

# Kerberos Administration System

## KADM5 API Functional Specifications\*

Barry Jaspan

August 3, 2011

### Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Versions of the API</b>	<b>2</b>
<b>3</b>	<b>Policies and Password Quality</b>	<b>3</b>
<b>4</b>	<b>Data Structures</b>	<b>4</b>
4.1	Principals, <code>kadm5_principal_ent_t</code> . . . . .	4
4.2	Policies, <code>kadm5_policy_ent_t</code> . . . . .	8
4.3	Configuration parameters . . . . .	9
4.4	Principal keys . . . . .	11
4.5	Field masks . . . . .	12
<b>5</b>	<b>Constants, Header Files, Libraries</b>	<b>14</b>
<b>6</b>	<b>Error Codes</b>	<b>15</b>
<b>7</b>	<b>Authentication and Authorization</b>	<b>18</b>
<b>8</b>	<b>Functions</b>	<b>19</b>
8.1	Overview . . . . .	19
8.2	<code>kadm5_init_*</code> . . . . .	21
8.3	<code>kadm5_flush</code> . . . . .	24
8.4	<code>kadm5_destroy</code> . . . . .	25
8.5	<code>kadm5_create_principal</code> . . . . .	25

---

\*api-funcspec.tex 17363 2005-08-29 19:22:52Z hartmans

8.6	kadm5_delete_principal . . . . .	27
8.7	kadm5_modify_principal . . . . .	27
8.8	kadm5_rename_principal . . . . .	29
8.9	kadm5_chpass_principal . . . . .	29
8.10	kadm5_chpass_principal_util . . . . .	31
8.11	kadm5_randkey_principal . . . . .	33
8.12	kadm5_setkey_principal . . . . .	35
8.13	kadm5_get_principal . . . . .	36
8.14	kadm5_decrypt_key . . . . .	37
8.15	kadm5_get_principals . . . . .	37
8.16	kadm5_create_policy . . . . .	38
8.17	kadm5_delete_policy . . . . .	39
8.18	kadm5_modify_policy . . . . .	39
8.19	kadm5_get_policy . . . . .	40
8.20	kadm5_get_policies . . . . .	41
8.21	kadm5_free_principal_ent, _policy_ent . . . . .	41
8.22	kadm5_free_name_list . . . . .	42
8.23	kadm5_free_key_data . . . . .	42
8.24	kadm5_get_privs . . . . .	42

## 1 Introduction

This document describes the Admin API that can be used to maintain principals and policies. It describes the data structures used for each function and the interpretation of each data type field, the semantics of each API function, and the possible return codes.

The Admin API is intended to be used by remote clients using an RPC interface. It is implemented by the admin server running on the Kerberos master server. It is also possible for a program running on the Kerberos master server to use the Admin API directly, without going through the admin server.

## 2 Versions of the API

The versions of this API and a brief description of the changes for each are:

**KADM5\_API\_VERSION\_1** The initial version of this API, written by OpenVision Technologies and donated to MIT for including in the public release. Originally called OVSEC.KADM\_API\_VERSION\_1. Most everything has been renamed in one way or

another, including functions, header files, and data structures. Where possible, the old OVSEC\_KADM names have been left behind for compatibility with version 1, and KADM5\_API\_VERSION\_1 is compatible with OVSEC\_KADM\_API\_VERSION\_1 at compile-, link-, and run-time.

The OVSEC\_KADM name compatibility will not be extended to new functionality in future versions because no existing OVSEC\_KADM clients will use that functionality; new clients should be written to the KADM5 API.

**KADM5\_API\_VERSION\_2** This version contains the initial changes necessary to make the OpenVision administration system work with the mid-1996 MIT version of Kerberos 5. Changes include

1. The `kadm5_init` functions now take a structure of parameters instead of just a realm name, allowing the calling program to specify non-default values for various configuration options. See section 4.3 for details.
2. The KADM5 API has been extended to support new features of the Kerberos database, including multiple encryption and salt types per principal. See section 4.4 for details.
3. `kadm5_get_principal` now allows a principal's keys to be retrieved *by local clients only*. This is necessary in order for the `kadm5` API to provide the primary Kerberos database interface.
4. The KADM5 authorization system has been completely changed.
5. The functions `kadm5_flush`, `kadm5_get_principals`, and `kadm5_get_policies` have been added.
6. The KADM5 API now obeys a caller-allocates rather than callee-allocates system. `kadm5_get_principal` and `kadm5_get_policy` are affected.

