AUTH	11

KRB5_KEYUSAGE_AP_REQ_AUTH_CKSUM

KRB5_KEYUSAGE_AP_REQ_AUTH_CKSUM

KRB5_KEYUSAGE_AP_REQ_AUTH_CKSUM	10

KRB5_KEYUSAGE_AS_REP_ENCPART

KRB5_KEYUSAGE_AS_REP_ENCPART

KRB5_KEYUSAGE_AS_REP_ENCPART	3

KRB5_KEYUSAGE_AS_REQ

KRB5_KEYUSAGE_AS_REQ

KRB5_KEYUSAGE_AS_REQ	56

KRB5_KEYUSAGE_AS_REQ_PA_ENC_TS

KRB5_KEYUSAGE_AS_REQ_PA_ENC_TS

KRB5_KEYUSAGE_AS_REQ_PA_ENC_TS	1

KRB5_KEYUSAGE_CAMMAC

KRB5_KEYUSAGE_CAMMAC

KRB5_KEYUSAGE_CAMMAC	64

KRB5_KEYUSAGE_ENC_CHALLENGE_CLIENT

KRB5_KEYUSAGE_ENC_CHALLENGE_CLIENT

KRB5_KEYUSAGE_ENC_CHALLENGE_CLIENT	54

KRB5_KEYUSAGE_ENC_CHALLENGE_KDC

KRB5_KEYUSAGE_ENC_CHALLENGE_KDC

KRB5_KEYUSAGE_ENC_CHALLENGE_KDC	55

KRB5_KEYUSAGE_FAST_ENC

KRB5_KEYUSAGE_FAST_ENC

KRB5_KEYUSAGE_FAST_ENC	51

KRB5_KEYUSAGE_FAST_FINISHED

KRB5_KEYUSAGE_FAST_FINISHED

KRB5_KEYUSAGE_FAST_FINISHED	53

KRB5_KEYUSAGE_FAST_REP

KRB5_KEYUSAGE_FAST_REP

KRB5_KEYUSAGE_FAST_REP	52

KRB5_KEYUSAGE_FAST_REQ_CHKSUM

KRB5_KEYUSAGE_FAST_REQ_CHKSUM

KRB5_KEYUSAGE_FAST_REQ_CHKSUM	50

KRB5_KEYUSAGE_GSS_TOK_MIC

KRB5_KEYUSAGE_GSS_TOK_MIC

KRB5_KEYUSAGE_GSS_TOK_MIC	22

KRB5_KEYUSAGE_GSS_TOK_WRAP_INTEG

KRB5_KEYUSAGE_GSS_TOK_WRAP_INTEG

KRB5_KEYUSAGE_GSS_TOK_WRAP_INTEG	23

KRB5_KEYUSAGE_GSS_TOK_WRAP_PRIV

KRB5_KEYUSAGE_GSS_TOK_WRAP_PRIV

KRB5_KEYUSAGE_GSS_TOK_WRAP_PRIV	24

KRB5_KEYUSAGE_IAKERB_FINISHED

KRB5_KEYUSAGE_IAKERB_FINISHED

KRB5_KEYUSAGE_IAKERB_FINISHED	42

KRB5_KEYUSAGE_KDC_REP_TICKET

KRB5_KEYUSAGE_KDC_REP_TICKET

KRB5_KEYUSAGE_KDC_REP_TICKET	2

KRB5_KEYUSAGE_KRB_CRED_ENCPART

KRB5_KEYUSAGE_KRB_CRED_ENCPART

KRB5_KEYUSAGE_KRB_CRED_ENCPART	14

KRB5_KEYUSAGE_KRB_ERROR_CKSUM

KRB5_KEYUSAGE_KRB_ERROR_CKSUM

KRB5_KEYUSAGE_KRB_ERROR_CKSUM	18

KRB5_KEYUSAGE_KRB_PRIV_ENCPART

KRB5_KEYUSAGE_KRB_PRIV_ENCPART

KRB5_KEYUSAGE_KRB_PRIV_ENCPART	13

KRB5_KEYUSAGE_KRB_SAFE_CKSUM

KRB5_KEYUSAGE_KRB_SAFE_CKSUM

KRB5_KEYUSAGE_KRB_SAFE_CKSUM	15

KRB5_KEYUSAGE_PA_AS_FRESHNESS

KRB5_KEYUSAGE_PA_AS_FRESHNESS

Used for freshness tokens.

KRB5_KEYUSAGE_PA_AS_FRESHNESS	514

KRB5_KEYUSAGE_PA_FX_COOKIE

KRB5_KEYUSAGE_PA_FX_COOKIE

Used for encrypted FAST cookies.

KRB5_KEYUSAGE_PA_FX_COOKIE	513

KRB5_KEYUSAGE_PA_OTP_REQUEST

KRB5_KEYUSAGE_PA_OTP_REQUEST

See RFC 6560 section 4.2.

KRB5_KEYUSAGE_PA_OTP_REQUEST	45

KRB5_KEYUSAGE_PA_PKINIT_KX

KRB5_KEYUSAGE_PA_PKINIT_KX

KRB5_KEYUSAGE_PA_PKINIT_KX	44

KRB5_KEYUSAGE_PA_S4U_X509_USER_REPLY

KRB5_KEYUSAGE_PA_S4U_X509_USER_REPLY

Note conflict with KRB5_KEYUSAGE_PA_SAM_RESPONSE .

