Kerberos Application Developer Guide

Release 1.11.1

MIT
## CONTENTS

1 Developing with GSSAPI
   1.1 Name types ...................................................... 1
   1.2 Initiator credentials ........................................... 1
   1.3 Acceptor names ................................................. 2
   1.4 Importing and exporting credentials .......................... 3

2 Differences between Heimdal and MIT Kerberos API ................. 5

3 Initial credentials
   3.1 Options for get_init_creds ..................................... 8
   3.2 Getting anonymous credentials .................................. 8
   3.3 User interaction .................................................. 9
   3.4 Verifying initial credentials .................................... 11

4 Principal manipulation and parsing ................................... 13

5 Complete reference - API and datatypes ............................. 15
   5.1 krb5 API ....................................................... 15
   5.2 krb5 types and structures ....................................... 131
   5.3 krb5 simple macros .............................................. 159

Index ............................................................................. 207
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. This documentation will describe how various ways of using 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_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.

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/host 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 FILE:/etc/krb5.keytab, 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 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 unserializes 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.
**DIFFERENCES BETWEEN HEIMDAL AND MIT KERBEROS API**

<table>
<thead>
<tr>
<th>Function</th>
<th>H5I:</th>
<th>MIT:</th>
</tr>
</thead>
<tbody>
<tr>
<td>krb5_auth_con_getaddr()</td>
<td>If either of the pointers to local_addr and remote_addr is not NULL, it is freed first and then reallocated before being populated with the content of corresponding address from authentication context.</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_setaddr()</td>
<td>If either address is NULL, the previous address remains in place.</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_setports()</td>
<td>Not implemented as of version 1.3.3.</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_setrecvsubkey()</td>
<td>If either port is NULL, the previous port remains in place.</td>
<td></td>
</tr>
<tr>
<td>krb5_auth_con_setsendsubkey()</td>
<td>Not implemented as of version 1.3.3.</td>
<td></td>
</tr>
<tr>
<td>krb5_cc_set_config()</td>
<td>Before version 1.10 it was assumed that the last argument data is ALWAYS non-zero.</td>
<td></td>
</tr>
<tr>
<td>krb5_cccol_last_change_time()</td>
<td>H5I takes 3 arguments: krb5_context context, const char *type, krb5_timestamp *change_time.</td>
<td></td>
</tr>
<tr>
<td>krb5_set_default_realm()</td>
<td>Caches the computed default realm context field. If the second argument is NULL, it tries to retrieve it from libdefaults or DNS.</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER THREE

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:

```c
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;
```
3.1 Options for get_init_creds

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

```c
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;
```

3.2 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 \texttt{anonymous_pkinit}), but does not support issuing realm-exposed anonymous credentials at this time.

To obtain fully anonymous credentials, call \texttt{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 \texttt{@realmname}). Authentication will take place using anonymous PKINIT; if successful, the client principal of the resulting tickets will be \texttt{WELLKNOWN/ANONYMOUS@WELLKNOWN:ANONYMOUS}. Here is an example:

```c
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 \texttt{WELLKNOWN/ANONYMOUS@realmname}.
3.3 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.

### 3.3.1 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:

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

### 3.3.2 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.

Example

Here is an example of using a responder callback:

```c
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;
}
```

```c
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);
```
3.4 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:

```c
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 FOUR

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()
5.1 krb5 API

5.1.1 Frequently used public interfaces

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

```c
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_alloc_va()** 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.

```c
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

---

15
[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.**

```c
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.**

```c
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 credentials cache name.

krb5_error_code krb5_cc_default (krb5_context context, krb5_ccache *ccache)

param [inout] 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

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.

Try the environment variable KRB5CCNAME first then, if it is not set, fall back on the default ccache name for the OS.

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.

```c
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.

```c
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

**return**
- 0 Success
- 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.

```c
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.

```c
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
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.

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

**Parameters**
- `context` [in]: Library context.
- `type` [in]: Credential cache type name.
- `hint` [in]: Unused.
- `id` [out]: Credential cache handle.

**Return Values**
- `0` Success.
- 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_cc_resolve - Resolve a credential cache name.

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

**Parameters**
- `context` [in]: Library context.
- `name` [in]: Credential cache name to be resolved.
- `cache` [out]: Credential cache handle.

**Return Values**
- `0` Success.
- Kerberos error codes.

### krb5_change_password - Change a password for an existing Kerberos account.

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

**Parameters**
- `context` [in]: Library context.
- `creds` [in]: Credentials for kadmin/changepw service.
- `newpw` [in]: New password.

**Return Values**
- `0` Success.
- Kerberos error codes.
[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.

```
#include <lib/krb5.h>
#include <lib/krb5_error_code.h>

void 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:** First introduced in 1.11

**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().

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

- **param [in]** ctx - Library context
- **param [in]** msg - Pointer to error message

**krb5_free_principal** - Free the storage assigned to a principal.

```c
void krb5_free_principal (krb5_context context, krb5_principal val)
```

- **param [in]** context - Library context
- **param [in]** val - Principal to be freed

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

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

- **param [in]** context - Library context
- **param [in]** auth_context - Authentication context
- **param [in]** rhost - Remote host
- **param [in]** client - Client principal of TGT
- **param [in]** server - Principal of server to receive TGT
- **param [in]** cc - Credential cache handle (NULL to use default)
- **param [in]** forwardable - Whether TGT should be forwardable
- **param [out]** outbuf - KRB-CRED message

**return**

- **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

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.

```c
krb5_error_code krb5_get_default_realm (krb5_context context, char ** lrealm)
```
Retrieve 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.**

```c
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.**

```c
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
- **[inout] ccache** - Credential cache handle
- **[in] in_creds** - Input credentials
- **[out] out_creds** - Output updated credentials

**return**
- **0 Success**
- **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.
**Kerberos Application Developer Guide, Release 1.11.1**

**krb5_get_init_creds_keytab** - Get initial credentials using a key table.

```c
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)
```

**Parameters**
- **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 NLUL)
- **[in] k5_gic_options** - Initial credential options

**Returns**
- **retval**
  - 0 Success
  - 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.

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

**Parameters**
- **param [in] context** - Library context
- **[out] opt** - New options structure

**Returns**
- **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.

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

**Parameters**
- **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.

```c
defined 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.

```c
defined 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.

```c
defined 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.

```c
defined 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.

```c
defined 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.

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

**Parameters**
- **[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.

```c
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)
```

**Parameters**
- **[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:** First introduced in 1.9

**krb5_get_init_creds_opt_set_fast_ccache** - Set FAST armor cache in initial credential options.

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

Chapter 5. Complete reference - API and datatypes
**Kerberos Application Developer Guide, Release 1.11.1**

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

documentation for `krb5_get_init_creds_opt_set_forwardable()`

### 5.1. krb5 API

---

**krb5_get_init_creds_opt_set_fast_flags()** - Set FAST flags in initial credential options.

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

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

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

---

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

```c
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)
```

### krb5_get_init_creds_opt_set_fast_flags - Set FAST flags in initial credential options.

```c
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] 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:** First introduced in 1.9

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.

The following flag values are valid:

- **KRB5_FAST_REQUIRED** - Require FAST to be used

---

5.1. krb5 API
Kerberos Application Developer Guide, Release 1.11.1

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.

```c
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:** First introduced in 1.11

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

```c
krb5_error_code krb5_get_init_creds_opt_set_outccache (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.

```c
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_preauth_list - Set preauthentication types in initial credential options.

```c
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.

```c
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.

```c
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_setResponder - Set the responder function in initial credential options.

```c
krb5_error_code krb5_get_init_creds_opt_setResponder (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:** First introduced in 1.11

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

```c
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

5.1. krb5 API
When getting initial credentials with a password, a salt string is 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.

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

Param [in] opt - Options structure

Param [in] tkt_life - Ticket lifetime

**krb5_get_init_creds_password** - Get initial credentials using a password.

```c
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

Param [out] creds - New credentials

Param [in] client - Client principal

Param [in] password - Password (or NULL)

Param [in] prompter - Prompter function

Param [in] data - Prompter callback data

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

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

Param [in] k5_gic_options - Initial credential options

Return

- 0 Success
- EINVAL Invalid argument
- KRB5_KDC_UNREACH Cannot contact any KDC for requested realm
- KRB5_PREAUTH_FAILED Generic Pre-authentication 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.**

```c
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
- 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.**

```c
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.**

```c
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
- 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.**

```c
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)

**Return**
- 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

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.**

```c
krb5_error_code krb5_init_context (krb5_context * context)
```

**Param**
- `[out] context` - Library context

**Return**
- 0 Success
- 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.

```c
krb5_error_code krb5_init_secure_context(krb5_context * context)
    param [out] context - Library context
    retval
        • 0 Success
        • 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.

```c
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.

```c
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.

```c
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.

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

**Note:** First introduced in 1.11

### krb5_kt_default - Resolve the default key table.

```c
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.

```c
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_NOTENUFFSPACE Buffer is too short

**Return:**
- Kerberos error codes

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

### krb5_kt_get_name - Get a key table name.

```c
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_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. The key table is not opened.

**Note:** 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.

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

**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.**

```c
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.**

```c
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.

```c
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.

```c
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.

```c
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

5.1. krb5 API
• **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.

```c
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`
- **[inout] prompts** - Array of output 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()`.

Writs *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.

```c
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.

```c
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

Note: First introduced in 1.11

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

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

param [in] ctx - Library context
    [in] rctx - Responder context

Note: First introduced in 1.11

**krb5_responder_set_answer** - Answer a named question in the responder context.

```c
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)

Note: First introduced in 1.11

**krb5_responder_otp_get_challenge** - Decode the KRB5_RESPONDER_QUESTION_OTP to a C struct.

```c
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: First introduced in 1.11

5.1. krb5 API
krb5_responder_otp_set_answer - Answer the KRB5_RESPONDER_QUESTION_OTP question.

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

- **ctx** - Library context
- **rctx** - Responder context
- **ti** - The index of the tokeninfo selected
- **value** - The value to set, or NULL for none
- **pin** - The pin to set, or NULL for none

**Note:** First introduced in 1.11

---

krb5_responder_otp_challenge_free - Free the value returned by krb5_responder_otp_get_challenge() .

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

- **ctx** - Library context
- **rctx** - Responder context
- **chl** - The challenge to free

**Note:** First introduced in 1.11

---

krb5_set_default_realm - Override the default realm for the specified context.

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

- **context** - Library context
- **lrealm** - Realm name for the default realm

**return**

- 0 Success
- Kerberos error codes

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

---

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

```c
krb5_error_code krb5_set_password (krb5_context context, krb5_creds * creds, 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.

5.1. 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, 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.**

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

- **param** `[inout] context` - Library context
- **param** `[in] principal` - Principal name
- **param** `[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.**

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

- **param** `[in] context` - Library context
- **param** `[in] fn` - Callback function
- **param** `[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:** First introduced in 1.9

**krb5_set_trace_filename - Specify a file name for directing trace events.**

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

- **param** `[in] context` - Library context
- **param** `[in] filename` - File name

Specify a file name for directing trace events.
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: First introduced 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.

5.1. krb5 API
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.

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

- **param** `context` - Library context
- **param** `principal` - Principal
- **param** `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.

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

- **param** `context` - Library context
- **param** `principal` - Principal
- **param** `name` - String representation of principal name
- **param** `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()`. 

---

44 Chapter 5. Complete reference - API and datatypes
**krb5_unparse_name_flags** - Convert krb5_principal structure to a string with flags.

```c
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.

```c
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.

```c
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

---

**5.1. krb5 API**
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.**

```c
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)
```

**Parameter Descriptions:**
- **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`.

### 5.1.2 Rarely used public interfaces

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

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

**Parameter Descriptions:**
- **param [in] context** - Library context
- **[in] name** - V4 name
- **[in] instance** - V4 instance
- **[in] realm** - Realm
- **[out] princ** - V5 principal

**Return Values:**
- 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.**

```c
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.**

```c
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.**

```c
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
  • < 0 First address is less than second
  • > 0 First address is greater than second

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

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

5.1. krb5 API
**Kerberos Application Developer Guide, Release 1.11.1**

```c
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.**

```c
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.**

```c
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.**

```c
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.

```c
const krb5_data * krb5_anonymous_realm (void 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.

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

### Parameters:
- **context**: Library context
- **appname**: Application name
- **realm**: Realm name
- **option**: Option to be checked
- **default_value**: Default value to return if no match is found
- **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_boolean()

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

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

### Parameters:
- **context**: Library context
- **appname**: Application name
- **realm**: Realm name
- **option**: Option to be checked
- **default_value**: Default value to return if no match is found
- **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()
**Kerberos Application Developer Guide, Release 1.11.1**

**krb5_auth_con_free - Free a krb5_auth_context structure.**

```c
krb5_error_code krb5_auth_con_free(krb5_context context, krb5_auth_context auth_context)
```

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

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

- **param [in] context** - Library context
- **param [in] auth_context** - Authentication context
- **param [in] infd** - Connected socket descriptor
- **param [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`.

- **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.**

```c
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
- **param [in] auth_context** - Authentication context
- **param [out] func** - Checksum callback
- **param [out] data** - Callback argument

**retval**
- 0 (always)
krb5_auth_con_getaddrs - Retrieve address fields from an auth context.

```c
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
- **param** [in] auth_context - Authentication context
- **param** [out] local_addr - Local address (NULL if not needed)
- **param** [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.**

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

- **param** [in] context - Library context
- **param** [in] auth_context - Authentication context
- **param** [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.**

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

- **param** [in] context - Library context
- **param** [in] auth_context - Authentication context
- **param** [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.

```c
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.

```c
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.

```c
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.

```c
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

c 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)

c param [in] ctx - Library context
  [in] ac - Authentication context
  [out] keyblock - Receiving subkey

c 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)

c param [in] ctx - Library context
  [in] ac - Authentication context
  [out] key - Receiving subkey

c 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)

c param [in] context - Library context
  [in] auth_context - Authentication context
  [out] seqnumber - Remote sequence number

c retval
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

- **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.

```c
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 auth context.

```c
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.

```c
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

[inout] 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 handle

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.

```c
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.

```c
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:** First introduced in 1.9

---

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

```c
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.