### 3 Policies and Password Quality

The Admin API Password Quality mechanism provides the following controls. Note that two strings are defined to be “significantly different” if they differ by at least one character. The compare is not case sensitive.

- A minimum length can be required; a password with fewer than the specified number of characters will not be accepted.
- A minimum number of character classes can be required; a password that does not contain at least one character from at least the specified number of character classes will not be accepted. The character classes are defined by `islower()`, `isupper()`, `isdigit()`, `ispunct()`, and other.

- Passwords can be required to be different from previous passwords; a password that generates the same encryption key as any of the principal's specified previous number of passwords will not be accepted. This comparison is performed on the encryption keys generated from the passwords, not on the passwords themselves.
- A single “forbidden password” dictionary can be specified for all users; a password that is not significantly different from every word in the dictionary will not be accepted.

## 4 Data Structures

This section describes the data structures used by the Admin API. They are defined in `<kadm5/admin.h>`.

### 4.1 Principals, `kadm5_principal_ent_t`

A Kerberos principal entry is represented by a `kadm5_principal_ent_t`. It contains a subset of the information stored in the master Kerberos database as well as the additional information maintained by the admin system. In the current version, the only additional information is the principal's policy and the `aux_attributes` flags.

The principal may or may not have a policy enforced on it. If the `POLICY` bit (see section 4.5) is set in `aux_attributes`, the `policy` field names the principal's policy. If the `POLICY` bit is not set in `aux_attributes`, no policy is enforced on the principal and the value of the `policy` field is undefined.

The fields of an `kadm5_principal_ent_t` are interpreted as follows.

**principal** The name of the principal; must conform to Kerberos naming specifications.

**princ\_expire\_time** The expire time of the principal as a Kerberos timestamp. No Kerberos tickets will be issued for a principal after its expire time.

**last\_pwd\_change** The time this principal's password was last changed, as a Kerberos timestamp.

**pw\_expiration** The expire time of the user's current password, as a Kerberos timestamp. No application service tickets will be issued for the principal once the password expire time has passed. Note that the user can only obtain tickets for services that have the `PW_CHANGE_SERVICE` bit set in the `attributes` field.

**max\_life** The maximum lifetime of any Kerberos ticket issued to this principal.

```

typedef struct _kadm5_principal_ent_t {
    krb5_principal principal;

    krb5_timestamp princ_expire_time;
    krb5_timestamp last_pwd_change;
    krb5_timestamp pw_expiration;
    krb5_deltat max_life;
    krb5_principal mod_name;
    krb5_timestamp mod_date;
    krb5_flags attributes;
    krb5_kvno kvno;
    krb5_kvno mkvno;

    char * policy;
    u_int32 aux_attributes;

    krb5_deltat max_renewable_life;
    krb5_timestamp last_success;
    krb5_timestamp last_failed;
    krb5_kvno fail_auth_count;
    krb5_int16 n_key_data;
    krb5_int16 n_tl_data;
    krb5_tl_data *tl_data;
    krb5_key_data *key_data;
} kadm5_principal_ent_rec, *kadm5_principal_ent_t;

```

Figure 1: Definition of kadm5\_principal\_ent\_t.

**attributes** A bitfield of attributes for use by the KDC. The symbols and constant values are defined below; their interpretation appears in the libkdb functional specification.

Name	Value
KRB5_KDB_DISALLOW_POSTDATED	0x00000001
KRB5_KDB_DISALLOW_FORWARDABLE	0x00000002
KRB5_KDB_DISALLOW_TGT_BASED	0x00000004
KRB5_KDB_DISALLOW_RENEWABLE	0x00000008
KRB5_KDB_DISALLOW_PROXIABLE	0x00000010
KRB5_KDB_DISALLOW_DUP_SKEY	0x00000020
KRB5_KDB_DISALLOW_ALL_TIX	0x00000040
KRB5_KDB_REQUIRES_PRE_AUTH	0x00000080
KRB5_KDB_REQUIRES_HW_AUTH	0x00000100
KRB5_KDB_REQUIRES_PWCHANGE	0x00000200
KRB5_KDB_DISALLOW_SVR	0x00001000
KRB5_KDB_PWCHANGE_SERVICE	0x00002000
KRB5_KDB_SUPPORT_DESMD5	0x00004000
KRB5_KDB_NEW_PRINC	0x00008000