KRB5_KEYUSAGE_PA_S4U_X509_USER_REPLY	27

KRB5_KEYUSAGE_PA_S4U_X509_USER_REQUEST

KRB5_KEYUSAGE_PA_S4U_X509_USER_REQUEST

Note conflict with KRB5_KEYUSAGE_PA_SAM_CHALLENGE_TRACKID .

KRB5_KEYUSAGE_PA_S4U_X509_USER_REQUEST	26

KRB5_KEYUSAGE_PA_SAM_CHALLENGE_CKSUM

KRB5_KEYUSAGE_PA_SAM_CHALLENGE_CKSUM

KRB5_KEYUSAGE_PA_SAM_CHALLENGE_CKSUM	25

KRB5_KEYUSAGE_PA_SAM_CHALLENGE_TRACKID

KRB5_KEYUSAGE_PA_SAM_CHALLENGE_TRACKID

Note conflict with KRB5_KEYUSAGE_PA_S4U_X509_USER_REQUEST .

KRB5_KEYUSAGE_PA_SAM_CHALLENGE_TRACKID	26

KRB5_KEYUSAGE_PA_SAM_RESPONSE

KRB5_KEYUSAGE_PA_SAM_RESPONSE

Note conflict with KRB5_KEYUSAGE_PA_S4U_X509_USER_REPLY .

KRB5_KEYUSAGE_PA_SAM_RESPONSE	27

KRB5_KEYUSAGE_SPAKE

KRB5_KEYUSAGE_SPAKE

KRB5_KEYUSAGE_SPAKE	65

KRB5_KEYUSAGE_TGS_REP_ENCPART_SESSKEY

KRB5_KEYUSAGE_TGS_REP_ENCPART_SESSKEY

KRB5_KEYUSAGE_TGS_REP_ENCPART_SESSKEY	8

KRB5_KEYUSAGE_TGS_REP_ENCPART_SUBKEY

KRB5_KEYUSAGE_TGS_REP_ENCPART_SUBKEY

KRB5_KEYUSAGE_TGS_REP_ENCPART_SUBKEY	9

KRB5_KEYUSAGE_TGS_REQ_AD_SESSKEY

KRB5_KEYUSAGE_TGS_REQ_AD_SESSKEY

KRB5_KEYUSAGE_TGS_REQ_AD_SESSKEY	4

KRB5_KEYUSAGE_TGS_REQ_AD_SUBKEY

KRB5_KEYUSAGE_TGS_REQ_AD_SUBKEY

KRB5_KEYUSAGE_TGS_REQ_AD_SUBKEY	5

KRB5_KEYUSAGE_TGS_REQ_AUTH

KRB5_KEYUSAGE_TGS_REQ_AUTH

KRB5_KEYUSAGE_TGS_REQ_AUTH	7

KRB5_KEYUSAGE_TGS_REQ_AUTH_CKSUM

KRB5_KEYUSAGE_TGS_REQ_AUTH_CKSUM

KRB5_KEYUSAGE_TGS_REQ_AUTH_CKSUM	6

KRB5_KPASSWD_ACCESSDENIED

KRB5_KPASSWD_ACCESSDENIED

Not authorized.

KRB5_KPASSWD_ACCESSDENIED	5

KRB5_KPASSWD_AUTHERROR

KRB5_KPASSWD_AUTHERROR

Authentication error.

KRB5_KPASSWD_AUTHERROR	3

KRB5_KPASSWD_BAD_VERSION

KRB5_KPASSWD_BAD_VERSION

Unknown RPC version.

KRB5_KPASSWD_BAD_VERSION	6

KRB5_KPASSWD_HARDERROR

KRB5_KPASSWD_HARDERROR

Server error.

KRB5_KPASSWD_HARDERROR	2

KRB5_KPASSWD_INITIAL_FLAG_NEEDED

KRB5_KPASSWD_INITIAL_FLAG_NEEDED

The presented credentials were not obtained using a password directly.

KRB5_KPASSWD_INITIAL_FLAG_NEEDED	7

KRB5_KPASSWD_MALFORMED

KRB5_KPASSWD_MALFORMED

Malformed request.

KRB5_KPASSWD_MALFORMED	1

KRB5_KPASSWD_SOFTERROR

KRB5_KPASSWD_SOFTERROR

Password change rejected.

KRB5_KPASSWD_SOFTERROR	4

KRB5_KPASSWD_SUCCESS

KRB5_KPASSWD_SUCCESS

Success.

KRB5_KPASSWD_SUCCESS	0

KRB5_LRQ_ALL_ACCT_EXPTIME

KRB5_LRQ_ALL_ACCT_EXPTIME

KRB5_LRQ_ALL_ACCT_EXPTIME	7

KRB5_LRQ_ALL_LAST_INITIAL

KRB5_LRQ_ALL_LAST_INITIAL

KRB5_LRQ_ALL_LAST_INITIAL	2

KRB5_LRQ_ALL_LAST_RENEWAL

KRB5_LRQ_ALL_LAST_RENEWAL

KRB5_LRQ_ALL_LAST_RENEWAL	4

KRB5_LRQ_ALL_LAST_REQ

KRB5_LRQ_ALL_LAST_REQ

KRB5_LRQ_ALL_LAST_REQ	5

KRB5_LRQ_ALL_LAST_TGT

KRB5_LRQ_ALL_LAST_TGT

KRB5_LRQ_ALL_LAST_TGT	1

KRB5_LRQ_ALL_LAST_TGT_ISSUED

KRB5_LRQ_ALL_LAST_TGT_ISSUED

KRB5_LRQ_ALL_LAST_TGT_ISSUED	3

KRB5_LRQ_ALL_PW_EXPTIME

KRB5_LRQ_ALL_PW_EXPTIME

KRB5_LRQ_ALL_PW_EXPTIME	6

KRB5_LRQ_NONE

KRB5_LRQ_NONE

KRB5_LRQ_NONE	0

KRB5_LRQ_ONE_ACCT_EXPTIME

KRB5_LRQ_ONE_ACCT_EXPTIME

KRB5_LRQ_ONE_ACCT_EXPTIME	(-7)