```c
krb5_error_code krb5_auth_con_setsendsubkey_k (krb5_context ctx, krb5_auth_context ac,
                                               krb5_key key)
```

---

5.1. krb5 API
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: First introduced 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: First introduced 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)
```

58 Chapter 5. Complete reference - API and datatypes
**krb5_cc_end_seq_get - Finish a series of sequential processing credential cache entries.**

```c
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

**[inout] 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.**

```c
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.**

```c
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 ** fullname_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:** First introduced in 1.10

---

**krb5_cc_last_change_time - Return a timestamp of the last modification to a credential cache.**

```
krb5_error_code krb5_cc_last_change_time ( krb5_context context, krb5_ccache ccache, krb5_timestamp * change_time )
```

- **param [in] context** - Library context
- **[in] ccache** - Credential cache handle
- **[out] change_time** - The last change time of `ccache`

If an error occurs, `change_time` is set to 0.

---

**krb5_cc_lock - Lock a credential cache.**

```
krb5_error_code krb5_cc_lock ( krb5_context context, krb5_ccache ccache )
```

- **param [in] context** - Library context
- **[in] ccache** - Credential cache handle

**retval**

* 0 Success; otherwise - Kerberos error codes

Use `krb5_cc_unlock()` to unlock the lock.

---

**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.**

```c
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
- [inout] 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.**

```c
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 Application Developer Guide, Release 1.11.1**

**Krb5_cc_retrieve_cred - Retrieve a specified credentials from a credential cache.**

```c
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 must 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.**

```c
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:** First introduced 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**
- [inout] context - Library context
- [in] name - Default credential cache name

**retval**
- 0 Success
- KV5M_CONTEXT Bad magic number for _krb5_context structure
• Kerberos error codes

This function frees the old default credential cache name and then sets it to \textit{name}.

\textbf{krb5\_cc\_set\_flags - Set options flags on a credential cache.}

\texttt{krb5\_error\_code krb5\_cc\_set\_flags (krb5\_context context, krb5\_ccache cache, krb5\_flags flags)}

\begin{itemize}
  \item \texttt{param [in] context - Library context}
  \item \texttt{ [inout] cache - Credential cache handle}
  \item \texttt{ [in] flags - Flag bit mask}
\end{itemize}

\texttt{retval}

\begin{itemize}
  \item 0 Success; otherwise - Kerberos error codes
\end{itemize}

This function resets \texttt{cache flags to flags}.

\textbf{krb5\_cc\_start\_seq\_get - Prepare to sequentially read every credential in a credential cache.}

\texttt{krb5\_error\_code krb5\_cc\_start\_seq\_get (krb5\_context context, krb5\_ccache cache, krb5\_cc\_cursor \*cursor)}

\begin{itemize}
  \item \texttt{param [in] context - Library context}
  \item \texttt{ [in] cache - Credential cache handle}
  \item \texttt{ [out] cursor - Cursor}
\end{itemize}

\texttt{retval}

\begin{itemize}
  \item 0 Success; otherwise - Kerberos error codes
\end{itemize}

\texttt{krb5\_cc\_end\_seq\_get ()} must be called to complete the retrieve operation.

\begin{itemize}
  \item \texttt{Note: If cache is modified between the time of the call to this function and the time of the final \texttt{krb5\_cc\_end\_seq\_get ()}, the results are undefined.}
\end{itemize}

\textbf{krb5\_cc\_store\_cred - Store credentials in a credential cache.}

\texttt{krb5\_error\_code krb5\_cc\_store\_cred (krb5\_context context, krb5\_ccache cache, krb5\_creds \*creds)}

\begin{itemize}
  \item \texttt{param [in] context - Library context}
  \item \texttt{ [inout] cache - Credential cache handle}
  \item \texttt{ [in] creds - Credentials to be stored in cache}
\end{itemize}

\texttt{retval}

\begin{itemize}
  \item 0 Success
\end{itemize}

\texttt{return}

\begin{itemize}
  \item Permission errors; storage failure errors; Kerberos error codes
\end{itemize}

This function stores \texttt{creds} into \texttt{cache}. If \texttt{creds->server} and the server in the decoded ticket \texttt{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.

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

**Parameters**
- `context` - Library context
- `type` - Credential cache type

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

**Note:** First introduced in 1.10

---

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

```c
krb5_error_code krb5_cc_switch (krb5_context context, krb5_ccache cache)
```

**Parameters**
- `context` - Library context
- `cache` - Credential cache handle

**Returns**
- 0 Success, or the type of cache doesn’t support switching
- 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_cc_unlock** - Unlock a credential cache.

```c
krb5_error_code krb5_cc_unlock (krb5_context context, krb5_ccache ccache)
```

**Parameters**
- `context` - Library context
- `ccache` - Credential cache handle

**Returns**
- 0 Success; otherwise - Kerberos error codes

This function unlocks the `ccache` locked by `krb5_cc_lock()`.

---

**krb5_cccol_cursor_free** - Free a credential cache collection cursor.

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

**Parameters**
- `context` - Library context
- `cursor` - Cursor

**Returns**
- 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
- **param [inout]** `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
- **param [inout]** `cursor` - Cursor
- **param [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:** First introduced in 1.11
**krb5_cccol_last_change_time** - Return a timestamp of the last modification of any known credential cache.

```c
krb5_error_code krb5_cccol_last_change_time(krb5_context context, krb5_timestamp change_time)
```

- **param** `[in]` `context` - Library context
- **param** `[out]` `change_time` - Last modification timestamp
- **retval**
  - 0 Success; otherwise - Kerberos error codes

This function returns the most recent modification time of any known credential cache, ignoring any caches which cannot supply a last modification time.

If there are no known credential caches, `change_time` is set to 0.

**krb5_cccol_lock** - Acquire a global lock for credential caches.

```c
krb5_error_code krb5_cccol_lock(krb5_context context)
```

- **param** `[in]` `context` - Library context
- **retval**
  - 0 Success; otherwise - Kerberos error codes

This function locks the global credential cache collection, ensuring that no ccaches are added to or removed from it until the collection lock is released.

Use `krb5_cccol_unlock()` to unlock the lock.

**krb5_cccol_unlock** - Release a global lock for credential caches.

```c
krb5_error_code krb5_cccol_unlock(krb5_context context)
```

- **param** `[in]` `context` - Library context
- **retval**
  - 0 Success; otherwise - Kerberos error codes

This function unlocks the lock from `krb5_cccol_lock()`.

**krb5_clear_error_message** - Clear the extended error message in a context.

```c
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.
**Kerberos Application Developer Guide, Release 1.11.1**

**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
- **param** [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:** First introduced 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
- **param** [in] inaddr - Array of addresses to be copied
- **param** [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
- **param** [in] in_authdat - List of `krb5_authdata` structures
- **param** [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.

```c
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.

```c
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.

```c
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.

```c
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

5.1. krb5 API
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.**

```c
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.**

```c
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.**

```c
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.**

```c
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.**

```c
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.**

```c
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.**

```c
krb5_error_code krb5_find_authdata(krb5_context context, krb5_authdata *const *ticket_authdata, krb5_authdata *const *ap_req_authdata, krb5_authdatatypetype 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.
Kerberos Application Developer Guide, Release 1.11.1

**Note:** First introduced in 1.10

### krb5_free_addresses - Free the data stored in array of addresses.

```c
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.

```c
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.

```c
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.

```c
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.

```c
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.**

```c
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.**

```c
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.**

```c
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() .**

```c
void krb5_free_default_realm (krb5_context context, char * lrealm) {
    param [in] context - Library context
    [in] lrealm - Realm to be freed
}
```

**krb5_free_error - Free an error allocated by krb5_read_error() or krb5_sendauth() .**

```c
void krb5_free_error (krb5_context context, register 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() .**

```c
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
}
```
Krb5_free_keyblock - Free a krb5_keyblock structure.

```c
void krb5_free_keyblock (krb5_context context, register krb5_keyblock * val)
```

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

```c
void krb5_free_keyblock_contents (krb5_context context, register krb5_keyblock * key)
```

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

```c
krb5_error_code krb5_free_keytab_entry_contents (krb5_context context, krb5_keytab_entry * entry)
```

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

```c
void krb5_free_string (krb5_context context, char * val)
```

- **param** [in] context - Library context
- **param** [in] val - String to be freed

Note: First introduced in 1.10
**krb5_free_ticket - Free a ticket.**

```c
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.**

```c
void krb5_free_unparsed_name (krb5_context context, char * val)
```

- **param [in] context** - Library context
- **[in] val** - Name string to be freed

**krb5_get_permitted_enctypes - Return a list of encryption types permitted for session keys.**

```c
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()`.

**krb5_get_server_rcache - Generate a replay cache object for server use and open it.**

```c
krb5_error_code krb5_get_server_rcache (krb5_context context, const krb5_data * piece, krb5_rcache * rcptr)
```

- **param [in] context** - Library context
- **[in] piece** - Unique identifier for replay cache
- **[out] rcptr** - Handle to an open rcache

- **retval**
  - 0 Success; otherwise - Kerberos error codes

This function generates a replay cache name based on `piece` and opens a handle to it. Typically `piece` is the first component of the service principal name. Use `krb5_rc_close()` to close `rcptr` when it is no longer needed.

**krb5_get_time_offsets - Return the time offsets from the os context.**

```c
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

5.1. krb5 API
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.

```c
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.

```c
void krb5_init_creds_free (krb5_context context, krb5_init_creds_context ctx)
```

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

**krb5_init_creds_get** - Acquire credentials using an initial credentials context.

```c
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()`.

**krb5_init_creds_get_creds** - Retrieve acquired credentials from an initial credentials context.

```c
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 \textit{ctx} into \textit{creds}, after the successful completion of \texttt{krb5_init_creds_get()} or \texttt{krb5_init_creds_step()}. Use \texttt{krb5_free_cred_contents()} to free \textit{creds} when it is no longer needed.

\textbf{krb5_init_creds_get_error} - Get the last error from KDC from an initial credentials context.

\begin{verbatim}
krb5_error_code krb5_init_creds_get_error(krb5_context context, krb5_init_creds_context ctx, krb5_error **error);
\end{verbatim}

- \texttt{context} - Library context
- \texttt{ctx} - Initial credentials context
- \texttt{error} - Error from KDC, or NULL if none was received

\texttt{retval}
- 0 Success; otherwise - Kerberos error codes

\textbf{krb5_init_creds_get_times} - Retrieve ticket times from an initial credentials context.

\begin{verbatim}
krb5_error_code krb5_init_creds_get_times(krb5_context context, krb5_init_creds_context ctx, krb5_ticket_times *times);
\end{verbatim}

- \texttt{context} - Library context
- \texttt{ctx} - Initial credentials context
- \texttt{times} - Ticket times for acquired credentials

\texttt{retval}
- 0 Success; otherwise - Kerberos error codes

The initial credentials context must have completed obtaining credentials via either \texttt{krb5_init_creds_get()} or \texttt{krb5_init_creds_step()}. 

\textbf{krb5_init_creds_init} - Create a context for acquiring initial credentials.

\begin{verbatim}
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);
\end{verbatim}

- \texttt{context} - Library context
- \texttt{client} - Client principal to get initial creds for
- \texttt{prompter} - Prompter callback
- \texttt{data} - Prompter callback argument
- \texttt{start_time} - Time when credentials become valid (0 for now)
- \texttt{options} - Options structure (NULL for default)
- \texttt{ctx} - New initial credentials context

\texttt{retval}
- 0 Success; otherwise - Kerberos error codes
This function creates a new context for acquiring initial credentials. Use \texttt{krb5_init_creds_free()} to free \texttt{ctx} when it is no longer needed.

\textbf{krb5_init_creds_set_keytab - Specify a keytab to use for acquiring initial credentials.}

\begin{verbatim}
krb5_error_code krb5_init_creds_set_keytab (krb5_context context, krb5_init_creds_context ctx, krb5_keytab keytab)
\end{verbatim}

\begin{itemize}
  \item param [in] context - Library context
  \item [in] ctx - Initial credentials context
  \item [in] keytab - Key table handle
\end{itemize}

\texttt{retval}

\begin{itemize}
  \item 0 Success; otherwise - Kerberos error codes
\end{itemize}

This function supplies a keytab containing the client key for an initial credentials request.

\textbf{krb5_init_creds_set_password - Set a password for acquiring initial credentials.}

\begin{verbatim}
krb5_error_code krb5_init_creds_set_password (krb5_context context, krb5_init_creds_context ctx, const char * password)
\end{verbatim}

\begin{itemize}
  \item param [in] context - Library context
  \item [in] ctx - Initial credentials context
  \item [in] password - Password
\end{itemize}

\texttt{retval}

\begin{itemize}
  \item 0 Success; otherwise - Kerberos error codes
\end{itemize}

This function supplies a password to be used to construct the client key for an initial credentials request.

\textbf{krb5_init_creds_set_service - Specify a service principal for acquiring initial credentials.}

\begin{verbatim}
krb5_error_code krb5_init_creds_set_service (krb5_context context, krb5_init_creds_context ctx, const char * service)
\end{verbatim}

\begin{itemize}
  \item param [in] context - Library context
  \item [in] ctx - Initial credentials context
  \item [in] service - Service principal string
\end{itemize}

\texttt{retval}

\begin{itemize}
  \item 0 Success; otherwise - Kerberos error codes
\end{itemize}

This function supplies a service principal string to acquire initial credentials for instead of the default krbtgt service. \textit{service} is parsed as a principal name; any realm part is ignored.

\textbf{krb5_init_creds_step - Get the next KDC request for acquiring initial credentials.}

\begin{verbatim}
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)
\end{verbatim}
### krb5_init_keyblock - Initialize an empty krb5_keyblock

The function `krb5_init_keyblock` initializes a new keyblock and allocates 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.

#### Parameters
- `context` [in] Library context
- `enctype` [in] Encryption type
- `length` [in] Length of keyblock (or 0)
- `out` [out] New keyblock structure

#### Return Value
- 0 Success; otherwise - Kerberos error codes

#### Note
If `length` is set to 0, contents are left unallocated.

### krb5_is_referral_realm - Check for a match with KRB5_REFERRAL_REALM

The function `krb5_is_referral_realm` checks if the given realm matches KRB5_REFERRAL_REALM.

#### Parameters
- `r` [in] Realm to check

#### Return Value
- TRUE if `r` is zero-length, FALSE otherwise

#### Example
```c
const krb5_data *r;
krb5_boolean result = krb5_is_referral_realm(r);
```
**Kerberos Application Developer Guide, Release 1.11.1**

**krb5_kt_add_entry** - Add a new entry to a key table.

```c
krb5_error_code krb5_kt_add_entry (krb5_context context, krb5_keytab id, krb5_keytab_entry * entry)
```

<table>
<thead>
<tr>
<th>param</th>
<th>[in] context - Library context</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[in] id - Key table handle</td>
</tr>
<tr>
<td></td>
<td>[in] entry - Entry to be added</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>retval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Success</td>
</tr>
<tr>
<td>ENOMEM Insufficient memory</td>
</tr>
<tr>
<td>KRB5_KT_NOWRITE Key table is not writeable</td>
</tr>
</tbody>
</table>

**return**

- Kerberos error codes

**krb5_kt_end_seq_get** - Release a keytab cursor.

```c
krb5_error_code krb5_kt_end_seq_get (krb5_context context, krb5_keytab keytab, krb5_kt_cursor * cursor)
```

<table>
<thead>
<tr>
<th>param</th>
<th>[in] context - Library context</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[in] keytab - Key table handle</td>
</tr>
<tr>
<td></td>
<td>[out] cursor - Cursor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>retval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Success</td>
</tr>
</tbody>
</table>

**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.

```c
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)
```

<table>
<thead>
<tr>
<th>param</th>
<th>[in] context - Library context</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[in] keytab - Key table handle</td>
</tr>
<tr>
<td></td>
<td>[in] principal - Principal name</td>
</tr>
<tr>
<td></td>
<td>[in] vno - Key version number (0 for highest available)</td>
</tr>
<tr>
<td></td>
<td>[in] enctype - Encryption type (0 zero for any enctype)</td>
</tr>
<tr>
<td></td>
<td>[out] entry - Returned entry from key table</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>retval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Success</td>
</tr>
<tr>
<td>Kerberos error codes on failure</td>
</tr>
</tbody>
</table>
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.

```c
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:** First introduced in 1.11

### krb5_kt_next_entry - Retrieve the next entry from the key table.

```c
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
- `[inout] 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`.