**mod\_name** The name of the Kerberos principal that most recently modified this principal.

**mod\_date** The time this principal was last modified, as a Kerberos timestamp.

**kvno** The version of the principal's current key.

**mkvno** The version of the Kerberos Master Key in effect when this principal's key was last changed. In KADM5\_API\_VERSION\_2, this field is always zero.

**policy** If the POLICY bit is set in aux.attributes, the name of the policy controlling this principal.

**aux.attributes** A bitfield of flags for use by the administration system. Currently, the only valid flag is POLICY, and it indicates whether or not the principal has a policy enforced on it.

**max\_renewable\_life** The maximum renewable lifetime of any Kerberos ticket issued to or for this principal. This field only exists in KADM5\_API\_VERSION\_2.

**last\_success** The KDC time of the last successful AS\_REQ. This is only updated if KRBCONF\_KDC\_MODIFIES\_KDB is defined during compilation of the KDC. This field only exists in KADM5\_API\_VERSION\_2.

**last\_failed** The KDC time of the last failed AS\_REQ. This is only updated if KRBCONF\_KDC\_MODIFIES\_KDB is defined during compilation of the KDC. This field only exists in KADM5\_API\_VERSION\_2.

**fail\_auth\_count** The number of consecutive failed AS\_REQs. When this number reaches KRB5\_MAX\_FAIL\_COUNT, the KRB5\_KDC\_DISALLOW\_ALL\_TIX is set on the principal. This is only updated if KRBCONF\_KDC\_MODIFIES\_KDB is defined during compilation. This field only exists in KADM5\_API\_VERSION\_2.

**n\_tl\_data** The number of elements in the `tl_data` linked list. This field only exists in KADM5\_API\_VERSION\_2.

**n\_key\_data** The number of elements in the `key_data` array. This field only exists in KADM5\_API\_VERSION\_2.

**tl\_data** A linked list of tagged data. This list is a mechanism by which programs can store extended information in a principal entry, without having to modify the database API. Each element is of type `krb5_tl_data`:

```
typedef struct _krb5_tl_data {
    struct _krb5_tl_data* tl_data_next;
    krb5_int16             tl_data_type;
    krb5_int16             tl_data_length;
    krb5_octet             * tl_data_contents;
} krb5_tl_data;
```

The KADM5 API only allows elements whose `tl_data_type` is greater than or equal to 256. Values less than 256 are reserved for internal use by the KADM5 or kdb system. They are filtered out of the list returned by `kadm5_get_principal`, and generate an error if given to `kadm5_modify_principal`.

The `libkdb` library defines the tagged data types `KRB5_TL_LAST_PWD_CHANGE`, `KRB5_TL_MOD_PRINC`, and `KRB5_TL_KADM_DATA`, all with values less than 256, which store the last password modification time, time and modifier of last principal modification, and administration system data. All of these entries are expected by the administration system and parsed out into fields of the `kadm5_principal_ent_rec` structure; as described above, they are not included in the `tl_data` list.

Tagged data elements with types greater than 256 are handled without interpretation by KADM5. Note that an application that calls `kadm5_modify_principal` with the `KADM5_TL_DATA` mask bit set is responsible for providing the *complete* `tl_data` list, which it necessarily must obtain from `kadm5_get_principal`. It is *never* possible for an application to construct a complete `tl_data` list from scratch.

**key\_data** An array of the principal's keys. The keys contained in this array are encrypted in the Kerberos master key. See section 4.4 for a discussion of the `krb5_key_data` structure.