KRB5_LRQ_ONE_LAST_INITIAL

KRB5_LRQ_ONE_LAST_INITIAL

KRB5_LRQ_ONE_LAST_INITIAL	(-2)

KRB5_LRQ_ONE_LAST_RENEWAL

KRB5_LRQ_ONE_LAST_RENEWAL

KRB5_LRQ_ONE_LAST_RENEWAL	(-4)

KRB5_LRQ_ONE_LAST_REQ

KRB5_LRQ_ONE_LAST_REQ

KRB5_LRQ_ONE_LAST_REQ	(-5)

KRB5_LRQ_ONE_LAST_TGT

KRB5_LRQ_ONE_LAST_TGT

KRB5_LRQ_ONE_LAST_TGT	(-1)

KRB5_LRQ_ONE_LAST_TGT_ISSUED

KRB5_LRQ_ONE_LAST_TGT_ISSUED

KRB5_LRQ_ONE_LAST_TGT_ISSUED	(-3)

KRB5_LRQ_ONE_PW_EXPTIME

KRB5_LRQ_ONE_PW_EXPTIME

KRB5_LRQ_ONE_PW_EXPTIME	(-6)

KRB5_NT_ENTERPRISE_PRINCIPAL

KRB5_NT_ENTERPRISE_PRINCIPAL

Windows 2000 UPN.

KRB5_NT_ENTERPRISE_PRINCIPAL	10

KRB5_NT_ENT_PRINCIPAL_AND_ID

KRB5_NT_ENT_PRINCIPAL_AND_ID

NT 4 style name and SID.

KRB5_NT_ENT_PRINCIPAL_AND_ID	-130

KRB5_NT_MS_PRINCIPAL

KRB5_NT_MS_PRINCIPAL

Windows 2000 UPN and SID.

KRB5_NT_MS_PRINCIPAL	-128

KRB5_NT_MS_PRINCIPAL_AND_ID

KRB5_NT_MS_PRINCIPAL_AND_ID

NT 4 style name.

KRB5_NT_MS_PRINCIPAL_AND_ID	-129

KRB5_NT_PRINCIPAL

KRB5_NT_PRINCIPAL

Just the name of the principal as in DCE, or for users.

KRB5_NT_PRINCIPAL	1

KRB5_NT_SMTP_NAME

KRB5_NT_SMTP_NAME

Name in form of SMTP email name.

KRB5_NT_SMTP_NAME	7

KRB5_NT_SRV_HST

KRB5_NT_SRV_HST

Service with host name as instance (telnet, rcommands)

KRB5_NT_SRV_HST	3

KRB5_NT_SRV_INST

KRB5_NT_SRV_INST

Service and other unique instance (krbtgt)

KRB5_NT_SRV_INST	2

KRB5_NT_SRV_XHST

KRB5_NT_SRV_XHST

Service with host as remaining components.

KRB5_NT_SRV_XHST	4

KRB5_NT_UID

KRB5_NT_UID

Unique ID.

KRB5_NT_UID	5

KRB5_NT_UNKNOWN

KRB5_NT_UNKNOWN

Name type not known.

KRB5_NT_UNKNOWN	0

KRB5_NT_WELLKNOWN

KRB5_NT_WELLKNOWN

Well-known (special) principal.

KRB5_NT_WELLKNOWN	11

KRB5_NT_X500_PRINCIPAL

KRB5_NT_X500_PRINCIPAL

PKINIT.

KRB5_NT_X500_PRINCIPAL	6

KRB5_PAC_CLIENT_INFO

KRB5_PAC_CLIENT_INFO

Client name and ticket info.

KRB5_PAC_CLIENT_INFO	10

KRB5_PAC_CREDENTIALS_INFO

KRB5_PAC_CREDENTIALS_INFO

Credentials information.

KRB5_PAC_CREDENTIALS_INFO	2

KRB5_PAC_DELEGATION_INFO

KRB5_PAC_DELEGATION_INFO

Constrained delegation info.

KRB5_PAC_DELEGATION_INFO	11

KRB5_PAC_LOGON_INFO

KRB5_PAC_LOGON_INFO

Logon information.

KRB5_PAC_LOGON_INFO	1

KRB5_PAC_PRIVSVR_CHECKSUM

KRB5_PAC_PRIVSVR_CHECKSUM

KDC checksum.

KRB5_PAC_PRIVSVR_CHECKSUM	7

KRB5_PAC_SERVER_CHECKSUM

KRB5_PAC_SERVER_CHECKSUM

Server checksum.