### krb5_kt_read_service_key - Retrieve a service key from a key table.

```c
krb5_kt_read_service_key (krb5_context context, krb5_keytab keytab, krb5_keytab_entry * entry, 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 *pcreds, krb5_data **ppdata, krb5_replay_data *outdata)

param [in] context - Library context
[inout] auth_context - Authentication context
[in] pcreds - Pointer to credentials
[out] ppdata - Encoded credentials
[out] outdata - Replay cache data (NULL if not needed)

retval
• 0 Success
This is a convenience function that calls \texttt{krb5\_mk\_ncred()} with a single set of credentials.

### \texttt{krb5\_mk\_error} - Format and encode a KRB\_ERROR message.

\begin{verbatim}
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
\end{verbatim}

This function creates a KRB\_ERROR message in \texttt{enc\_err}. Use \texttt{krb5\_free\_data\_contents()} to free \texttt{enc\_err} when it is no longer needed.

### \texttt{krb5\_mk\_ncred} - Format a KRB\_CRED message for an array of credentials.

\begin{verbatim}
krb5_error_code krb5_mk_ncred (krb5_context context, krb5_auth_context auth_context, krb5_creds ** ppcreds, krb5_data ** ppdata, krb5_replay_data * outdata)

param [in] context - Library context
 [inout] auth_context - Authentication context
 [in] ppcreds - Null-terminated array of credentials
 [out] ppdata - Encoded credentials
 [out] outdata - 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
\end{verbatim}

This function takes an array of credentials \texttt{ppcreds} and formats a KRB\_CRED message \texttt{ppdata} to pass to \texttt{krb5\_rd\_cred()}. The message will be encrypted using the send subkey of \texttt{auth\_context} if it is present, or the session key otherwise.

**Note:** If the KRB5\_AUTH\_CONTEXT\_RET\_TIME or KRB5\_AUTH\_CONTEXT\_RET\_SEQUENCE flag is set in \texttt{auth\_context}, \texttt{outdata} is required.
**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 *outbuf, krb5_replay_data *outdata)`

- **param** [in] `context` - Library context
- **[inout]** `auth_context` - Authentication context
- **[in]** `userdata` - User data for KRB-PRIV message
- **[out]** `outbuf` - Formatted KRB-PRIV message
- **[out]** `outdata` - Replay cache handle (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-SAFE message. The remote address is optional; if specified, it will be used to form the receiver address used in the message.

- **KRB5_AUTH_CONTEXT_DO_TIME** - Use timestamps in `outdata`
- **KRB5_AUTH_CONTEXT_RET_TIME** - Copy timestamp to `outdata`
- **KRB5_AUTH_CONTEXT_DO_SEQUENCE** - Use local sequence numbers from `auth_context` in replay cache.
- **KRB5_AUTH_CONTEXT_RET_SEQUENCE** - Use local sequence numbers from `auth_context` as a sequence number in the encrypted message `outbuf`.

**Note:** If the `KRB5_AUTH_CONTEXT_RET_TIME` or `KRB5_AUTH_CONTEXT_RET_SEQUENCE` flag is set in `auth_context`, the `outdata` is required.

The flags from `auth_context` specify whether sequence numbers or timestamps will be used to identify the message. Valid values are:

---

**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
- **[inout]** `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.

```c
krb5_error_code krb5_mk_rep_dce (krb5_context context, krb5_auth_context auth_context, krb5_data * outbuf)
```

**Parameters**
- **context** [in] - Library context
- **auth_context** [inout] - Authentication context
- **outbuf** [out] - AP-REP message

**Return Value**
- 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.

```c
krb5_error_code krb5_mk_req (krb5_context context, krb5_auth_context * auth_context, krb5_flags ap_req_options, char * service, char * hostname, krb5_data * in_data, krb5_ccache ccache, krb5_data * outbuf)
```

**Parameters**
- **context** [in] - Library context
- **auth_context** [inout] - Pre-existing or newly created auth context
- **ap_req_options** [in] - AP_OPTS options
- **service** [in] - Service name, or NULL to use “host”
- **hostname** [in] - Host name, or NULL to use local hostname
- **in_data** [in] - Application data to be checksummed in the authenticator, or NULL
- **ccache** [in] - Credential cache used to obtain credentials for the desired service.
- **outbuf** [out] - AP-REQ message

**Return Value**
- 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.

```c
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)
```

**Parameters**
- **context** [in] - Library context
- **auth_context** [inout] - Pre-existing or newly created auth context
- **ap_req_options** [in] - AP_OPTS options
- **in_data** [in] - Application data to be checksummed in the authenticator, or NULL
- **in_creds** [in] - Credentials for the service with valid ticket and key
- **outbuf** [out] - AP-REQ message

86 Chapter 5. Complete reference - API and datatypes
Kerberos Application Developer Guide, Release 1.11.1

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 reciever.
- **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.

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 KRB5_AUTH_CONTEXT_DO_TIME flag is set in the *auth_context*, an entry describing the message is entered in the replay cache *auth_context->rcache* which enables the caller to detect if this message is reflected by an attacker. If KRB5_AUTH_CONTEXT_DO_TIME is not set, the replay cache is not used.

If either KRB5_AUTH_CONTEXT_DO_SEQUENCE or KRB5_AUTH_CONTEXT_RET_SEQUENCE is set, the *auth_context* local sequence number will be placed in *outdata* as its sequence number.

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

Note: The *outdata* argument is required if KRB5_AUTH_CONTEXT_RET_TIME or KRB5_AUTH_CONTEXT_RET_SEQUENCE flag is set in the *auth_context*.

5.1. krb5 API
**Kerberos Application Developer Guide, Release 1.11.1**

---

**krb5_os_localaddr** - Return all interface addresses for this host.

```c
krb5_error_code krb5_os_localaddr (krb5_context context, krb5_address *** addr)
```

**Parameters:**
- **context** - Library context
- **addr** - Array of krb5_address pointers, ending with NULL

**Return:**
- 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.

```c
krb5_error_code krb5_pac_add_buffer (krb5_context context, krb5_pac pac, krb5_ui_4 type, const krb5_data * data)
```

**Parameters:**
- **context** - Library context
- **pac** - PAC handle
- **type** - Buffer type
- **data** - contents

**Return:**
- 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.

```c
void krb5_pac_free (krb5_context context, krb5_pac pac)
```

**Parameters:**
- **context** - Library context
- **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.

```c
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.

```c
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.

```c
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.

```c
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**
Use `krb5_pac_free()` to free `pac` when it is no longer needed.

**krb5_pac_sign - Sign a PAC.**

```c
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:** First introduced in 1.10

**krb5_pac_verify - Verify a PAC.**

```c
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 Apple Mac OS X 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_principal2salt** - Convert a principal name into the default salt for that principal.

```
krb5_error_code krb5_principal2salt (krb5_context context, register 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 * pcred-data,
    krb5_creds *** pppcreds, krb5_replay_data * outdata)
```

- **param** [in] context - Library context
- **[inout]** auth_context - Authentication context
- **[in]** pcred-data - KRB-CRED message
- **[out]** pppcreds - Null-terminated array of forwarded credentials
- **[out]** outdata - Replay data (NULL if not needed)

**retval**

- 0 Success; otherwise - Kerberos error codes

`pcred-data` 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 `pppcreds` when it is no longer needed.

**Note:** The `outdata` argument is required if `KRB5_AUTH_CONTEXT_RET_TIME` or `KRB5_AUTH_CONTEXT_RET_SEQUENCE` flag is set in the `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.**

```c
krb5_error_code krb5_rd_priv(krb5_context context, krb5_auth_context auth_context, const krb5_data *inbuf, krb5_data *outbuf, krb5_replay_data *outdata)
```

**param**

- **[in]** context - Library context
- **[inout]** auth_context - Authentication structure
- **[in]** inbuf - KRB-PRIV message to be parsed
- **[out]** outbuf - Data parsed from KRB-PRIV message
- **[out]** outdata - 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 `outbuf`.

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 two additional checks are performed:

- The timestamp in the message must be within the permitted clock skew (which is usually five minutes).
- The message must not be a replayed message field in `auth_context`.

**Note:** If the `KRB5_AUTH_CONTEXT_RET_TIME` or `KRB5_AUTH_CONTEXT_RET_SEQUENCE` flag is set in `auth_context`, `outdata` is required.

`auth_context` must have a remote address set. This 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. Both addresses must use type `ADDRTYPE_ADDRPORT`.

---

**krb5_rd_rep - Parse and decrypt a KRB_AP_REP message.**

```c
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
- **[inout]** 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.
**Kerberos Application Developer Guide, Release 1.11.1**

**krb5_rd_rep_dce** - Parse and decrypt a KRB_AP_REP message for DCE RPC.

```c
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
- **[inout] 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.

```c
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 is present in the `auth_context`, it is used to decrypt the ticket in AP-REQ message. (This is useful for user-to-user authentication.) Otherwise, the decryption key is obtained from the `keytab`. If `keytab` is iterable, all of its key entries it will be tried against the ticket; otherwise, the server principal in the ticket will be looked up in the keytab and that key will be tried.

The client specified in the decrypted authenticator must match the client specified in the decrypted ticket. If `server` is non-null, the key in which the ticket is encrypted must correspond to a principal in `keytab` matching `server` according to the rules of `krb5_sname_match()`.

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.

5.1. krb5 API
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.**

```c
krb5_error_code krb5_rd_safe (krb5_context context, krb5_auth_context auth_context, const krb5_data *inbuf, krb5_data *outbuf, krb5_replay_data *outdata)
```

- **param**
  - `[in] context` - Library context
  - `[inout] auth_context` - Authentication structure
  - `[in] inbuf` - KRB-SAFE message to be parsed
  - `[out] outbuf` - Data parsed from KRB-SAFE message
  - `[out] outdata` - 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 `outbuf`.

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 two additional checks are performed:

- The timestamp in the message must be within the permitted clock skew (which is usually five minutes).
- The message must not be a replayed message field in `auth_context`.

Use `krb5_free_data_contents()` to free `outbuf` when it is no longer needed.

---

**Note:** The `outdata` argument is required if `KRB5_AUTH_CONTEXT_RET_TIME` or `KRB5_AUTH_CONTEXT_RET_SEQUENCE` flag is set in the `auth_context`.

`auth_context` must have a remote address set. This 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. Both addresses must use type `ADDRTYPE_ADDRPORT`.

---

**krb5_read_password - Read a password from keyboard input.**

```c
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.

```c
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.

```c
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
- [inout] `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.

```c
krb5_error_code krb5_set_default_tgs_enctypes (krb5_context context, const krb5_enctype *etypes)
```

**param**
- [inout] `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.**

```c
void krb5_set_error_message (krb5_context ctx, krb5_error_code code, const char * fmt, ...)
```

<table>
<thead>
<tr>
<th>param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] ctx</td>
<td>Library context</td>
</tr>
<tr>
<td>[in] code</td>
<td>Error code</td>
</tr>
<tr>
<td>[in] fmt</td>
<td>Error string for the error code</td>
</tr>
</tbody>
</table>

**krb5_set_real_time - Set time offset field in a krb5_context structure.**

```c
krb5_error_code krb5_set_real_time (krb5_context context, krb5_timestamp seconds, krb5_int32 microseconds)
```

<table>
<thead>
<tr>
<th>param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] context</td>
<td>Library context</td>
</tr>
<tr>
<td>[in] seconds</td>
<td>Real time, seconds portion</td>
</tr>
<tr>
<td>[in] microseconds</td>
<td>Real time, microseconds portion</td>
</tr>
</tbody>
</table>

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.**

```c
krb5_error_code krb5_string_to_cksumtype (char * string, krb5_cksumtype * cksumtypep)
```

<table>
<thead>
<tr>
<th>param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] string</td>
<td>String to be converted</td>
</tr>
</tbody>
</table>

[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.**

```c
krb5_error_code krb5_string_to_deltat (char * string, krb5_deltat * deltap)
```

<table>
<thead>
<tr>
<th>param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] string</td>
<td>String to be converted</td>
</tr>
</tbody>
</table>

[out] deltap - 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 - EINV

**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 - EINV

**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 - EINV

**krb5_timeofday - Retrieve the current time with context specific time offset adjustment.**

`krb5_error_code krb5_timeofday (krb5_context context, register krb5_timestamp * timeret)`

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

```c
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.

```c
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.

```c
void krb5_tkt_creds_free (krb5_context context, krb5_tkt_creds_context ctx)
```

**param**

- [in] context - Library context
- [in] ctx - TGS request context

**Note:** First introduced in 1.9

**krb5_tkt_creds_get** - Synchronously obtain credentials using a TGS request context.

```c
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 \texttt{krb5_tkt_creds_init()} . On successful return, the credentials can be retrieved with \texttt{krb5_tkt_creds_get_creds()}.

\textbf{Note:} First introduced in 1.9

\textbf{krb5\textunderscore tkt\textunderscore creds\textunderscore get\textunderscore creds} - Retrieve acquired credentials from a TGS request context.

\texttt{krb5\_error\_code \texttt{krb5\_tkt\_creds\_get\_creds}(\texttt{krb5\_context \ context, \ krb5\_tkt\_creds\_context \ ctx, \ krb5\_creds * \ creds)}

\textbf{param} \ [in] \ context - Library context
\[in\] \ ctx - TGS request context
\[out\] \ creds - Acquired credentials

\textbf{retval}
- \ 0 Success; otherwise - Kerberos error codes

This function copies the acquired initial credentials from \texttt{ctx} into \texttt{creds} , after the successful completion of \texttt{krb5\_tkt\_creds\_get()} or \texttt{krb5\_tkt\_creds\_step()} . Use \texttt{krb5\_free\_cred\_contents()} to free \texttt{creds} when it is no longer needed.

\textbf{Note:} First introduced in 1.9

\textbf{krb5\textunderscore tkt\textunderscore creds\textunderscore get\textunderscore times} - Retrieve ticket times from a TGS request context.

\texttt{krb5\_error\_code \texttt{krb5\_tkt\_creds\_get\_times}(\texttt{krb5\_context \ context, \ krb5\_tkt\_creds\_context \ ctx, \ krb5\_ticket\_times * \ times)}

\textbf{param} \ [in] \ context - Library context
\[in\] \ ctx - TGS request context
\[out\] \ times - Ticket times for acquired credentials

\textbf{retval}
- \ 0 Success; otherwise - Kerberos error codes

The TGS request context must have completed obtaining credentials via either \texttt{krb5\_tkt\_creds\_get()} or \texttt{krb5\_tkt\_creds\_step()}.

\textbf{Note:} First introduced in 1.9

\textbf{krb5\textunderscore tkt\textunderscore creds\textunderscore init} - Create a context to get credentials from a KDC's Ticket Granting Service.

\texttt{krb5\_error\_code \texttt{krb5\_tkt\_creds\_init}(\texttt{krb5\_context \ context, \ krb5\_ccache \ ccache, \ krb5\_creds * \ creds, \ krb5\_flags \ options, \ krb5\_tkt\_creds\_context * \ ctx)}

\textbf{param} \ [in] \ context - Library context
\[in\] \ ccache - Credential cache handle
\[in\] \ creds - Input credentials

\textbf{5.1. krb5 API}
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: First introduced in 1.9

---

### krb5_tkt_creds_step - Get the next KDC request in a TGS exchange.

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: First introduced in 1.9

---

### krb5_verify_init_creds - Verify initial credentials against a keytab.

This function verifies the initial credentials against the keytab, allowing the caller to control the transport of KDC requests and replies.

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 any other error, the TGS exchange has failed.