## 4.2 Policies, `kadm5_policy_ent_t`

If the `POLICY` bit is set in `aux.attributes`, the `policy` name field in the `kadm5_principal_ent_t` structure refers to a password policy entry defined in a `kadm5_policy_ent_t`.

```
typedef struct _kadm5_policy_ent_t {
    char *policy;

    u_int32 pw_min_life;
    u_int32 pw_max_life;
    u_int32 pw_min_length;
    u_int32 pw_min_classes;
    u_int32 pw_history_num;
    u_int32 policy_refcnt;
} kadm5_policy_ent_rec, *kadm5_policy_ent_t;
```

The fields of an `kadm5_policy_ent_t` are interpreted as follows. Note that a policy's values only apply to a principal using that policy.

**policy** The name of this policy, as a NULL-terminated string. The ASCII characters between 32 (space) and 126 (tilde), inclusive, are legal.

**pw\_min\_life** The minimum password lifetime, in seconds. A principal cannot change its password before `pw_min_life` seconds have passed since `last_pwd_change`.

**pw\_max\_life** The default duration, in seconds, used to compute `pw_expiration` when a principal's password is changed.

**pw\_min\_length** The minimum password length, in characters. A principal cannot set its password to anything with fewer than this number of characters. This value must be greater than zero.

**pw\_min\_classes** The minimum number of character classes in the password. This value can only be 1, 2, 3, 4, or 5. A principal cannot set its password to anything with fewer than this number of character classes in it.

**pw\_history\_num** The number of past passwords that are stored for the principal; the minimum value is 1 and the maximum value is 10. A principal cannot set its password to any of its previous `pw_history_num` passwords. The first "previous" password is the current password; thus, a principal with a policy can never reset its password to its current value.

**policy\_refcnt** The number of principals currently using this policy. A policy cannot be deleted unless this number is zero.



### 4.3 Configuration parameters

The KADM5 API acquires configuration information from the Kerberos configuration file (\$KRB5\_CONFIG or DEFAULT\_PROFILE\_PATH) and from the KDC configuration file (\$KRB5\_KDC\_CONFIG or DEFAULT\_KDC\_PROFILE). In KADM5\_API\_VERSION\_2, some of the configuration parameters used by the KADM5 API can be controlled by the caller by providing a `kadm5_config_params` structure to `kadm5_init`:

```
typedef struct _kadm5_config_params {
    u_int32 mask;

    /* Client and server fields */
    char *realm;
    char *profile;
    int kadmind_port;

    /* client fields */
    char *admin_server;

    /* server fields */
    char *dbname;
    char *admin_dbname;
    char *admin_lockfile;
    char *acl_file;
    char *dict_file;
    char *admin_keytab;

    /* server library (database) fields */
    int mkey_from_kbd;
    char *stash_file;
    char *mkey_name;
    krb5_enctype enctype;
    krb5_deltat max_life;
    krb5_deltat max_rlife;
    krb5_timestamp expiration;
    krb5_flags flags;
    krb5_key_salt_tuple *keysalts;
    krb5_int32 num_keysalts;
} kadm5_config_params;
```

The following list describes each of the fields of the structure, along with the profile relation it overrides, its mask value, its default value, and whether it is valid on the client, server, or

both, or neither.

**mask** No variable. No mask value. A bitfield specifying which fields of the structure contain valid information. A caller sets this mask before calling `kadm5_init_*`, indicating which parameters are specified. The mask values are defined in `<kadm5/admin.h>` and are all prefixed with `KADM5_CONFIG_`; the prefix is not included in the descriptions below.

**realm** No variable. `REALM`. Client and server. The realm to which these parameters apply, and the realm for which additional parameters are to be acquired, if any. If this field is not specified in the mask, the default local realm is used.

**profile** Variable: `profile` (server only). `PROFILE`. Client and server. The Kerberos profile to use. On the client, the default is the value of the `KRB5_CONFIG` environment variable, or `DEFAULT_PROFILE_PATH` if that is not set. On the server, the value of the “profile” variable of the KDC configuration file will be used as the first default if it exists; otherwise, the default is the value of the `KRB5_KDC_PROFILE` environment variable or `DEFAULT_KDC_PROFILE`.

**kadmind\_port** Variable: `kadmind_port`. `KADMIND_PORT`. Client and server. The port number the kadmind server listens on. The client uses this field to determine where to connect, and the server to determine where to listen. The default is 749, which has been assigned by IANA.