KRB5_PAC_SERVER_CHECKSUM	6

KRB5_PAC_UPN_DNS_INFO

KRB5_PAC_UPN_DNS_INFO

User principal name and DNS info.

KRB5_PAC_UPN_DNS_INFO	12

KRB5_PADATA_AFS3_SALT

KRB5_PADATA_AFS3_SALT

Cygnus.

RFC 4120, 3961

KRB5_PADATA_AFS3_SALT	10

KRB5_PADATA_AP_REQ

KRB5_PADATA_AP_REQ

KRB5_PADATA_AP_REQ	1

KRB5_PADATA_AS_CHECKSUM

KRB5_PADATA_AS_CHECKSUM

AS checksum.

KRB5_PADATA_AS_CHECKSUM	132

KRB5_PADATA_AS_FRESHNESS

KRB5_PADATA_AS_FRESHNESS

RFC 8070.

KRB5_PADATA_AS_FRESHNESS	150

KRB5_PADATA_ENCRYPTED_CHALLENGE

KRB5_PADATA_ENCRYPTED_CHALLENGE

RFC 6113.

KRB5_PADATA_ENCRYPTED_CHALLENGE	138

KRB5_PADATA_ENC_SANDIA_SECURID

KRB5_PADATA_ENC_SANDIA_SECURID

SecurId passcode.

RFC 4120

KRB5_PADATA_ENC_SANDIA_SECURID	6

KRB5_PADATA_ENC_TIMESTAMP

KRB5_PADATA_ENC_TIMESTAMP

RFC 4120.

KRB5_PADATA_ENC_TIMESTAMP	2

KRB5_PADATA_ENC_UNIX_TIME

KRB5_PADATA_ENC_UNIX_TIME

timestamp encrypted in key.

RFC 4120

KRB5_PADATA_ENC_UNIX_TIME	5

KRB5_PADATA_ETYPE_INFO

KRB5_PADATA_ETYPE_INFO

Etype info for preauth.

RFC 4120

KRB5_PADATA_ETYPE_INFO	11

KRB5_PADATA_ETYPE_INFO2

KRB5_PADATA_ETYPE_INFO2

RFC 4120.

KRB5_PADATA_ETYPE_INFO2	19

KRB5_PADATA_FOR_USER

KRB5_PADATA_FOR_USER

username protocol transition request

KRB5_PADATA_FOR_USER	129

KRB5_PADATA_FX_COOKIE

KRB5_PADATA_FX_COOKIE

RFC 6113.

KRB5_PADATA_FX_COOKIE	133

KRB5_PADATA_FX_ERROR

KRB5_PADATA_FX_ERROR

RFC 6113.

KRB5_PADATA_FX_ERROR	137

KRB5_PADATA_FX_FAST

KRB5_PADATA_FX_FAST

RFC 6113.

KRB5_PADATA_FX_FAST	136

KRB5_PADATA_GET_FROM_TYPED_DATA

KRB5_PADATA_GET_FROM_TYPED_DATA

Embedded in typed data.

RFC 4120

KRB5_PADATA_GET_FROM_TYPED_DATA	22

KRB5_PADATA_NONE

KRB5_PADATA_NONE

KRB5_PADATA_NONE	0

KRB5_PADATA_OSF_DCE

KRB5_PADATA_OSF_DCE

OSF DCE.

RFC 4120

KRB5_PADATA_OSF_DCE	8

KRB5_PADATA_OTP_CHALLENGE

KRB5_PADATA_OTP_CHALLENGE

RFC 6560 section 4.1.

KRB5_PADATA_OTP_CHALLENGE	141

KRB5_PADATA_OTP_PIN_CHANGE

KRB5_PADATA_OTP_PIN_CHANGE

RFC 6560 section 4.3.

KRB5_PADATA_OTP_PIN_CHANGE	144

KRB5_PADATA_OTP_REQUEST

KRB5_PADATA_OTP_REQUEST

RFC 6560 section 4.2.

KRB5_PADATA_OTP_REQUEST	142

KRB5_PADATA_PAC_OPTIONS

KRB5_PADATA_PAC_OPTIONS

MS-KILE and MS-SFU.

KRB5_PADATA_PAC_OPTIONS	167

KRB5_PADATA_PAC_REQUEST

KRB5_PADATA_PAC_REQUEST

include Windows PAC

KRB5_PADATA_PAC_REQUEST	128

KRB5_PADATA_PKINIT_KX

KRB5_PADATA_PKINIT_KX

RFC 6112.

KRB5_PADATA_PKINIT_KX	147

KRB5_PADATA_PK_AS_REP

KRB5_PADATA_PK_AS_REP

PKINIT.

RFC 4556

KRB5_PADATA_PK_AS_REP	17

KRB5_PADATA_PK_AS_REP_OLD

KRB5_PADATA_PK_AS_REP_OLD

PKINIT.

KRB5_PADATA_PK_AS_REP_OLD	15

KRB5_PADATA_PK_AS_REQ

KRB5_PADATA_PK_AS_REQ

PKINIT.

RFC 4556

KRB5_PADATA_PK_AS_REQ	16

KRB5_PADATA_PK_AS_REQ_OLD

KRB5_PADATA_PK_AS_REQ_OLD

PKINIT.

KRB5_PADATA_PK_AS_REQ_OLD	14

KRB5_PADATA_PW_SALT

KRB5_PADATA_PW_SALT

RFC 4120.