Note: First introduced in 1.9
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)
  [inout] 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.**

```c
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.**

```c
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_vset_error_message - Set an extended error message for an error code using a va_list.**

```c
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
```

5.1. krb5 API
5.1.3 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
- **param** [in] enctype - Encryption type
- **param** [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
- **param** [in] cksumtype - Checksum type
- **param** [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
- **param** [in] enctype - Encryption type
- **param** [in] type - Type field (See KRB5_CRYPTO_TYPE types)
- **param** [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
- **param** [in] enctype - Encryption type
- **param** [inout] data - IOV array
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).

```c
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
- **param** `[in] key` - Encryption key
- **param** `[in] usage` - Key usage (see `KRB5_KEYUSAGE` types)
- **param** `[inout] cipher_state` - Cipher state; specify NULL if not needed
- **param** `[in] input` - Encrypted data
- **param** `[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).

```c
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
- **param** `[in] keyblock` - Encryption key
- **param** `[in] usage` - Key usage (see `KRB5_KEYUSAGE` types)
- **param** `[in] cipher_state` - Cipher state; specify NULL if not needed
- **param** `[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.

This function is similar to `krb5_k_decrypt_iov()`, but operates on keyblock `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_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

[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.

This function is similar to krb5_k_encrypt_iov(), but operates on keyblock keyblock.

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().

```c
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.

```c
krb5_error_code krb5_c_fx_cf2_simple (krb5_context context, krb5_keyblock * k1, const char * pepper1, 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.

```c
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
Kerberos Application Developer Guide, Release 1.11.1

**Krb5 C API 107**

**Krb5 C Is Coll Proof Cksum** - Test whether a checksum type is collision-proof.

```c
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.

```c
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.

```c
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.

```c
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
**Kerberos Application Developer Guide, Release 1.11.1**

**krb5_c_make_checksum - Compute a checksum (operates on keyblock).**

```c
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)**

```c
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.

```c
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.

```c
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.

```c
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 prereinitialize `output` and allocate space for the result, using `krb5_c_prf_length()` to determine the required length.
krb5_c_prf_length - Get the output length of pseudo-random functions for an encryption type.

\[ \text{krb5_error_code krb5_c_prf_length (krb5\_context context, krb5\_enctype enctype, size\_t * len)} \]

- \text{param [in] context - Library context}
- \text{[in] enctype - Encryption type}
- \text{[out] len - Length of PRF output}

\[ \text{retval} \]
- \text{• 0 Success; otherwise - Kerberos error codes}

krb5_c_random_add_entropy - Add entropy to the pseudo-random number generator.

\[ \text{krb5_error_code krb5_c_random_add_entropy (krb5\_context context, unsigned int randsource, const krb5\_data * data)} \]

- \text{param [in] context - Library context}
- \text{[in] randsource - Entropy source (see KRB5\_RANDSOURCE types)}
- \text{[in] data - Data}

\[ \text{retval} \]
- \text{• 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.

\[ \text{krb5_error_code krb5_c_random_make_octets (krb5\_context context, krb5\_data * data)} \]

- \text{param [in] context - Library context}
- \text{[out] data - Random data}

\[ \text{retval} \]
- \text{• 0 Success; otherwise - Kerberos error codes}

Fills in \text{data} with bytes from the PRNG used by krb5 crypto operations. The caller must preinitialize \text{data} and allocate the desired amount of space.

krb5_c_random_os_entropy - Collect entropy from the OS if possible.

\[ \text{krb5_error_code krb5_c_random_os_entropy (krb5\_context context, int strong, int * success)} \]

- \text{param [in] context - Library context}
- \text{[in] strong - Strongest available source of entropy}
- \text{[out] success - 1 if OS provides entropy, 0 otherwise}

\[ \text{retval} \]
- \text{• 0 Success; otherwise - Kerberos error codes}

If \text{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.

```c
krb5_error_code krb5_c_random_to_key (krb5_context context, krb5_enctype enctype, krb5_data * random_data, krb5_keyblock * k5_random_key)
```

**Parameters**
- **[in] context** - Library context
- **[in] enctype** - Encryption type
- **[in] random_data** - Random input data
- **[out] k5_random_key** - Resulting key

**Return Values**
- **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.

```c
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)
```

**Parameters**
- **[in] context** - Library context
- **[in] enctype** - Encryption type
- **[in] string** - String to be converted
- **[in] salt** - Salt value
- **[out] key** - Generated key

**Return Values**
- **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.

```c
krb5_error_code krb5_c_string_to_key_with_params (krb5_context context, krb5_enctype enctype, const krb5_data * string, const krb5_data * salt, const krb5_data * params, krb5_keyblock * key)
```

**Parameters**
- **[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. 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).

```c
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
- **param** [in] cksumtype - Checksum type (0 for mandatory type)
- **param** [in] key - Encryption key for a keyed checksum
- **param** [in] usage - Key usage (see KRB5_KEYUSAGE types)
- **param** [in] data - IOV array
- **param** [in] num_data - Size of data
- **param** [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.

```c
krb5_error_code krb5_cksumtype_to_string (krb5_cksumtype cksumtype, char * buffer, size_t buflen)
```

- **param** [in] cksumtype - Checksum type
- **param** [out] buffer - Buffer to hold converted checksum type
- **param** [in] buflen - Storage available in buffer

**retval**

- 0 Success; otherwise - Kerberos error codes

---

**krb5_decode_authdata_container** - Unwrap authorization data.

```c
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
- **param** [in] type - KRB5_AUTHDATA type of `container`
- **param** [in] container - Authorization data to be decoded
- **param** [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.

```c
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.

```c
krb5_error_code krb5_deltat_to_string (krb5_deltat deltat, char * buffer, size_t buflen)

param [in] deltat - Relative time value to convert
[in] 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.

```c
krb5_error_code krb5_encode_authdata_container (krb5_context context, krb5_authdatatype type,
krb5_auhtdata * 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.

```c
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:** First introduced in 1.9

---

**krb5_enctype_to_string** - Convert an encryption type to a string.

```c
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.

```c
void krb5_free_checksum(krb5_context context, register 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.

```c
void krb5_free_checksum_contents(krb5_context context, register 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.

```c
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.

```c
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.

```c
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).

```c
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 \textit{input} and stores the output into \textit{output}. The actual decryption key will be derived from \textit{key} and \textit{usage} if key derivation is specified for the encryption type. If non-null, \textit{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.

\textbf{Note:} The caller must initialize \textit{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 \texttt{krb5_c_decrypt()} trim \textit{output->length}. For some enctypes, the resulting \textit{output->length} may include padding bytes.

\textbf{krb5_k_decrypt_iov - Decrypt data in place supporting AEAD (operates on opaque key).}

\begin{verbatim}
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
    [in] num_data - Size of data

retval
    • 0 Success; otherwise - Kerberos error codes
\end{verbatim}

This function decrypts the data block \textit{data} and stores the output in-place. The actual decryption key will be derived from \textit{key} and \textit{usage} if key derivation is specified for the encryption type. If non-null, \textit{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:

\texttt{krb5_k_encrypt_iov()}  

\textbf{Note:} On return from a \texttt{krb5_c_decrypt_iov()} call, the \textit{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.

This function is similar to \texttt{krb5_c_decrypt_iov()}, but operates on opaque key \textit{key}.

\textbf{krb5_k_encrypt - Encrypt data using a key (operates on opaque key).}

\begin{verbatim}
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

\end{verbatim}

5.1. krb5 API 117
Kerberos Application Developer Guide, Release 1.11.1

[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).

void 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
[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.

This function is similar to krb5_c_encrypt_iov(), but operates on opaque key key.

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_encrypt** - Retrieve the enctype of a krb5_key structure.

```c
krb5_enctype krb5_k_key_encrypt(krb5_context context, krb5_key key)
```

- **param context** - Library context
- **key**

**krb5_k_key_keyblock** - Retrieve a copy of the keyblock from a krb5_key structure.

```c
krb5_error_code krb5_k_key_keyblock(krb5_context context, krb5_key key, krb5_keyblock **key_data)
```

- **param context** - Library context
- **key**
- **key_data**

**krb5_k_make_checksum** - Compute a checksum (operates on opaque key).

```c
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
- **cksumtype** - Checksum type (0 for mandatory type)
- **key** - Encryption key for a keyed checksum
- **usage** - Key usage (see KRB5_KEYUSAGE types)
- **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)

```c
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)
```

5.1. krb5 API
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).

```c
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
- **param** [in] `key` - Encryption key for a keyed checksum
- **param** [in] `usage` - Key usage
- **param** [in] `data` - Data to be used to compute a new checksum using `key` to compare `cksum` against
- **param** [in] `cksum` - Checksum to be verified
- **param** [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. 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).

```c
krb5_error_code krb5_k_verify_checksum_iov(krb5_context context, krb5_cksumtype cksum-type, krb5_key key, krb5_keyusage usage, const krb5_crypto_iov * data, size_t num_data, krb5_boolean * valid)
```

- **param** [in] `context` - Library context
- **param** [in] `cksum-type` - Checksum type (0 for mandatory type)
- **param** [in] `key` - Encryption key for a keyed checksum
- **param** [in] `usage` - Key usage (see `KRB5_KEYUSAGE` types)
- **param** [in] `data` - IOV array
- **param** [in] `num_data` - Size of `data`
- **param** [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`.

---

5.1. krb5 API 121
5.1.4 Legacy convenience interfaces

**krb5_recvauth** - Server function for sendauth protocol.

```c
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
- **param [in]** auth_context - Authentication context
- **param [in]** fd - File descriptor
- **param [in]** appl_version - Application protocol version to be matched against the client’s application version
- **param [in]** server - Server principal (NULL for any in keytab)
- **param [in]** flags - Additional specifications
- **param [in]** keytab - Key table containing service keys
- **param [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.

```c
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
- **param [in]** auth_context - Authentication context
- **param [in]** fd - File descriptor
- **param [in]** server - Server principal (NULL for any in keytab)
- **param [in]** flags - Additional specifications
- **param [in]** keytab - Decryption key
- **param [out]** ticket - Ticket (NULL if not needed)
- **param [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.

```c
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` - Authentication 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()`
5.1.5 Deprecated public interfaces

krb5_524_convert_creds - Convert a Kerberos V5 credentials to a Kerberos V4 credentials.

```c
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
```

```c
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
```

```c
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
```

```c
krb5_error_code krb5_auth_con_initivector (krb5_context context, krb5_auth_context auth_context)
```

- param context
- auth_context

DEPRECATED Not replaced.

RFC 4120 doesn’t have anything like the initvector concept; only really old protocols may need this API.
**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**

```c
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**

```c
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**

```c
krb5_get_in_tkt_with_keytab (krb5_context context, krb5_flags options, krb5_address *addr, krb5_enctype *ktypes, krb5_preauthtype *pre_auth_types, krb5_keytab keytab, krb5_ccache ccache, krb5_creds *creds, krb5_kdc_rep **ret_as_reply)
```

param context
  - options
  - addr
  - ktypes
  - pre_auth_types
  - keytab
  - ccache
  - creds
  - ret_as_reply

DEPRECATED Replaced by krb5_get_init_creds_keytab() .

**krb5_get_init_creds_opt_init**

```c
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**

```c
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**

```c
krb5_kt_free_entry (krb5_context context, krb5_keytab_entry *entry)
```

param context
  - entry

DEPRECATED Use krb5_free_keytab_entry_contents instead.

---

5.1. krb5 API
**krb5_random_key**

```c
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**

```c
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**

```c
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**

```c
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**

```c
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()`

### 5.2 krb5 types and structures

#### 5.2.1 Public

**krb5_address**

```c
typedef struct _krb5_address krb5_address
```

**Structure for address.**

**Declaration**

```c
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**

```c
typedef krb5_int32 krb5_addrtype
```

**Declaration**

```c
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 representation 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.

5.2. krb5 types and structures
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
    authorization 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**

**krb5_const_pointer**

**Declaration**

typedef void const* krb5_const_pointer

**krb5_const_principal**

**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**

**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.

5.2. krb5 types and structures
krb5_cred_enc_part

Cleartext credentials information.

**Declaration**

```c
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

Credentials information inserted into `EncKrbCredPart`.

**Declaration**

```c
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 addr (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**

<table>
<thead>
<tr>
<th>krb5_cryptotype krb5_crypto_iov.flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRB5_CRYPTO_TYPE type of the iov</td>
</tr>
</tbody>
</table>

| krb5_data krb5_crypto_iov.data |

**krb5_cryptotype**

**Declaration**

typedef krb5_int32 krb5_cryptotype

**krb5_data**

**Declaration**

typedef struct _krb5_data krb5_data

**Members**

<table>
<thead>
<tr>
<th>krb5_magic krb5_data.magic</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int krb5_data.length</td>
</tr>
<tr>
<td>char * krb5_data.data</td>
</tr>
</tbody>
</table>

**krb5_deltat**

**Declaration**

typedef krb5_int32 krb5_deltat
**krb5_enc_data**

**Declaration**

```c
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**

**C representation of EncKDCRepPart protocol message.**

This is the cleartext message that is encrypted and inserted in KDC-REP.