**admin\_server** Variable: `admin_server`. `ADMIN_SERVER`. Client. The host name of the admin server to which to connect. There is no default. If the value of this field contains a colon (`:`), the text following the colon is treated as an integer and assigned to the `kadmind_port` field, overriding any value of the `kadmind_port` variable.

**dbname** Variable: `dbname`. `DBNAME`. Server. The Kerberos database name to use; the Kerberos database stores principal information. The default is `DEFAULT_KDB_FILE`.

**admin\_dbname** Variable: `admin_database_name`. `ADBNAM`. Neither. If the `dbname` field is set, this field is set to the value of `dbname` followed by “.kadm5”.

**admin\_lockfile** Variable: `admin_database_lockfile`. `ADB_LOCKFILE`. Neither. If the `admin_dbname` field is set, this field is set to the value of `admin_dbname` followed by “.lock”.

**acl\_file** Variable: `acl_file`. `ACL_FILE`. Server. The admin server’s ACL file. The default is `DEFAULT_KADM5_ACL_FILE`.

**dict\_file** Variable: `admin_dict_file`. `DICT_FILE`. Server. The admin server’s dictionary file of passwords to disallow. No default.

**admin\_keytab** Variable: `admin_keytab`. `ADMIN_KEYTAB`. Server. The keytab file containing the `kadmin/admin` and `kadmin/changepw` entries for the server to use. The default is the value of the `KRB5_KTNAME` environment variable, if defined, else `DEFAULT_KADM5_KEYTAB`.

**mkey\_from\_keyboard** No variable. `MKEY_FROM_KEYBOARD`. Server. If non-zero, prompt for the master password via the tty instead of using the stash file. If this mask bit is not set, or is set and the value is zero, the stash file is used.

**stash\_file** Variable: `key_stash_file`. `STASH_FILE`. Server. The file name containing the master key stash file. No default; libkdb will work with a NULL value.

**mkey\_name** Variable: `master_key_name`. `MKEY_NAME`. Server. The name of the master principal for the realm. No default; libkdb will work with a NULL value.

**enctype** Variable: `master_key_type`. `ENCTYPE`. Server. The encryption type of the master principal. The default is `DEFAULT_KDC_ENCTYPE`.

**max\_life** Variable: `max_life`. `MAX_LIFE`. Maximum lifetime for all tickets issued to the principal. The default is 28800, which is 8 hours.

**max\_rlife, expiration, flags** Variables: `max_renewable_life`, `default_principal_expiration`, `default_principal_flags`. `MAX_LIFE`, `MAX_RLIFE`, `EXPIRATION`, `FLAGS`. Server. Default values for new principals. All default to 0.

**keysalts, num\_keysalts** Variable: `supported_encypes`. `ENCYPES`. Server. The list of supported encryption type/salt type tuples; both fields must be assigned if `ENCYPES` is set. The default is a list containing one enctype, `DES-CBC-CRC` with normal salt.

## 4.4 Principal keys

In `KADM5_API_VERSION_1`, all principals had a single key. The encryption method was always DES, and the salt type was determined outside the API (by command-line options to the administration server).

In `KADM5_API_VERSION_2`, principals can have multiple keys, each with its own encryption type and salt. Each time a principal's key is changed with `kadm5_create_principal`, `kadm5_chpass_principal` or `kadm5_randkey_principal`, existing key entries are removed and a key entry for each encryption and salt type tuple specified in the configuration parameters is added. There is no provision for specifying encryption and salt type information on a per-principal basis; in a future version, this will probably be part of the admin policy. There is also presently no provision for keeping multiple key versions for a single principal active in the database.

A single key is represented by a `krb5_key_data`:

```
typedef struct _krb5_key_data {
    krb5_int16      key_data_ver;           /* Version */
    krb5_int16      key_data_kvno;         /* Key Version */
    krb5_int16      key_data_type[2];      /* Array of types */
    krb5_int16      key_data_length[2];    /* Array of lengths */
    krb5_octet      * key_data_contents[2]; /* Array of pointers */
} krb5_key_data;
```

**key\_data\_ver** The version number of the structure. Versions 1 and 2 are currently defined. If `key_data_ver` is 1 then the key is either a random key (not requiring a salt) or the salt is the normal v5 salt which is the same as the realm and therefore doesn't need to be saved in the database.