KRB5_PADATA_PW_SALT	3

KRB5_PADATA_REFERRAL

KRB5_PADATA_REFERRAL

draft referral system

KRB5_PADATA_REFERRAL	25

KRB5_PADATA_S4U_X509_USER

KRB5_PADATA_S4U_X509_USER

certificate protocol transition request

KRB5_PADATA_S4U_X509_USER	130

KRB5_PADATA_SAM_CHALLENGE

KRB5_PADATA_SAM_CHALLENGE

SAM/OTP.

KRB5_PADATA_SAM_CHALLENGE	12

KRB5_PADATA_SAM_CHALLENGE_2

KRB5_PADATA_SAM_CHALLENGE_2

draft challenge system, updated

KRB5_PADATA_SAM_CHALLENGE_2	30

KRB5_PADATA_SAM_REDIRECT

KRB5_PADATA_SAM_REDIRECT

SAM/OTP.

RFC 4120

KRB5_PADATA_SAM_REDIRECT	21

KRB5_PADATA_SAM_RESPONSE

KRB5_PADATA_SAM_RESPONSE

SAM/OTP.

KRB5_PADATA_SAM_RESPONSE	13

KRB5_PADATA_SAM_RESPONSE_2

KRB5_PADATA_SAM_RESPONSE_2

draft challenge system, updated

KRB5_PADATA_SAM_RESPONSE_2	31

KRB5_PADATA_SESAME

KRB5_PADATA_SESAME

Sesame project.

RFC 4120

KRB5_PADATA_SESAME	7

KRB5_PADATA_SPAKE

KRB5_PADATA_SPAKE

KRB5_PADATA_SPAKE	151

KRB5_PADATA_SVR_REFERRAL_INFO

KRB5_PADATA_SVR_REFERRAL_INFO

Windows 2000 referrals.

RFC 6820

KRB5_PADATA_SVR_REFERRAL_INFO	20

KRB5_PADATA_TGS_REQ

KRB5_PADATA_TGS_REQ

KRB5_PADATA_TGS_REQ	KRB5_PADATA_AP_REQ

KRB5_PADATA_USE_SPECIFIED_KVNO

KRB5_PADATA_USE_SPECIFIED_KVNO

RFC 4120.

KRB5_PADATA_USE_SPECIFIED_KVNO	20

KRB5_PRINCIPAL_COMPARE_CASEFOLD

KRB5_PRINCIPAL_COMPARE_CASEFOLD

case-insensitive

KRB5_PRINCIPAL_COMPARE_CASEFOLD	4

KRB5_PRINCIPAL_COMPARE_ENTERPRISE

KRB5_PRINCIPAL_COMPARE_ENTERPRISE

UPNs as real principals.

KRB5_PRINCIPAL_COMPARE_ENTERPRISE	2

KRB5_PRINCIPAL_COMPARE_IGNORE_REALM

KRB5_PRINCIPAL_COMPARE_IGNORE_REALM

ignore realm component

KRB5_PRINCIPAL_COMPARE_IGNORE_REALM	1

KRB5_PRINCIPAL_COMPARE_UTF8

KRB5_PRINCIPAL_COMPARE_UTF8

treat principals as UTF-8

KRB5_PRINCIPAL_COMPARE_UTF8	8

KRB5_PRINCIPAL_PARSE_ENTERPRISE

KRB5_PRINCIPAL_PARSE_ENTERPRISE

Create single-component enterprise principle.

KRB5_PRINCIPAL_PARSE_ENTERPRISE	0x4

KRB5_PRINCIPAL_PARSE_IGNORE_REALM

KRB5_PRINCIPAL_PARSE_IGNORE_REALM

Ignore realm if present.

KRB5_PRINCIPAL_PARSE_IGNORE_REALM	0x8

KRB5_PRINCIPAL_PARSE_NO_REALM

KRB5_PRINCIPAL_PARSE_NO_REALM

Error if realm is present.

KRB5_PRINCIPAL_PARSE_NO_REALM	0x1

KRB5_PRINCIPAL_PARSE_REQUIRE_REALM

KRB5_PRINCIPAL_PARSE_REQUIRE_REALM

Error if realm is not present.

KRB5_PRINCIPAL_PARSE_REQUIRE_REALM	0x2

KRB5_PRINCIPAL_UNPARSE_DISPLAY

KRB5_PRINCIPAL_UNPARSE_DISPLAY

Don't escape special characters.

KRB5_PRINCIPAL_UNPARSE_DISPLAY	0x4

KRB5_PRINCIPAL_UNPARSE_NO_REALM

KRB5_PRINCIPAL_UNPARSE_NO_REALM

Omit realm always.

KRB5_PRINCIPAL_UNPARSE_NO_REALM	0x2

KRB5_PRINCIPAL_UNPARSE_SHORT

KRB5_PRINCIPAL_UNPARSE_SHORT

Omit realm if it is the local realm.

KRB5_PRINCIPAL_UNPARSE_SHORT	0x1

KRB5_PRIV

KRB5_PRIV

Private application message.

KRB5_PRIV	((krb5_msgtype)21)

KRB5_PROMPT_TYPE_NEW_PASSWORD

KRB5_PROMPT_TYPE_NEW_PASSWORD

Prompt for new password (during password change)

KRB5_PROMPT_TYPE_NEW_PASSWORD	0x2

KRB5_PROMPT_TYPE_NEW_PASSWORD_AGAIN

KRB5_PROMPT_TYPE_NEW_PASSWORD_AGAIN

Prompt for new password again.