**Declaration**

```c
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 addr, optional.

krb5_pa_data **krb5_enc_kdc_rep_part.enc_padata**
Encrypted preauthentication data.

**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**

**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.

5.2. krb5 types and structures
krb5_error_code

Declaration

typedef krb5_int32 krb5_error_code

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

Declaration

typedef krb5_int32 krb5_flags

krb5_get_init_creds_opt

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
Kerberos Application Developer Guide, Release 1.11.1

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**

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_int32**

krb5_int32

krb5_int32 is a signed 32-bit integer type

**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.

5.2. krb5 types and structures
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

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**

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**

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.

5.2. krb5 types and structures
krb5_keyusage

Declaration

typedef krb5_int32 krb5_keyusage

krb5_kt_cursor

Declaration

typedef krb5_pointer krb5_kt_cursor

krb5_kvno

Declaration

typedef unsigned int krb5_kvno

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
typedef krb5_error_code krb5_magic

**krb5_mk_req_checksumfunc**

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**

**Declaration**

typedef unsigned int krb5_msgtype

**krb5_octet**

**Declaration**

typedef unsigned char krb5_octet

**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**

**Declaration**

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**

Declaration

typedef void* krb5_pointer

**krb5_preauthtype**

Declaration

typedef krb5_int32 krb5_preauthtype

**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**

Declaration

typedef struct krb5_principal_data krb5_principal_data

5.2. krb5 types and structures
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_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_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**

**Declaration**

typedef krb5_int32 krb5_prompt_type

**krb5_prompter_fct**

**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**

**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_fn**

**Declaration**

typedef krb5_error_code( *krb5_responder_fn)(krb5_context ctx, void *data, krb5_responder_context rctx)

5.2. krb5 types and structures

151
**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**

**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_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

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

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
kerb5_enc_data krb5_ticket.enc_part
    encryption type, kvno, encrypted encoding

kerb5_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**

**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_authentication
krb5 Flags krb5_tkt_authent_ap_options

krb5_trace_callback

Declaration

typedef void( * krb5_trace_callback)(krb5 Context context, const krb5_trace_info *info, void *cb_data)

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_4**

krb5_ui_4

krb5_ui_4 is an unsigned 32-bit integer type.

**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

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

5.2.2 Internal

krb5_auth_context

Declaration

typedef struct _krb5_auth_context* krb5_auth_context

krb5_cksumtype

Declaration

typedef krb5_int32 krb5_cksumtype

krb5_context

Declaration

typedef struct _krb5_context* krb5_context

krb5_cc_cursor

Declaration

Cursor for sequential lookup.

5.2. krb5 types and structures
typedef krb5_pointer krb5_cc_cursor


krb5_ccache


declaration
typedef struct _krb5_ccache* krb5_ccache


krb5_cccol_cursor


declaration
typedef struct _krb5_cccol_cursor* krb5_cccol_cursor


krb5_init_creds_context


declaration
typedef struct _krb5_init_creds_context* krb5_init_creds_context


krb5_key


declaration
typedef struct krb5_key_st* krb5_key


krb5_keytab


declaration
typedef krb5_keytab
5.3 krb5 simple macros

5.3.1 Public

ADDRTYPE_ADDRPORT

ADDRTYPE_ADDRPORT

ADDRTYPE_CHAOS

ADDRTYPE_CHAOS
ADDRTYPE_DDP

ADDRTYPE_DDP

ADDRTYPE_INET

ADDRTYPE_INET

ADDRTYPE_INET6

ADDRTYPE_INET6

ADDRTYPE_IPPORT

ADDRTYPE_IPPORT

ADDRTYPE_ISO

ADDRTYPE_ISO

ADDRTYPE_IS_LOCAL

ADDRTYPE_IS_LOCAL

ADDRTYPE_NETBIOS

ADDRTYPE_NETBIOS

ADDRTYPE_XNS

ADDRTYPE_XNS
AD_TYPE_EXTERNAL

AD_TYPE_REGISTERED

AD_TYPE_RESERVED

AP_OPTS_ETYPE_NEGOTIATION

AP_OPTS_MUTUAL_REQUIRED

AP_OPTS_RESERVED

AP_OPTS_USE_SESSION_KEY

5.3. krb5 simple macros
**AP_OPTS_USE_SUBKEY**

Generate a subsession key from the current session key obtained from the credentials.

**AP_OPTS_WIRE_MASK**

**CKSUMTYPE_CMAC_CAMELLIA128**

**CKSUMTYPE_CMAC_CAMELLIA256**

**CKSUMTYPE_CRC32**

**CKSUMTYPE_DESCBC**

**CKSUMTYPE_HMAC_MD5_ARCFOUR**

**CKSUMTYPE_HMAC_SHA1_96_AES128**

RFC 3962.

Used with ENCTYPE_AES128_CTS_HMAC_SHA1_96
CKSUMTYPE_HMAC_SHA1_96_AES256

CKSUMTYPE_HMAC_SHA1_96_AES256
RFC 3962.
Used with ENCTYPE_AES256_CTS_HMAC_SHA1_96

CKSUMTYPE_HMAC_SHA1_DES3

CKSUMTYPE_HMAC_SHA1_DES3

CKSUMTYPE_MD5_HMAC_ARCFOUR

CKSUMTYPE_MD5_HMAC_ARCFOUR

CKSUMTYPE_NIST_SHA

CKSUMTYPE_NIST_SHA

CKSUMTYPE_RSA_MD4

CKSUMTYPE_RSA_MD4

CKSUMTYPE_RSA_MD4_DES

CKSUMTYPE_RSA_MD4_DES

CKSUMTYPE_RSA_MD5

CKSUMTYPE_RSA_MD5

CKSUMTYPE_RSA_MD5_DES

CKSUMTYPE_RSA_MD5_DES

5.3. krb5 simple macros
ENCTYPE_AES128_CTS_HMAC_SHA1_96
RFCS 3962.

ENCTYPE_AES256_CTS_HMAC_SHA1_96
RFCS 3962.

ENCTYPE_ARCFOUR_HMAC

ENCTYPE_ARCFOUR_HMAC_EXP

ENCTYPE_CAMELLIA128_CTS_CMAC

ENCTYPE_CAMELLIA256_CTS_CMAC

ENCTYPE_DES3_CBC_ENV

ENCTYPE_DES3_CBC_RAW
5.3. krb5 simple macros

ENCTYPE_DES3_CBC_SHA

ENCTYPE_DES3_CBC_SHA

ENCTYPE_DES3_CBC_SHA

ENCTYPE_DES3_CBC_SHA

ENCTYPE_DES_CBC_SHA

ENCTYPE_DES_CBC_CRC

ENCTYPE_DES_CBC_CRC

DES cbc mode with CRC-32.

ENCTYPE_DES_CBC_CRC

ENCTYPE_DES_CBC_CRC

ENCTYPE_DES_CBC_MD4

ENCTYPE_DES_CBC_MD4

DES cbc mode with RSA-MD4.

ENCTYPE_DES_CBC_MD4

ENCTYPE_DES_CBC_MD4

ENCTYPE_DES_CBC_MD5

ENCTYPE_DES_CBC_MD5

DES cbc mode with RSA-MD5.

ENCTYPE_DES_CBC_MD5

ENCTYPE_DES_CBC_MD5

ENCTYPE_DES_CBC_RAW

ENCTYPE_DES_CBC_RAW

ENCTYPE_DES_CBC_RAW

ENCTYPE_DES_HMAC_SHA1

ENCTYPE_DES_HMAC_SHA1

ENCTYPE_DES_HMAC_SHA1

ENCTYPE_DSA_SHA1_CMS

ENCTYPE_DSA_SHA1_CMS

DSA with SHA1, CMS signature.

ENCTYPE_DSA_SHA1_CMS

ENCTYPE_DSA_SHA1_CMS
**ENCTYPE_MD5_RSA_CMS**

MD5 with RSA, CMS signature.

```
ENCTYPE_MD5_RSA_CMS  0x000a
```

**ENCTYPE_NULL**

```
ENCTYPE_NULL  0x0000
```

**ENCTYPE_RC2_CBC_ENV**

RC2 cbc mode, CMS enveloped data.

```
ENCTYPE_RC2_CBC_ENV  0x000c
```

**ENCTYPE_RSA_ENV**

RSA encryption, CMS enveloped data.

```
ENCTYPE_RSA_ENV  0x000d
```

**ENCTYPE_RSA_ES_OAEP_ENV**

RSA w/OEAP encryption, CMS enveloped data.

```
ENCTYPE_RSA_ES_OAEP_ENV  0x000e
```

**ENCTYPE_SHA1_RSA_CMS**

SHA1 with RSA, CMS signature.

```
ENCTYPE_SHA1_RSA_CMS  0x000b
```

**ENCTYPE_UNKNOWN**

```
ENCTYPE_UNKNOWN  0x01ff
```
<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDC_OPT_ALLOW_POSTDATE</td>
<td></td>
<td>0x04000000</td>
</tr>
<tr>
<td>KDC_OPT_CANONICALIZE</td>
<td></td>
<td>0x00010000</td>
</tr>
<tr>
<td>KDC_OPT_CNAME_IN_ADDL_TKT</td>
<td></td>
<td>0x00020000</td>
</tr>
<tr>
<td>KDC_OPT_DISABLE_TRANSITED_CHECK</td>
<td></td>
<td>0x00000020</td>
</tr>
<tr>
<td>KDC_OPT_ENC_TKT_IN_SKEY</td>
<td></td>
<td>0x00000008</td>
</tr>
<tr>
<td>KDC_OPT_FORWARDABLE</td>
<td></td>
<td>0x40000000</td>
</tr>
<tr>
<td>KDC_OPT_FORWARDED</td>
<td></td>
<td>0x20000000</td>
</tr>
<tr>
<td>KDC_OPT_POSTDATED</td>
<td></td>
<td>0x02000000</td>
</tr>
</tbody>
</table>
KDC_OPT_PROXIAIBLE

KDC_OPT_PROXIAIBLE
KDC_OPT_PROXIAIBLE | 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 | 0x05480000
KRB5_ALTAUTH_ATT_CHALLENGE_RESPONSE

alternate authentication types

KRB5_ANONYMOUS_PRINCSTR

Anonymous principal name.

KRB5_ANONYMOUS_REALMSTR

Anonymous realm.

KRB5_AP_REP

Response to mutual AP request.

KRB5_AP_REQ

Auth req to application server.

KRB5_AS_REP

Response to AS request.

KRB5_AS_REQ

Initial authentication request.
<table>
<thead>
<tr>
<th>KR5_AUTHDATA_AND_OR</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>KR5_AUTHDATA_ETYPE_NEGOTIATION</td>
<td>129</td>
</tr>
<tr>
<td>KR5_AUTHDATA_FX_ARMOR</td>
<td>71</td>
</tr>
<tr>
<td>KR5_AUTHDATA_IF_RELEVANT</td>
<td>1</td>
</tr>
<tr>
<td>KR5_AUTHDATA_INITIAL_VERIFIED_CAS</td>
<td>9</td>
</tr>
<tr>
<td>KR5_AUTHDATA_KDC_ISSUED</td>
<td>4</td>
</tr>
<tr>
<td>KR5_AUTHDATA_MANDATORY_FOR_KDC</td>
<td>8</td>
</tr>
<tr>
<td>KR5_AUTHDATA_OSF_DCE</td>
<td>64</td>
</tr>
</tbody>
</table>
**KRB5_AUTHDATA_SESAME**

**KRB5_AUTHDATA_SESAME**

formerly 142 in krb5 1.8

**KRB5_AUTHDATA_SIGNTICKET**

**KRB5_AUTHDATA_SIGNTICKET**

Prevent replays with sequence numbers.

**KRB5_AUTHDATA_WIN2K_PAC**

**KRB5_AUTHDATA_WIN2K_PAC**

Generate the local network address.

**KRB5_AUTH_CONTEXT_DO_SEQUENCE**

**KRB5_AUTH_CONTEXT_DO_SEQUENCE**

Generate the local network address and the local port.

**KRB5_AUTH_CONTEXT_DO_TIME**

**KRB5_AUTH_CONTEXT_DO_TIME**

Generate the local network address.

**KRB5_AUTH_CONTEXT_GENERATE_LOCAL_ADDR**

**KRB5_AUTHCONTEXT_GENERATE_LOCAL_ADDR**

Generate the local network address and the local port.

**KRB5_AUTH_CONTEXT_GENERATE_LOCAL_FULL_ADDR**

5.3. krb5 simple macros
**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**

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 0x00000010
```

**KRB5_AUTH_CONTEXT_RET_SEQUENCE**

Save sequence numbers for application.

```
KRB5_AUTH_CONTEXT_RET_SEQUENCE 0x00000008
```