KRB5_PROMPT_TYPE_NEW_PASSWORD_AGAIN	0x3

KRB5_PROMPT_TYPE_PASSWORD

KRB5_PROMPT_TYPE_PASSWORD

Prompt for password.

KRB5_PROMPT_TYPE_PASSWORD	0x1

KRB5_PROMPT_TYPE_PREAUTH

KRB5_PROMPT_TYPE_PREAUTH

Prompt for preauthentication data (such as an OTP value)

KRB5_PROMPT_TYPE_PREAUTH	0x4

KRB5_PVNO

KRB5_PVNO

Protocol version number.

KRB5_PVNO	5

KRB5_REALM_BRANCH_CHAR

KRB5_REALM_BRANCH_CHAR

KRB5_REALM_BRANCH_CHAR	'.'

KRB5_RECVAUTH_BADAUTHVERS

KRB5_RECVAUTH_BADAUTHVERS

KRB5_RECVAUTH_BADAUTHVERS	0x0002

KRB5_RECVAUTH_SKIP_VERSION

KRB5_RECVAUTH_SKIP_VERSION

KRB5_RECVAUTH_SKIP_VERSION	0x0001

KRB5_REFERRAL_REALM

KRB5_REFERRAL_REALM

Constant for realm referrals.

KRB5_REFERRAL_REALM	""

KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_USER_PIN_COUNT_LOW

KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_USER_PIN_COUNT_LOW

This flag indicates that an incorrect PIN was supplied at least once since the last time the correct PIN was supplied.

KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_USER_PIN_COUNT_LOW	(1 << 0)

KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_USER_PIN_FINAL_TRY

KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_USER_PIN_FINAL_TRY

This flag indicates that supplying an incorrect PIN will cause the token to lock itself.

KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_USER_PIN_FINAL_TRY	(1 << 1)

KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_USER_PIN_LOCKED

KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_USER_PIN_LOCKED

This flag indicates that the user PIN is locked, and you can't log in to the token with it.

KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_USER_PIN_LOCKED	(1 << 2)

KRB5_RESPONDER_QUESTION_PKINIT

KRB5_RESPONDER_QUESTION_PKINIT

PKINIT responder question.

The PKINIT responder question is asked when the client needs a password that's being used to protect key information, and is formatted as a JSON object. A specific identity's flags value, if not zero, is the bitwise-OR of one or more of the KRB5_RESPONDER_PKINIT_FLAGS_TOKEN_* flags defined below, and possibly other flags to be added later. Any resemblance to similarly-named CKF_* values in the PKCS#11 API should not be depended on.

{

identity <string> : flags <number>,

...

}

The answer to the question MUST be JSON formatted:

{

identity <string> : password <string>,

...

}

KRB5_RESPONDER_QUESTION_PKINIT	"pkinit"

KRB5_RESPONDER_OTP_FLAGS_COLLECT_PIN

KRB5_RESPONDER_OTP_FLAGS_COLLECT_PIN

This flag indicates that the PIN value MUST be collected.

KRB5_RESPONDER_OTP_FLAGS_COLLECT_PIN	0x0002

KRB5_RESPONDER_OTP_FLAGS_COLLECT_TOKEN

KRB5_RESPONDER_OTP_FLAGS_COLLECT_TOKEN

This flag indicates that the token value MUST be collected.

KRB5_RESPONDER_OTP_FLAGS_COLLECT_TOKEN	0x0001

KRB5_RESPONDER_OTP_FLAGS_NEXTOTP

KRB5_RESPONDER_OTP_FLAGS_NEXTOTP

This flag indicates that the token is now in re-synchronization mode with the server.

The user is expected to reply with the next code displayed on the token.

KRB5_RESPONDER_OTP_FLAGS_NEXTOTP	0x0004

KRB5_RESPONDER_OTP_FLAGS_SEPARATE_PIN

KRB5_RESPONDER_OTP_FLAGS_SEPARATE_PIN

This flag indicates that the PIN MUST be returned as a separate item.

This flag only takes effect if KRB5_RESPONDER_OTP_FLAGS_COLLECT_PIN is set. If this flag is not set, the responder may either concatenate PIN + token value and store it as "value" in the answer or it may return them separately. If they are returned separately, they will be concatenated internally.

KRB5_RESPONDER_OTP_FLAGS_SEPARATE_PIN	0x0008

KRB5_RESPONDER_OTP_FORMAT_ALPHANUMERIC

KRB5_RESPONDER_OTP_FORMAT_ALPHANUMERIC

KRB5_RESPONDER_OTP_FORMAT_ALPHANUMERIC	2

KRB5_RESPONDER_OTP_FORMAT_DECIMAL

KRB5_RESPONDER_OTP_FORMAT_DECIMAL

These format constants identify the format of the token value.

KRB5_RESPONDER_OTP_FORMAT_DECIMAL	0

KRB5_RESPONDER_OTP_FORMAT_HEXADECIMAL

KRB5_RESPONDER_OTP_FORMAT_HEXADECIMAL

KRB5_RESPONDER_OTP_FORMAT_HEXADECIMAL	1

KRB5_RESPONDER_QUESTION_OTP

KRB5_RESPONDER_QUESTION_OTP

OTP responder question.

The OTP responder question is asked when the KDC indicates that an OTP value is required in order to complete the authentication. The JSON format of the challenge is:

{

"service": <string (optional)>,

"tokenInfo": [

{

"flags": <number>,

"vendor": <string (optional)>,

"challenge": <string (optional)>,

"length": <number (optional)>,

"format": <number (optional)>,

"tokenID": <string (optional)>,

"algID": <string (optional)>,

},

...

]

}

The answer to the question MUST be JSON formatted:

{

"tokeninfo": <number>,

"value": <string (optional)>,

"pin": <string (optional)>,

}

For more detail, please see RFC 6560.

KRB5_RESPONDER_QUESTION_OTP	"otp"

KRB5_RESPONDER_QUESTION_PASSWORD

KRB5_RESPONDER_QUESTION_PASSWORD

Long-term password responder question.

This question is asked when the long-term password is needed. It has no challenge and the response is simply the password string.

KRB5_RESPONDER_QUESTION_PASSWORD	"password"

KRB5_SAFE

KRB5_SAFE

Safe application message.

KRB5_SAFE	((krb5_msgtype)20)

KRB5_SAM_MUST_PK_ENCRYPT_SAD

KRB5_SAM_MUST_PK_ENCRYPT_SAD

currently must be zero

KRB5_SAM_MUST_PK_ENCRYPT_SAD	0x20000000

KRB5_SAM_SEND_ENCRYPTED_SAD

KRB5_SAM_SEND_ENCRYPTED_SAD

KRB5_SAM_SEND_ENCRYPTED_SAD	0x40000000

KRB5_SAM_USE_SAD_AS_KEY

KRB5_SAM_USE_SAD_AS_KEY

KRB5_SAM_USE_SAD_AS_KEY	0x80000000

KRB5_TC_MATCH_2ND_TKT

KRB5_TC_MATCH_2ND_TKT

The second ticket must match.

KRB5_TC_MATCH_2ND_TKT	0x00000080

KRB5_TC_MATCH_AUTHDATA

KRB5_TC_MATCH_AUTHDATA

The authorization data must match.

KRB5_TC_MATCH_AUTHDATA	0x00000020

KRB5_TC_MATCH_FLAGS

KRB5_TC_MATCH_FLAGS

All the flags set in the match credentials must be set.

KRB5_TC_MATCH_FLAGS	0x00000004

KRB5_TC_MATCH_FLAGS_EXACT

KRB5_TC_MATCH_FLAGS_EXACT

All the flags must match exactly.

KRB5_TC_MATCH_FLAGS_EXACT	0x00000010

KRB5_TC_MATCH_IS_SKEY

KRB5_TC_MATCH_IS_SKEY

The is_skey field must match exactly.

KRB5_TC_MATCH_IS_SKEY	0x00000002

KRB5_TC_MATCH_KTYPE

KRB5_TC_MATCH_KTYPE

The encryption key type must match.

KRB5_TC_MATCH_KTYPE	0x00000100

KRB5_TC_MATCH_SRV_NAMEONLY

KRB5_TC_MATCH_SRV_NAMEONLY

Only the name portion of the principal name must match.

KRB5_TC_MATCH_SRV_NAMEONLY	0x00000040

KRB5_TC_MATCH_TIMES

KRB5_TC_MATCH_TIMES

The requested lifetime must be at least as great as the time specified.

KRB5_TC_MATCH_TIMES	0x00000001

KRB5_TC_MATCH_TIMES_EXACT

KRB5_TC_MATCH_TIMES_EXACT

All the time fields must match exactly.

KRB5_TC_MATCH_TIMES_EXACT	0x00000008

KRB5_TC_NOTICKET

KRB5_TC_NOTICKET

KRB5_TC_NOTICKET	0x00000002

KRB5_TC_OPENCLOSE

KRB5_TC_OPENCLOSE

Open and close the file for each cache operation.

KRB5_TC_OPENCLOSE	0x00000001

KRB5_TC_SUPPORTED_KTYPES

KRB5_TC_SUPPORTED_KTYPES

The supported key types must match.

KRB5_TC_SUPPORTED_KTYPES	0x00000200

KRB5_TGS_NAME

KRB5_TGS_NAME

KRB5_TGS_NAME	"krbtgt"

KRB5_TGS_NAME_SIZE

KRB5_TGS_NAME_SIZE

KRB5_TGS_NAME_SIZE	6

KRB5_TGS_REP

KRB5_TGS_REP

Response to TGS request.

KRB5_TGS_REP	((krb5_msgtype)13)

KRB5_TGS_REQ

KRB5_TGS_REQ

Ticket granting server request.

KRB5_TGS_REQ	((krb5_msgtype)12)

KRB5_TKT_CREDS_STEP_FLAG_CONTINUE

KRB5_TKT_CREDS_STEP_FLAG_CONTINUE

More responses needed.

KRB5_TKT_CREDS_STEP_FLAG_CONTINUE	0x1

KRB5_VERIFY_INIT_CREDS_OPT_AP_REQ_NOFAIL

KRB5_VERIFY_INIT_CREDS_OPT_AP_REQ_NOFAIL

KRB5_VERIFY_INIT_CREDS_OPT_AP_REQ_NOFAIL	0x0001

KRB5_WELLKNOWN_NAMESTR

KRB5_WELLKNOWN_NAMESTR

First component of NT_WELLKNOWN principals.