**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 0x00000020
```

**KRB5_CRED**

Cred forwarding message.

```
KRB5_CRED ((krb5_msgtype)22)
```
KERBS_CRYPTO_TYPE_CHECKSUM

KERBS_CRYPTO_TYPE_CHECKSUM
[out] checksum for MIC

KERBS_CRYPTO_TYPE_DATA

KERBS_CRYPTO_TYPE_DATA
[in, out] plaintext

KERBS_CRYPTO_TYPE_EMPTY

KERBS_CRYPTO_TYPE_EMPTY
[in] ignored

KERBS_CRYPTO_TYPE_HEADER

KERBS_CRYPTO_TYPE_HEADER
[out] header

KERBS_CRYPTO_TYPE_PADDING

KERBS_CRYPTO_TYPE_PADDING
[out] padding

KERBS_CRYPTO_TYPE_SIGNONLY

KERBS_CRYPTO_TYPE_SIGNONLY
[in] associated data

KERBS_CRYPTO_TYPE_STREAM

KERBS_CRYPTO_TYPE_STREAM
[in] entire message without decomposing the structure into header, data and trailer buffers

5.3. krb5 simple macros
<table>
<thead>
<tr>
<th>KRB5_CRYPTO_TYPE_TRAILER</th>
<th>[out] checksum for encrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KRB5_CYBERSAFE_SECUREID</th>
<th>Cybersafe.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KRB5_DOMAIN_X500_COMPRESS</th>
<th>Transited encoding types.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KRB5_ENCPADATA_REQ_ENC_PA_REP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>149</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KRB5_ERROR</th>
<th>Error response.</th>
</tr>
</thead>
<tbody>
<tr>
<td>((krb5_msgtype)30)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KRB5_FAST_REQUIRED</th>
<th>Require KDC to support FAST.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KRB5_GC_CACHED</th>
<th>Want cached ticket only.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
KRB5_GC_CANONICALIZE

Set canonicalize KDC option.

KRB5_GCCONSTRAINED_DELEGATION

Constrained delegation.

KRB5_GC_FORWARDABLE

Acquire forwardable tickets.

KRB5_GC_NO_STORE

Do not store in credential cache.

KRB5_GC_NO_TRANSIT_CHECK

Disable transited check.

KRB5_GC_USER_USER

Want user-user ticket.

KRB5_GET_INIT_CREDS_OPT_ADDRESS_LIST

5.3. krb5 simple macros
KRB5_GET_INIT_CREDS_OPT_ANONYMOUS

KRB5_GET_INIT_CREDS_OPT_CANONICALIZE

KRB5_GET_INIT_CREDS_OPT_CHG_PWD_PRMPT

KRB5_GET_INIT_CREDS_OPT_ETYPE_LIST

KRB5_GET_INIT_CREDS_OPT_FORWARDABLE

KRB5_GET_INIT_CREDS_OPT_PREAUTH_LIST

KRB5_GET_INIT_CREDS_OPT_PROXIABLE

KRB5_GET_INIT_CREDS_OPT_RENEW_LIFE
5.3. krb5 simple macros

**KRB5_GET_INIT_CRED_OPT_SALT**

KRB5_GET_INIT_CRED_OPT_SALT

**KRB5_GET_INIT_CRED_OPT_TKT_LIFE**

KRB5_GET_INIT_CRED_OPT_TKT_LIFE

**KRB5_INIT_CONTEXT_SECURE**

Use secure context configuration.

**KRB5_INIT_CONTEXT_KDC**

Use KDC configuration if available.

**KRB5_INIT_CREDS_STEP_FLAG_CONTINUE**

More responses needed.

**KRB5_INT16_MAX**

KRB5_INT16_MAX

**KRB5_INT16_MIN**

KRB5_INT16_MIN

**KRB5_INT32_MAX**

KRB5_INT32_MAX
KRB5_INT32_MIN

KRB5_KEYUSAGE_AD_ITE

KRB5_KEYUSAGE_AD_MTE

KRB5_KEYUSAGE_AD_SIGNEDPATH

KRB5_KEYUSAGE_APP_DATA_CKSUM

KRB5_KEYUSAGE_APP_DATA_ENCRYPT

KRB5_KEYUSAGE_AP_REP_ENCPART
KRB5_KEYUSAGE_AP_REQ_AUTH

KRB5_KEYUSAGE_AP_REQ_AUTH

KRB5_KEYUSAGE_AP_REQ_AUTH_CKSUM

KRB5_KEYUSAGE_AP_REQ_AUTH_CKSUM

KRB5_KEYUSAGE_AS_REP_ENCPART

KRB5_KEYUSAGE_AS_REP_ENCPART

KRB5_KEYUSAGE_AS_REQ

KRB5_KEYUSAGE_AS_REQ

KRB5_KEYUSAGE_AS_REQ_PA_ENC_TS

KRB5_KEYUSAGE_AS_REQ_PA_ENC_TS

KRB5_KEYUSAGE_ENC_CHALLENGE_CLIENT

KRB5_KEYUSAGE_ENC_CHALLENGE_CLIENT

KRB5_KEYUSAGE_ENC_CHALLENGE_KDC

KRB5_KEYUSAGE_ENC_CHALLENGE_KDC

KRB5_KEYUSAGE_FAST_ENC

KRB5_KEYUSAGE_FAST_ENC

5.3. krb5 simple macros
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
5.3. krb5 simple macros

KRB5_KEYUSAGE_KRB_CRED_ENCPART

KRB5_KEYUSAGE_KRB_CRED_ENCPART

KRB5_KEYUSAGE_KRB_CRED_ENCPART

KRB5_KEYUSAGE_KRB_ERROR_CKSUM

KRB5_KEYUSAGE_KRB_ERROR_CKSUM

KRB5_KEYUSAGE_KRB_ERROR_CKSUM

KRB5_KEYUSAGE_KRB_PRIV_ENCPART

KRB5_KEYUSAGE_KRB_PRIV_ENCPART

KRB5_KEYUSAGE_KRB_PRIV_ENCPART

KRB5_KEYUSAGE_KRB_SAFE_CKSUM

KRB5_KEYUSAGE_KRB_SAFE_CKSUM

KRB5_KEYUSAGE_KRB_SAFE_CKSUM

KRB5_KEYUSAGE_PA_OTP_REQUEST

KRB5_KEYUSAGE_PA_OTP_REQUEST

See RFC 6560 section 4.2.

KRB5_KEYUSAGE_PA_OTP_REQUEST

KRB5_KEYUSAGE_PA_PKINIT_KX

KRB5_KEYUSAGE_PA_PKINIT_KX

KRB5_KEYUSAGE_PA_PKINIT_KX

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

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
KRB5_KEYUSAGE_PA_SAM_CHALLENGE_CKSUM

KRB5_KEYUSAGE_PA_SAM_CHALLENGE_CKSUM

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_RESPONSE

KRB5_KEYUSAGE_PA_SAM_RESPONSE
Note conflict with KRB5_KEYUSAGE_PA_S4U_X509_USER_REPLY.

KRB5_KEYUSAGE_TGS_REP_ENCPART_SESSKEY

KRB5_KEYUSAGE_TGS_REP_ENCPART_SESSKEY

KRB5_KEYUSAGE_TGS_REP_ENCPART_SUBKEY

KRB5_KEYUSAGE_TGS_REP_ENCPART_SUBKEY

KRB5_KEYUSAGE_TGS_REQ_AD_SESSKEY

KRB5_KEYUSAGE_TGS_REQ_AD_SESSKEY

KRB5_KEYUSAGE_TGS_REQ_AD_SUBKEY

KRB5_KEYUSAGE_TGS_REQ_AD_SUBKEY

KRB5_KEYUSAGE_TGS_REQ_AUTH

KRB5_KEYUSAGE_TGS_REQ_AUTH
KRB5_KEYUSAGE_TGS_REQ_AUTH_CKSUM

KRB5_KEYUSAGE_TGS_REQ_AUTH_CKSUM

KRB5_KPASSWD_ACCESSDENIED

KRB5_KPASSWD_ACCESSDENIED
Not authorized.

KRB5_KPASSWD_AUTHERROR

KRB5_KPASSWD_AUTHERROR
Authentication error.

KRB5_KPASSWD_BAD_VERSION

KRB5_KPASSWD_BAD_VERSION
Unknown RPC version.

KRB5_KPASSWD_HARDERROR

KRB5_KPASSWD_HARDERROR
Server error.

KRB5_KPASSWD_INITIAL_FLAG_NEEDED

KRB5_KPASSWD_INITIAL_FLAG_NEEDED
The presented credentials were not obtained using a password directly.

KRB5_KPASSWD_MALFORMED

KRB5_KPASSWD_MALFORMED
Malformed request.
KRB5_KPASSWD_SOFTERROR
Password change rejected.

KRB5_KPASSWD_SUCCESS
Success.

KRB5_LRQ_ALL_ACCT_EXPTIME

KRB5_LRQ_ALL_LAST_INITIAL

KRB5_LRQ_ALL_LAST_RENEWAL

KRB5_LRQ_ALL_LAST_REQ

KRB5_LRQ_ALL_LAST_TGT

KRB5_LRQ_ALL_LAST_TGT_ISSUED
### Kerberos Application Developer Guide, Release 1.11.1

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>KRB5_LRQ_ALL_PW_EXPIRE</code></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td><code>KRB5_LRQ_NONE</code></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td><code>KRB5_LRQ_ONE_ACCT_EXPIRE</code></td>
<td></td>
<td>-7</td>
</tr>
<tr>
<td><code>KRB5_LRQ_ONE_LAST_INITIAL</code></td>
<td></td>
<td>-2</td>
</tr>
<tr>
<td><code>KRB5_LRQ_ONE_LAST_RENEWAL</code></td>
<td></td>
<td>-4</td>
</tr>
<tr>
<td><code>KRB5_LRQ_ONE_LAST_REQ</code></td>
<td></td>
<td>-5</td>
</tr>
<tr>
<td><code>KRB5_LRQ_ONE_LAST_TGT</code></td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td><code>KRB5_LRQ_ONE_LAST_TGT_ISSUED</code></td>
<td></td>
<td>-3</td>
</tr>
</tbody>
</table>

5.3. krb5 simple macros

185
**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

Service with host name as instance (telnet, rcommands)

KRB5_NT_SRV_INST

Service and other unique instance (krbtgt)

KRB5_NT_SRV_XHST

Service with host as remaining components.

KRB5_NT_UID

Unique ID.

KRB5_NT_UNKNOWN

Name type not known.

KRB5_NT_WELLKNOWN

Well-known (special) principal.

KRB5_NT_X500_PRINCIPAL

PKINIT.

5.3. krb5 simple macros
KRB5_PAC_CLIENT_INFO

Client name and ticket info.

KRB5_PAC_CREDENTIALS_INFO

Credentials information.

KRB5_PAC_DELEGATION_INFO

Constrained delegation info.

KRB5_PAC_LOGON_INFO

Logon information.

KRB5_PAC_PRIVSVR_CHECKSUM

KDC checksum.

KRB5_PAC_SERVER_CHECKSUM

Server checksum.

KRB5_PAC_UPN_DNS_INFO

User principal name and DNS info.
KRB5_PADATA_AFS3_SALT

Cygnus.

KRB5_PADATA_AFS3_SALT

KRB5_PADATA_AP_REQ

KRB5_PADATA_AP_REQ

KRB5_PADATA_ENCRYPTED_CHALLENGE

KRB5_PADATA_ENCRYPTED_CHALLENGE

KRB5_PADATA_ENC_SANDIA_SECURID

SecurId passcode.

KRB5_PADATA_ENC_SANDIA_SECURID

KRB5_PADATA_ENC_TIMESTAMP

timestamp encrypted in key

KRB5_PADATA_ENC_TIMESTAMP

KRB5_PADATA_ENC_UNIX_TIME

timestamp encrypted in key

KRB5_PADATA_ENC_UNIX_TIME

KRB5_PADATA_ETYPE_INFO

Etype info for preauth.

KRB5_PADATA_ETYPE_INFO

KRB5_PADATA_ETYPE_INFO2

KRB5_PADATA_ETYPE_INFO2

5.3. krb5 simple macros

189
**KR5_PADATA_FOR_USER**

KR5_PADATA_FOR_USER

username protocol transition request

**KR5_PADATA_FX_COOKIE**

KR5_PADATA_FX_COOKIE

**KR5_PADATA_FX_ERROR**

KR5_PADATA_FX_ERROR

**KR5_PADATA_FX_FAST**

KR5_PADATA_FX_FAST

**KR5_PADATA_GET_FROM_TYPED_DATA**

KR5_PADATA_GET_FROM_TYPED_DATA

**KR5_PADATA_NONE**

KR5_PADATA_NONE

**KR5_PADATA_OSF_DCE**

KR5_PADATA_OSF_DCE

OSF DCE.

**KR5_PADATA_OTP_CHALLENGE**

KR5_PADATA_OTP_CHALLENGE

See RFC 6560 section 4.1.
5.3. krb5 simple macros

**KRB5_PADATA_OTP_PIN_CHANGE**

See RFC 6560 section 4.3.

**KRB5_PADATA_OTP_REQUEST**

See RFC 6560 section 4.2.

**KRB5_PADATA_PAC_REQUEST**

include Windows PAC

**KRB5_PADATA_PKINIT_KX**

**KRB5_PADATA_PK_AS_REP**

PKINIT.

**KRB5_PADATA_PK_AS_REP_OLD**

PKINIT.

**KRB5_PADATA_PK_AS_REQ**

PKINIT.
KRB5_PADATA_PK_AS_REQ_OLD
PKINIT.

KRB5_PADATA_PW_SALT
certificate protocol transition request

draft challenge system

draft referral system

draft challenge system, updated

KRB5_PADATA_SAM_REDIRECT

KRB5_PADATA_SAM_CHALLENGE_2
draft challenge system, updated

KRB5_PADATA_SAM_REDIRECT

KRB5_PADATA_S4U_X509_USER
certificate protocol transition request

KRB5_PADATA_SAM_CHALLENGE
draft challenge system

KRB5_PADATA_SAM_REDIRECT

KRB5_PADATA_SAM_CHALLENGE_2
draft challenge system, updated

KRB5_PADATA_SAM_REDIRECT
5.3. krb5 simple macros
**KRB5_PRINCIPAL_COMPARE_ENTERPRISE**

UPNs as real principals.

```
KRB5_PRINCIPAL_COMPARE_ENTERPRISE 2
```

**KRB5_PRINCIPAL_COMPARE_IGNORE_REALM**

ignore realm component

```
KRB5_PRINCIPAL_COMPARE_IGNORE_REALM 1
```

**KRB5_PRINCIPAL_COMPARE_UTF8**

treat principals as UTF-8

```
KRB5_PRINCIPAL_COMPARE_UTF8 8
```

**KRB5_PRINCIPAL_PARSE_ENTERPRISE**

Create single-component enterprise principle.

```
KRB5_PRINCIPAL_PARSE_ENTERPRISE 0x4
```

**KRB5_PRINCIPAL_PARSE_IGNORE_REALM**

Ignore realm if present.

```
KRB5_PRINCIPAL_PARSE_IGNORE_REALM 0x8
```

**KRB5_PRINCIPAL_PARSE_NO_REALM**

Error if realm is present.

```
KRB5_PRINCIPAL_PARSE_NO_REALM 0x1
```

**KRB5_PRINCIPAL_PARSE_REQUIRE_REALM**

Error if realm is not present.

```
KRB5_PRINCIPAL_PARSE_REQUIRE_REALM 0x2
```
**KRB5_PRINCIPAL_UNPARSE_DISPLAY**

Don’t escape special characters.

```
KRB5_PRINCIPAL_UNPARSE_DISPLAY 0x4
```

**KRB5_PRINCIPAL_UNPARSE_NO_REALM**

Omit realm always.

```
KRB5_PRINCIPAL_UNPARSE_NO_REALM 0x2
```

**KRB5_PRINCIPAL_UNPARSE_SHORT**

Omit realm if it is the local realm.

```
KRB5_PRINCIPAL_UNPARSE_SHORT 0x1
```

**KRB5_PRIV**

Private application message.

```
KRB5_PRIV ((krb5_msgtype)21)
```

**KRB5_PROMPT_TYPE_NEW_PASSWORD**

Prompt for new password (during password change)

```
KRB5_PROMPT_TYPE_NEW_PASSWORD 0x2
```

**KRB5_PROMPT_TYPE_NEW_PASSWORD_AGAIN**

Prompt for new password again.

```
KRB5_PROMPT_TYPE_NEW_PASSWORD_AGAIN 0x3
```

**KRB5_PROMPT_TYPE_PASSWORD**

Prompt for password.

```
KRB5_PROMPT_TYPE_PASSWORD 0x1
```

5.3. krb5 simple macros
KRB5_PROMPT_TYPE_PREAUTH

Prompt for preauthentication data (such as an OTP value)

KRB5_PVNO

Protocol version number.

KRB5_REALM_BRANCH_CHAR

Constant for realm referrals.

KRB5_RECVAUTH_BADAUTHVERS

This flag indicates that the PIN value MUST be collected.

KRB5_REFERRAL_REALM

Chapter 5. Complete reference - API and datatypes
**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_NEXTTOTP**

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_NEXTTOTP 0x0004
```

**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 2
```

**KRB5_RESPONDER_OTP_FORMAT_DECIMAL**

```
KRB5_RESPONDER_OTP_FORMAT_DECIMAL 0
```

**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:

```json
{ "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:

```json
{ "tokeninfo": <number>, "value": <string (optional)>, "pin": <string (optional)> }
```

For more detail, please see RFC 6560.

**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_SAFE**

**KRB5_SAFE**

Safe application message.

**KRB5_SAM_MUST_PK_ENCRYPT_SAD**

**KRB5_SAM_MUST_PK_ENCRYPT_SAD**

Currently must be zero

**KRB5_SAM_SEND_ENCRYPTED_SAD**

**KRB5_SAM_SEND_ENCRYPTED_SAD**

Currently must be zero

**KRB5_SAM_USE_SAD_AS_KEY**

**KRB5_SAM_USE_SAD_AS_KEY**

Currently must be zero
The second ticket must match.

The authorization data must match.

All the flags set in the match credentials must be set.

All the flags must match exactly.

The is_skey field must match exactly.

The encryption key type must match.

Only the name portion of the principal name must match.
**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**

All the time fields must match exactly.

| KRB5_TC_MATCH_TIMES_EXACT | 0x00000008 |

**KRB5_TC_NOTICKET**

| KRB5_TC_NOTICKET | 0x00000002 |

**KRB5_TC_OPENCLOSE**

Open and close the file for each cache operation.

| KRB5_TC_OPENCLOSE | 0x00000001 |

**KRB5_TC_SUPPORTED_KTYPES**

The supported key types must match.

| KRB5_TC_SUPPORTED_KTYPES | 0x00000200 |

**KRB5_TGS_NAME**

| KRB5_TGS_NAME | "krbtgt" |

**KRB5_TGS_NAME_SIZE**

| KRB5_TGS_NAME_SIZE | 6 |
**KRB5_TGS_REP**

Response to TGS request.

```
KRB5_TGS_REP ((krb5_msgtype)13)
```

**KRB5_TGS_REQ**

Ticket granting server request.

```
KRB5_TGS_REQ ((krb5_msgtype)12)
```

**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 0x0001
```

**KRB5_WELLKNOWN_NAMESTR**

First component of NT_WELLKNOWN principals.

```
KRB5_WELLKNOWN_NAMESTR "WELLKNOWN"
```

**LR_TYPE_INTERPRETATION_MASK**

```
LR_TYPE_INTERPRETATION_MASK 0x7fff
```

**LR_TYPE_THIS_SERVER_ONLY**

```
LR_TYPE_THIS_SERVER_ONLY 0x8000
```

---

**5.3. krb5 simple macros**
MAX_KEYTAB_NAME_LEN

Long enough for MAXPATHLEN + some extra.

MSEC_DIRBIT

MSEC_DIRBIT

MSEC_VAL_MASK

MSEC_VAL_MASK

SALT_TYPE_AFS_LENGTH

SALT_TYPE_AFS_LENGTH

SALT_TYPE_NO_LENGTH

SALT_TYPE_NO_LENGTH

THREEPARAMOPEN

THREEPARAMOPEN

TKT_FLG_ANONYMOUS

TKT_FLG_ANONYMOUS

TKT_FLG_ENC_PA_REP

TKT_FLG_ENC_PA_REP
TKT_FLG_FORWARDABLE

TKT_FLG_FORWARDABLE

TKT_FLG_FORWARDED

TKT_FLG_FORWARDED

TKT_FLG_HW_AUTH

TKT_FLG_HW_AUTH

TKT_FLG_INITIAL

TKT_FLG_INITIAL

TKT_FLG_INVALID

TKT_FLG_INVALID

TKT_FLG_MAY_POSTDATE

TKT_FLG_MAY_POSTDATE

TKT_FLG_OK_AS_DELEGATE

TKT_FLG_OK_AS_DELEGATE

TKT_FLG_POSTDATED

TKT_FLG_POSTDATED
TKT_FLG_PRE_AUTH

0x00200000

TKT_FLG_PROXIABLE

0x10000000

TKT_FLG_PROXY

0x08000000

TKT_FLG_RENEWABLE

0x00800000

TKT_FLG_TRANSIT_POLICY_CHECKED

0x00080000

VALID_INT_BITS

0x00000000

VALID_UINT_BITS

INT_MAX

UINT_MAX

const

0x00000000


5.3. krb5 simple macros

**krb5_princ_component**

```c
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**

```c
krb5_princ_name

krb5_princ_name

krb5_princ_name (context, princ)  
(princ)->data
```

**krb5_princ_realm**

```c
krb5_princ_realm

krb5_princ_realm

krb5_princ_realm (context, princ)  
(& (princ)->realm)
```

**krb5_princ_set_realm**

```c
krb5_princ_set_realm

krb5_princ_set_realm

krb5_princ_set_realm (context, princ, value)  
((princ)->realm = *(value))
```

**krb5_princ_set_realm_data**

```c
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**

```c
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**

```c
krb5_princ_size

krb5_princ_size

krb5_princ_size (context, princ)  
(princ)->length
```

**krb5_princ_type**

```c
krb5_princ_type

krb5_princ_type

krb5_princ_type (context, princ)  
(princ)->type
```
**krb5_roundup**

\[
\text{krb5_roundup} \ (x, y) = \left(\frac{(x + (y - 1))}{y}\right) \times y
\]

**krb5_x**

\[
\text{krb5_x} \ (\text{ptr, args}) = \left(\text{ptr}\right) ? \left(\left(\text{ptr}\right) \text{ args}\right) : (\text{abort()}, 1)
\]

**krb5 xc**

\[
\text{krb5 xc} \ (\text{ptr, args}) = \left(\text{ptr}\right) ? \left(\left(\text{ptr}\right) \text{ args}\right) : (\text{abort()}, (\text{char}*) 0)
\]

### 5.3.2 Deprecated macros

**krb524_convert_creds_kdc**

\[
\text{krb524_convert_creds_kdc}
\]

**krb524_init_ets**

\[
\text{krb524_init_ets} \ (x) = 0
\]
INDEX

A
AD_TYPE_EXTERNAL (built-in variable), 161
AD_TYPE_FIELD_TYPE_MASK (built-in variable), 161
AD_TYPE_REGISTERED (built-in variable), 161
AD_TYPE_RESERVED (built-in variable), 161
ADRTYPE_ADDRPORT (built-in variable), 159
ADRTYPE_CHAOS (built-in variable), 159
ADRTYPE_DDP (built-in variable), 160
ADRTYPE_INET (built-in variable), 160
ADRTYPE_INET6 (built-in variable), 160
ADRTYPE_INET_PORT (built-in variable), 160
ADRTYPE_ISO (built-in variable), 160
ADRTYPE_NETBIOS (built-in variable), 160
ADRTYPE_XNS (built-in variable), 160
AP_OPTS_ETYPE_NEGOTIATION (built-in variable), 161
AP_OPTS_MUTUAL_REQUIRED (built-in variable), 161
AP_OPTS_RESERVED (built-in variable), 161
AP_OPTS_USE_SESSION_KEY (built-in variable), 161
AP_OPTS_USE_SUBKEY (built-in variable), 162
AP_OPTS_WIRE_MASK (built-in variable), 162

C
CKSUMTYPE_CMAC_CAMELLIA128 (built-in variable), 162
CKSUMTYPE_CMAC_CAMELLIA256 (built-in variable), 162
CKSUMTYPE_CRC32 (built-in variable), 162
CKSUMTYPE_DES_CBC (built-in variable), 162
CKSUMTYPE_HMAC_MD5_ARCFOUR (built-in variable), 162
CKSUMTYPE_HMAC_SHA1_96_AES128 (built-in variable), 162
CKSUMTYPE_HMAC_SHA1_96_AES256 (built-in variable), 163
CKSUMTYPE_HMAC_SHA1_96_AES256 (built-in variable), 163
CKSUMTYPE_RSA_MD5 (built-in variable), 163
CKSUMTYPE_RSA_MD5 (built-in variable), 163
CKSUMTYPE_RSA_MD5 (built-in variable), 163

E
ENCTYPE_AES128_CTS_HMAC_SHA1_96 (built-in variable), 164
ENCTYPE_AES256_CTS_HMAC_SHA1_96 (built-in variable), 164
ENCTYPE_ARCFOUR_HMAC (built-in variable), 164
ENCTYPE_ARCFOUR_HMAC_EXP (built-in variable), 164
ENCTYPE_CAMELLIA128_CTS_CMAC (built-in variable), 164
ENCTYPE_CAMELLIA128_CTS_CMAC (built-in variable), 164
ENCTYPE_DES_CBC_ENV (built-in variable), 164
ENCTYPE_DES_CBC_RAW (built-in variable), 164
ENCTYPE_DES_CBC_SHA (built-in variable), 165
ENCTYPE_DES_CBC_SHA1 (built-in variable), 165
ENCTYPE_DES_CBC_CRC (built-in variable), 165
ENCTYPE_DES_CBC_MD4 (built-in variable), 165
ENCTYPE_DES_CBC_MD5 (built-in variable), 165
ENCTYPE_DES_CBC_RAW (built-in variable), 165
ENCTYPE_DES_CBC_SHA1 (built-in variable), 165
ENCTYPE_DSA_SHA1 (built-in variable), 164
ENCTYPE_NULL (built-in variable), 166
ENCTYPE_RC2_CBC_ENV (built-in variable), 166
ENCTYPE_RSA_ENV (built-in variable), 166
ENCTYPE_RSA_ES_OAEP_ENV (built-in variable), 166
ENCTYPE_SHA1_RSA_CMS (built-in variable), 166
ENCTYPE_UNKNOWN (built-in variable), 166

K
KDC_OPT_ALLOW_POSTDATE (built-in variable), 167
KDC_OPT_CANONICALIZE (built-in variable), 167
KDC_OPT_CNAME_IN_ADDL_TKT (built-in variable), 167
KDC_OPT_DISABLE_TRANSITED_CHECK (built-in variable), 167
KDC_OPT_ENC_TKT_IN_SKEY (built-in variable), 167
KDC_OPT_FORWARDABLE (built-in variable), 167
KDC_OPT_FORWARDED (built-in variable), 167
KDC_OPT_POSTDATED (built-in variable), 167
KDC_OPT_PROXIABLE (built-in variable), 167
KDC_OPT_PROXY (built-in variable), 168
KDC_OPT_RENEW (built-in variable), 168
KDC_OPT_RENEWABLE (built-in variable), 168
KDC_OPT_RENEWABLE_OK (built-in variable), 168
KDC_OPT_REQUEST_ANONYMOUS (built-in variable), 168
KDC_OPT_VALIDATE (built-in variable), 168
KDC_TKT_COMMON_MASK (built-in variable), 168
krb524_convert_creds_kdc (built-in variable), 206
krb524_init_ets (built-in variable), 206
krb5_425_conv_principal (C function), 46
krb5_524_conv_principal (C function), 46
krb5_524_convert_creds (C function), 124
krb5_address (C type), 131
krb5_address.addrtype (C member), 131
krb5_address.contents (C member), 131
krb5_address.length (C member), 131
krb5_address.magic (C member), 131
krb5_address_compare (C function), 47
krb5_address_order (C function), 47
krb5_address_search (C function), 47
krb5_addrtype (C type), 131
krb5_allow_weak_crypto (C function), 48
KRB5_ALTAUTH_ATT_CHALLENGE_RESPONSE (built-in variable), 169
krb5_aname_to_localname (C function), 48
KRB5_ANONYMOUS_PRINCSTR (built-in variable), 48
KRB5_ANONYMOUS_REALMSTR (built-in variable), 49
KRB5_ANONYMOUS_REALM (C function), 49
KRB5_AUTH_CONTEXT_DO_SEQUENCE (built-in variable), 171
KRB5_AUTH_CONTEXT_DO_TIME (built-in variable), 171
KRB5_AUTHCONTEXT_GENERATE_LOCAL_ADDR (built-in variable), 171
KRB5_AUTHCONTEXT_GENERATE_LOCAL_FULL_ADDR (built-in variable), 171
KRB5_AUTHCONTEXT_GENERATE_REMOTE_ADDR (built-in variable), 172
KRB5_AUTHCONTEXT_GENERATE_REMOTE_FULL_ADDR (built-in variable), 172
KRB5_AUTH_CONTEXT_PERMIT_ALL (built-in variable), 172
KRB5_AUTH_CONTEXT_DO_SEQUENCE (built-in variable), 171
KRB5_AUTH_CONTEXT_DO_TIME (built-in variable), 171
KRB5_AUTHCONTEXT_GENERATE_LOCAL_ADDR (built-in variable), 171
KRB5_AUTHCONTEXT_GENERATE_LOCAL_FULL_ADDR (built-in variable), 171
KRB5_AUTHCONTEXT_GENERATE_REMOTE_ADDR (built-in variable), 172
KRB5_AUTHCONTEXT_GENERATE_REMOTE_FULL_ADDR (built-in variable), 172
KRB5_AUTH_CONTEXT_PERMIT_ALL (built-in variable), 172
krb5_ap_req (C type), 132
krb5_ap_req.request (C function), 132
krb5_ap_req.authenticator (C member), 132
krb5_ap_req.magic (C member), 132
krb5_ap_req.ticket (C member), 132
krb5_appdefault_boolean (C function), 49
krb5_appdefault_string (C function), 49
KRB5 AS REP (built-in variable), 169
KRB5 AS REQ (built-in variable), 169
krb5_auth_con_free (C function), 50
krb5_auth_con_get_checksum_func (C function), 50
krb5_auth_con_getaddr (C function), 51
krb5_auth_con_getauthenticator (C function), 51
krb5_auth_con_getflags (C function), 51
krb5_auth_con_getkey (C function), 52
krb5_auth_con_getkey_k (C function), 52
krb5_auth_con_getlocalsequence_number (C function), 52
krb5_auth_con_getlocalsubkey (C function), 124
krb5_auth_con_getrcache (C function), 52
krb5_auth_con_getrecvsubkey (C function), 52
krb5_auth_con_getrecvsubkey_k (C function), 52
krb5_auth_con_getremotesubkey (C function), 52
krb5_auth_con_getremote_subkey_k (C function), 52
krb5_auth_con_getinit (C function), 54
krb5_auth_con_initvector (C function), 124
krb5_auth_con_set_checksum_func (C function), 55
krb5_auth_con_set_flags (C function), 55
krb5_auth_con_setrecvsubkey (C function), 55
krb5_auth_con_setrecvsubkey_k (C function), 55
krb5_auth_con_setrecv_subkey_k (C function), 55
krb5_auth_con_setremotesequence_number (C function), 53
krb5_auth_con_getremoteseqnumber (C function), 124
krb5_auth_con_getremotesubkey (C function), 124
krb5_auth_con_getset_flags (C function), 56
krb5_auth_con_setflags (C function), 56
krb5_auth_con_setrcache (C function), 56
krb5_auth_con_setrecv_subkey (C function), 57
krb5_auth_con_getrecv_subkey (C function), 57
krb5_auth_con_setrecv_subkey_k (C function), 57
krb5_auth_con_setflags (C function), 57
krb5_auth_con_setrecv_subkey_k (C function), 57
krb5_auth_con_setrecv_subkey_k (C function), 57
krb5_auth_con_setrecvuserkey (C function), 58
krb5_auth_context (C type), 157
KRB5_AUTH_CONTEXT_DO_SEQUENCE (built-in variable), 171
KRB5_AUTH_CONTEXT_DO_TIME (built-in variable), 171
KRB5_AUTHCONTEXT_GENERATE_LOCAL_ADDR (built-in variable), 171
KRB5_AUTHCONTEXT_GENERATE_LOCAL_FULL_ADDR (built-in variable), 171
KRB5_AUTHCONTEXT_GENERATE_REMOTE_ADDR (built-in variable), 172
KRB5_AUTHCONTEXT_GENERATE_REMOTE_FULL_ADDR (built-in variable), 172
KRB5_AUTH_CONTEXT_PERMIT_ALL (built-in variable), 172
Kerberos Application Developer Guide, Release 1.11.1

krb5_cc_set_flags (C function), 64
krb5_cc_start_seq_get (C function), 64
krb5_cc_store_cred (C function), 64
krb5_cc_support_switch (C function), 65
krb5_cc_switch (C function), 65
krb5_cc_unlock (C function), 65
krb5_ccache (C type), 158
krb5_cccol_cursor (C type), 158
krb5_cccol_cursor_free (C function), 65
krb5_ccol_next (C function), 66
krb5_ccol_have_content (C function), 66
krb5_ccol_last_change_time (C function), 67
krb5_ccol_lock (C function), 67
krb5_change_password (C function), 19
krb5_cksumtype_to_string (C function), 113
krb5_clear_error_message (C function), 67
krb5_const (built-in variable), 204
krb5_const_pointer (C type), 135
krb5 const_principal (C type), 135, 150
krb5 const principal.data (C member), 135, 150
krb5 const principal.length (C member), 135, 150
krb5 const principal.magic (C member), 135, 150
krb5 const principal.realm (C member), 135, 150
krb5 const principal.type (C member), 135, 150
krb5 context (C type), 157
krb5_copy_addresses (C function), 68
krb5_copy_authdata (C function), 68
krb5_copy_authenticator (C function), 69
krb5_copy_checksum (C function), 69
krb5_copy_context (C function), 69
krb5_copy_creds (C function), 69
krb5_copy_data (C function), 70
krb5_copy_error_message (C function), 70
krb5_copy_keyblock (C function), 70
krb5_copy_keyblock_contents (C function), 70
krb5_copy_principal (C function), 71
krb5_copy_ticket (C function), 71
KRB5_CRED (built-in variable), 172
krb5 cred (C type), 135
krb5 cred.enc_part(C member), 135
krb5 cred.enc_part2 (C member), 135
krb5 cred.magic (C member), 135
krb5 cred.tickets (C member), 135
KRB5 CRED_enc_part (C type), 136
krb5 cred_enc_part.magic (C member), 136
krb5 cred_enc_part.nonce (C member), 136
krb5 cred_enc_part_r_address (C member), 136
krb5 cred_enc_part_s_address (C member), 136
krb5 cred_enc_part.ticket_info (C member), 136
krb5 cred_enc_part.timestamp (C member), 136
krb5 cred_enc_part.usec (C member), 136
krb5 cred_info (C type), 136
krb5 cred_info.caddrs (C member), 137
krb5 cred_info.client (C member), 136
krb5 cred_info.flags (C member), 136
krb5 cred_info.magic (C member), 136
krb5 cred_info.server (C member), 136
krb5 cred_info.session (C member), 136
krb5 cred_info.times (C member), 137
krb5 creds (C type), 137
krb5 creds.addresses (C member), 137
krb5 creds.authdata (C member), 137
krb5 creds.client (C member), 137
krb5 creds.is_skey (C member), 137
krb5 creds.keyblock (C member), 137
krb5 creds.magic (C member), 137
krb5 creds.second_ticket (C member), 137
krb5 creds.server (C member), 137
krb5 creds.ticket (C member), 137
krb5 creds.ticket_flags (C member), 137
krb5 creds.times (C member), 137
krb5 crypto iov (C type), 137
krb5 crypto iov.data (C member), 138
krb5 crypto iov.flags (C member), 138
KRB5_CRYPTO_TYPE_CHECKSUM (built-in variable), 173
KRB5_CRYPTO_TYPE_DATA (built-in variable), 173
KRB5_CRYPTO_TYPE_EMPTY (built-in variable), 173
KRB5_CRYPTO_TYPE_HEADER (built-in variable), 173
KRB5_CRYPTO_TYPE_PADDING (built-in variable), 173
KRB5_CRYPTO_TYPE_SIGNONLY (built-in variable), 173
KRB5_CRYPTO_TYPE_STREAM (built-in variable), 173
KRB5_CRYPTO_TYPE TRAILER (built-in variable), 174
krb5 cryptotype (C type), 138
KRB5_CYBERSAFE_SECUREID (built-in variable), 174
krb5_data (C type), 138
krb5_data.data (C member), 138
krb5_data.length (C member), 138
krb5_data.magic (C member), 138
krb5 decode_authdata_container (C function), 113
krb5 decode_ticket (C function), 114