KRB5_WELLKNOWN_NAMESTR	"WELLKNOWN"

LR_TYPE_INTERPRETATION_MASK

LR_TYPE_INTERPRETATION_MASK

LR_TYPE_INTERPRETATION_MASK	0x7fff

LR_TYPE_THIS_SERVER_ONLY

LR_TYPE_THIS_SERVER_ONLY

LR_TYPE_THIS_SERVER_ONLY	0x8000

MAX_KEYTAB_NAME_LEN

MAX_KEYTAB_NAME_LEN

Long enough for MAXPATHLEN + some extra.

MAX_KEYTAB_NAME_LEN	1100

MSEC_DIRBIT

MSEC_DIRBIT

MSEC_DIRBIT	0x8000

MSEC_VAL_MASK

MSEC_VAL_MASK

MSEC_VAL_MASK	0x7fff

SALT_TYPE_AFS_LENGTH

SALT_TYPE_AFS_LENGTH

SALT_TYPE_AFS_LENGTH	UINT_MAX

SALT_TYPE_NO_LENGTH

SALT_TYPE_NO_LENGTH

SALT_TYPE_NO_LENGTH	UINT_MAX

THREEPARAMOPEN

THREEPARAMOPEN

THREEPARAMOPEN (x, y, z)	open(x,y,z)

TKT_FLG_ANONYMOUS

TKT_FLG_ANONYMOUS

TKT_FLG_ANONYMOUS	0x00008000

TKT_FLG_ENC_PA_REP

TKT_FLG_ENC_PA_REP

TKT_FLG_ENC_PA_REP	0x00010000

TKT_FLG_FORWARDABLE

TKT_FLG_FORWARDABLE

TKT_FLG_FORWARDABLE	0x40000000

TKT_FLG_FORWARDED

TKT_FLG_FORWARDED

TKT_FLG_FORWARDED	0x20000000

TKT_FLG_HW_AUTH

TKT_FLG_HW_AUTH

TKT_FLG_HW_AUTH	0x00100000

TKT_FLG_INITIAL

TKT_FLG_INITIAL

TKT_FLG_INITIAL	0x00400000

TKT_FLG_INVALID

TKT_FLG_INVALID

TKT_FLG_INVALID	0x01000000

TKT_FLG_MAY_POSTDATE

TKT_FLG_MAY_POSTDATE

TKT_FLG_MAY_POSTDATE	0x04000000

TKT_FLG_OK_AS_DELEGATE

TKT_FLG_OK_AS_DELEGATE

TKT_FLG_OK_AS_DELEGATE	0x00040000

TKT_FLG_POSTDATED

TKT_FLG_POSTDATED

TKT_FLG_POSTDATED	0x02000000

TKT_FLG_PRE_AUTH

TKT_FLG_PRE_AUTH

TKT_FLG_PRE_AUTH	0x00200000

TKT_FLG_PROXIABLE

TKT_FLG_PROXIABLE

TKT_FLG_PROXIABLE	0x10000000

TKT_FLG_PROXY

TKT_FLG_PROXY

TKT_FLG_PROXY	0x08000000

TKT_FLG_RENEWABLE

TKT_FLG_RENEWABLE

TKT_FLG_RENEWABLE	0x00800000

TKT_FLG_TRANSIT_POLICY_CHECKED

TKT_FLG_TRANSIT_POLICY_CHECKED

TKT_FLG_TRANSIT_POLICY_CHECKED	0x00080000

VALID_INT_BITS

VALID_INT_BITS

VALID_INT_BITS	INT_MAX

VALID_UINT_BITS

VALID_UINT_BITS

VALID_UINT_BITS	UINT_MAX

krb5_const

krb5_const

krb5_const	const

krb5_princ_component

krb5_princ_component

krb5_princ_component (context, princ, i)	(((i) < krb5_princ_size(context, princ)) ? (princ)->data + (i) : NULL)

krb5_princ_name

krb5_princ_name

krb5_princ_name (context, princ)	(princ)->data

krb5_princ_realm

krb5_princ_realm

krb5_princ_realm (context, princ)	(&(princ)->realm)

krb5_princ_set_realm

krb5_princ_set_realm

krb5_princ_set_realm (context, princ, value)	((princ)->realm = *(value))

krb5_princ_set_realm_data

krb5_princ_set_realm_data

krb5_princ_set_realm_data (context, princ, value)	(princ)->realm.data = (value)

krb5_princ_set_realm_length

krb5_princ_set_realm_length

krb5_princ_set_realm_length (context, princ, value)	(princ)->realm.length = (value)

krb5_princ_size

krb5_princ_size

krb5_princ_size (context, princ)	(princ)->length

krb5_princ_type

krb5_princ_type

krb5_princ_type (context, princ)	(princ)->type

krb5_roundup

krb5_roundup

krb5_roundup (x, y)	((((x) + (y) - 1)/(y))*(y))

krb5_x

krb5_x

krb5_x (ptr, args)	((ptr)?((*(ptr)) args):(abort(),1))

krb5_xc

krb5_xc

krb5_xc (ptr, args)	((ptr)?((*(ptr)) args):(abort(),(char*)0))

6.3.2 Deprecated macros

krb524_convert_creds_kdc

krb524_convert_creds_kdc

krb524_convert_creds_kdc	krb5_524_convert_creds

krb524_init_ets

krb524_init_ets

krb524_init_ets (x)	(0)

INDEX

K

L

M

P

R

S

T

V