Index
Index 211
Index

krb5_get_init_creds_opt (C type), 142
krb5_get_init_creds_opt.address_list (C member), 142
krb5_get_init_creds_opt.etype_list (C member), 142
krb5_get_init_creds_opt.etype_list_length (C member), 142
krb5_get_init_creds_opt.flags (C member), 142
krb5_get_init_creds_opt.forwardable (C member), 142
krb5_get_init_creds_opt.preauth_list (C member), 143
krb5_get_init_creds_opt.preauth_list_length (C member), 143
krb5_get_init_creds_opt.proxiable (C member), 142
krb5_get_init_creds_opt.renew_life (C member), 142
krb5_get_init_creds_opt.salt (C member), 143
krb5_get_init_creds_opt.tkt_life (C member), 142
KRB5_GET_INIT_CREDS_OPT_ADDRESS_LIST (built-in variable), 175
krb5_get_init_creds_opt_alloc (C function), 24
KRB5_GET_INIT_CREDS_OPT_ANONYMOUS (built-in variable), 176
KRB5_GET_INIT_CREDS_OPT_CANONICALIZE (built-in variable), 176
KRB5_GET_INIT_CREDS_OPT_CHG_PWD_PRMPT (built-in variable), 176
KRB5_GET_INIT_CREDS_OPT_ETYPE_LIST (built-in variable), 176
KRB5_GET_INIT_CREDS_OPT_FORWARDABLE (built-in variable), 176
krb5_get_init_creds_opt.free (C function), 24
krb5_get_init_creds_opt.get_fast_flags (C function), 25
krb5_get_init_creds_opt.init (C function), 129
KRB5_GET_INIT_CREDS_OPT_PREAUTH_LIST (built-in variable), 176
KRB5_GET_INIT_CREDS_OPT_PROXIABLE (built-in variable), 176
KRB5_GET_INIT_CREDS_OPT_RENEW_LIFE (built-in variable), 176
KRB5_GET_INIT_CREDS_OPT_SALT (built-in variable), 177
krb5_get_init_creds_opt.set_address_list (C function), 25
krb5_get_init_creds_opt.set_anonymous (C function), 25
krb5_get_init_creds_opt.set_change_password_prompt (C function), 25
krb5_get_init_creds_opt.set_etype (C function), 26
krb5_get_init_creds_opt.set_expire_callback (C function), 26
krb5_get_init_creds_opt.set_fast_ccache (C function), 26
krb5_get_init_creds_opt.set_fast_ccache_name (C function), 27
krb5_get_init_creds_opt.set_fast_flags (C function), 27
krb5_k_create_key (C function), 116
krb5_k_decrypt (C function), 116
krb5_k_decrypt_iov (C function), 117
krb5_k_encrypt (C function), 117
krb5_k_encrypt_iov (C function), 118
krb5_k_free_key (C function), 118
krb5_k_key_enctype (C function), 119
krb5_k_key_keyblock (C function), 119
krb5_k_make_checksum (C function), 119
krb5_k_make_checksum_iov (C function), 119
krb5_k_prf (C function), 120
krb5_k_reference_key (C function), 120
krb5_k_verify_checksum (C function), 121
krb5_k_verify_checksum_iov (C function), 121
krb5_kdc_rep (C type), 143
krb5_kdc_rep.client (C member), 143
krb5_kdc_rep.enc_part (C member), 144
krb5_kdc_rep.enc_part2 (C member), 144
krb5_kdc_rep.magic (C member), 143
krb5_kdc_rep.msg_type (C member), 143
krb5_kdc_rep.ticket (C member), 143
krb5_kdc_rep.padata (C member), 143
krb5_kdc_rep.padata (C member), 143
krb5_key (C type), 144
krb5_keyblock (C type), 145
krb5_keyblock.contents (C member), 145
krb5_keyblock.enctype (C member), 145
krb5_keyblock.length (C member), 145
krb5_keyblock.magic (C member), 145
krb5_keytab (C type), 145
krb5_keytab_entry (C type), 145
krb5_keytab_entry.key (C member), 145
krb5_keytab_entry.magic (C member), 145
krb5_keytab_entry.principal (C member), 145
krb5_keytab_entry.timestamp (C member), 145
krb5_keytab_entry.vno (C member), 145
krb5_keyusage (C type), 146
KRB5_KEYUSAGE_AD_KDC_ISSUED_CKSUM (built-in variable), 178
KRB5_KEYUSAGE_AD_MTE (built-in variable), 178
KRB5_KEYUSAGE_AD_SIGNEDPATH (built-in variable), 178
KRB5_KEYUSAGE_AP_REQ_ENCPART (built-in variable), 178
KRB5_KEYUSAGE_AP_REQ_AUTH (built-in variable), 179
KRB5_KEYUSAGE_AP_REQ_AUTH_CKSUM (built-in variable), 179
KRB5_KEYUSAGE_APP_DATA_CKSUM (built-in variable), 178
KRB5_KEYUSAGE_APP_DATA_ENCRYPT (built-in variable), 178
KRB5_KEYUSAGE_AS_REQ (built-in variable), 179
KRB5_KEYUSAGE_AS_REQ_PA_ENC_TS (built-in variable), 179
KRB5_KEYUSAGE_IAKERB_FINISHED (built-in variable), 179
KRB5_KEYUSAGE_KDC_REP_TICKET (built-in variable), 180
KRB5_KEYUSAGE_KRB_CRED_ENCPART (built-in variable), 181
KRB5_KEYUSAGE_KRB_ERROR_CKSUM (built-in variable), 181
KRB5_KEYUSAGE_KRB_PRIV_ENCPART (built-in variable), 181
KRB5_KEYUSAGE_KRB_SAFE_CKSUM (built-in variable), 181
KRB5_KEYUSAGE_PA_OTP_REQUEST (built-in variable), 181
KRB5_KEYUSAGE_PA_PKINIT_KX (built-in variable), 181
KRB5_KEYUSAGE_PA_S4U_X509_USER_REPLY (built-in variable), 181

Index 213
KRB5_RESPONDER_OTP_FORMAT_ALPHANUMERIC (built-in variable), 197
KRB5_RESPONDER_OTP_FORMAT_DECIMAL (built-in variable), 197
KRB5_RESPONDER_OTP_FORMAT_HEXADECIMAL (built-in variable), 197
krb5_responder_otp_get_challenge (C function), 39
krb5_responder_otp_set_answer (C function), 40
krb5_responder_otp_tokeninfo (C type), 152
krb5_responder_otp_tokeninfo.alg_id (C member), 152
krb5_responder_otp_tokeninfo.challenge (C member), 152
krb5_responder_otp_tokeninfo.flags (C member), 152
krb5_responder_otp_tokeninfo.format (C member), 152
krb5_responder_otp_tokeninfo.length (C member), 152
krb5_responder_otp_tokeninfo.token_id (C member), 152
krb5_responder_otp_tokeninfo.vendor (C member), 152
KRB5_RESPONDER_QUESTION_OTP (built-in variable), 198
KRB5_RESPONDER_QUESTION_PASSWORD (built-in variable), 198
krb5_responder_set_answer (C function), 39
krb5_response (C type), 152
krb5_response.expected_nonce (C member), 153
krb5_response.magic (C member), 153
krb5_response.message_type (C member), 153
krb5_response.request_time (C member), 153
krb5_response.response (C member), 153
krb5_roundup (built-in variable), 206
KRB5_SAFE (built-in variable), 198
krb5_salttype_to_string (C function), 95
KRB5_SAM_MUST_PK_ENCRYPT_SAD (built-in variable), 198
KRB5_SAM_SEND_ENCRYPTED_SAD (built-in variable), 198
KRB5_SAM_USE_SAD_AS_KEY (built-in variable), 198
krb5_sendauth (C function), 123
krb5_server_decrypt_ticket_keytab (C function), 95
KRB5_TIMEOFDAY (C function), 97
krb5_string_to_key (C function), 130
krb5_string_to_salttype (C function), 97
krb5_string_to_timestamp (C function), 97
KRB5_TC_MATCH_2ND_TKT (built-in variable), 199
KRB5_TC_MATCH_AUTHDATA (built-in variable), 199
KRB5_TC_MATCH_FLAGS (built-in variable), 199
KRB5_TC_MATCH_FLAGS_EXACT (built-in variable), 199
KRB5_TC_MATCH_IS_SKEY (built-in variable), 199
KRB5_TC_MATCH_KTYPE (built-in variable), 199
KRB5_TC_MATCH_SRV_NAMEONLY (built-in variable), 199
KRB5_TC_MATCH_TIMES (built-in variable), 200
KRB5_TC_MATCH_TIMES_EXACT (built-in variable), 200
KRB5_TC_NOTICKET (built-in variable), 200
KRB5_TC_OPENCLOSE (built-in variable), 200
KRB5_TC_SUPPORTED_KTYPES (built-in variable), 200
KRB5_TGS_NAME (built-in variable), 200
KRB5_TGS_NAME_SIZE (built-in variable), 200
KRB5_TGS_REQ (built-in variable), 201
KRB5_TGS_REP (built-in variable), 201
krb5_ticket (C type), 153
krb5_ticket.enc_part (C member), 153
krb5_ticket.enc_part2 (C member), 154
krb5_ticket.magic (C member), 153
krb5_ticket.server (C member), 153
krb5_ticket_times (C type), 154
krb5_ticket_times.authtime (C member), 154
krb5_ticket_times.endtime (C member), 154
krb5_ticket_times.renew_till (C member), 154
krb5_ticket_times.starttime (C member), 154
krb5_ticket_times (C type), 154
KR5_TIMEOFDAY (C function), 97
krb5_timestamp (C type), 154
krb5_timestamp_to_sfstring (C function), 98
krb5_timestamp_to_string (C function), 98
krb5_tkt_authent (C type), 154
krb5_tkt_authent.ap_options (C member), 155
krb5_tkt_authent.authenticator (C member), 155
krb5_tkt_authent.magic (C member), 155
krb5_tkt_authent.ticket (C member), 155
krb5_tkt_creds_context (C type), 159
krb5_tkt_creds_free (C function), 98
krb5_tkt_creds_get (C function), 98
krb5_tkt_creds_free (C function), 98
krb5_tkt_creds_get_creds (C function), 99
krb5_tkt_creds_get (C function), 99
krb5_tkt_creds_get_times (C function), 99
krb5_tkt_creds_init (C function), 99
krb5_tkt_creds_step (C function), 100
KRB5_TKT_CREDS_STEP_FLAG_CONTINUE (built-in variable), 201
krb5_trace_callback (C type), 155
krb5_trace_info (C type), 155
Index

Kerberos Application Developer Guide, Release 1.11.1

krb5_trace_info.message (C member), 155
krb5_transited (C type), 155
krb5_transited_magic (C member), 156
krb5_transited_tr_contents (C member), 156
krb5_transited_tr_type (C member), 156
krb5_transited_data (C type), 156
krb5_transited_data_magic (C member), 156
krb5_transited_data_magic (C member), 156
krb5_transited_data_type (C member), 156
krb5_ui_4 (C type), 156
krb5_unparse_name (C function), 44
krb5_unparse_name_ext (C function), 44
krb5_unparse_name_flags (C function), 45
krb5_unparse_name_flags_ext (C function), 45
krb5_us_timeofday (C function), 45
krb5_use_encrypt (C function), 130
krb5_verify_authdata_kdc_issued (C function), 46
krb5_verify_checksum (C function), 131
krb5_verify_init_creds (C function), 100
krb5_verify_init_creds_opt (C type), 156
krb5_verify_init_creds_opt_init (C function), 101
krb5_verify_init_creds_opt_init (C function), 101
krb5_verify_init_creds_opt_set_ap_req_nofail (C function), 101
krb5_verify_init_creds_opt_set_ap_req_nofail (C function), 101
krb5_vset_error_message (C function), 101
KRB5_VERIFY_INIT_CREDS_OPT_AP_REQ_NOFAIL (built-in variable), 201
KRB5_VERIFY_INIT_CREDS_OPT_AP_REQ_NOFAIL (built-in variable), 201

L
LR_TYPE_INTERPRETATION_MASK (built-in variable), 201
LR_TYPE_THIS_SERVER_ONLY (built-in variable), 201

M
MAX_KEYTAB_NAME_LEN (built-in variable), 202
MSEC_DIRBIT (built-in variable), 202
MSEC_VAL_MASK (built-in variable), 202

P
passwd_phrase_element (C type), 157
passwd_phrase_element_magic (C member), 157
passwd_phrase_element_passwd (C member), 157
passwd_phrase_element_phrase (C member), 157

R
RFC

RFC 2743, 1
RFC 2744, 1

S
SALT_TYPE_AFS_LENGTH (built-in variable), 202
SALT_TYPE_NO_LENGTH (built-in variable), 202

T
THREEPARAMOPEN (built-in variable), 202
TKT_FLG_ANONYMOUS (built-in variable), 202
TKT_FLG_ENC_PA_REP (built-in variable), 202
TKT_FLG_FORWARDABLE (built-in variable), 203
TKT_FLG_FORWARDED (built-in variable), 203
TKT_FLG_HW_AUTH (built-in variable), 203
TKT_FLG_INITIALIZ (built-in variable), 203
TKT_FLG_INVALID (built-in variable), 203
TKT_FLG_MAY_POSTDATE (built-in variable), 203
TKT_FLG_OK_AS_DELEGATE (built-in variable), 203
TKT_FLG_POSTDATED (built-in variable), 203
TKT_FLG_PRE_AUTH (built-in variable), 204
TKT_FLG_PROXY (built-in variable), 204
TKT_FLG_PROXY (built-in variable), 204
TKT_FLG_RENEWABLE (built-in variable), 204
TKT_FLG_TRANSIT_POLICY_CHECKED (built-in variable), 204

V
VALID_INT_BITS (built-in variable), 204
VALID_UINT_BITS (built-in variable), 204