**key\_data\_kvno** The key version number of this key.

**key\_data\_type** The first element is the enctype of this key. In a version 2 structure, the second element is the salttype of this key. The legal encryption types are defined in `<krb5.h>`. The legal salt types are defined in `<k5-int.h>`.

**key\_data\_length** The first element is length this key. In a version 2 structure, the second element is length of the salt for this key.

**key\_data\_contents** The first element is the content of this key. In a version 2 structure, the second element is the contents of the salt for this key.

## 4.5 Field masks

The API functions for creating, retrieving, and modifying principals and policies allow for a relevant subset of the fields of the `kadm5_principal_ent_t` and `kadm5_policy_ent_t` to be specified or changed. The chosen fields are determined by a bitmask that is passed to the relevant function. Each API function has different rules for which mask values can be specified, and can specify whether a given mask value is mandatory, optional, or forbidden. Mandatory fields must be present and forbidden fields must not be present or an error is generated. When creating a principal or policy, optional fields have a default value if they are not specified. When modifying a principal or policy, optional fields are unchanged if they are not specified. When retrieving a principal, optional fields are simply not provided if they are not specified; not specifying undeclared fields for retrieval may improve efficiency. The values for forbidden fields are defined in the function semantics.

The masks for principals are in table 1 and the masks for policies are in table 2. They are defined in `<kadm5/admin.h>`. The KADM5\_ prefix has been removed from the Name fields. In the Create and Modify fields, M means mandatory, F means forbidden, and O means optional. Create fields that are optional specify the default value. The notation “K/M value” means that the field inherits its value from the corresponding field in the Kerberos master principal, for KADM5\_API\_VERSION\_1, and from the configuration parameters for KADM5\_API\_VERSION\_2.

All masks for principals are optional for retrieval, *except* that the KEY\_DATA mask is illegal when specified by a remote client; for details, see the function semantics for `kadm5_get_principal`.

Note that the POLICY and POLICY\_CLR bits are special. When POLICY is set, the policy is assigned to the principal. When POLICY\_CLR is specified, the policy is unassigned to the principal and as a result no policy controls the principal.

For convenience, the mask KADM5\_PRINCIPAL\_NORMAL\_MASK contains all of the principal masks *except* KADM5\_KEY\_DATA and KADM5\_TL\_DATA, and the mask KADM5\_POLICY\_NORMAL\_MASK contains all of the policy masks.

Name	Value	Fields Affected	Create	Modify
PRINCIPAL	0x000001	principal	M	F
PRINC_EXPIRE_TIME	0x000002	princ_expire_time	O, K/M value	O
PW_EXPIRATION	0x000004	pw_expiration	O, now+pw_max_life	O
LAST_PWD_CHANGE	0x000008	last_pwd_change	F	F
ATTRIBUTES	0x000010	attributes	O, 0	O
MAX_LIFE	0x000020	max_life	O, K/M value	O
MOD_TIME	0x000040	mod_date	F	F
MOD_NAME	0x000080	mod_name	F	F
KVNO	0x000100	kvno	O, 1	O
MKVNO	0x000200	mkvno	F	F
AUX_ATTRIBUTES	0x000400	aux_attributes	F	F
POLICY	0x000800	policy	O, none	O
POLICY_CLR	0x001000	policy	F	O
MAX_RLIFE	0x002000	max_renewable_life	O, K/M value	O
LAST_SUCCESS	0x004000	last_success	F	F
LAST_FAILED	0x008000	last_failed	F	F
FAIL_AUTH_COUNT	0x010000	fail_auth_count	F	O
KEY_DATA	0x020000	n_key_data, key_data	F	F
TL_DATA	0x040000	n_tl_data, tl_data	O, 0, NULL	O

Table 1: Mask bits for creating, retrieving, and modifying principals.

Name	Value	Field Affected	Create	Modify
POLICY	same	policy	M	F
PW_MAX_LIFE	0x004000	pw_max_life	O, 0 (infinite)	O
PW_MIN_LIFE	0x008000	pw_min_life	O, 0	O
PW_MIN_LENGTH	0x010000	pw_min_length	O, 1	O
PW_MIN_CLASSES	0x020000	pw_min_classes	O, 1	O
PW_HISTORY_NUM	0x040000	pw_history_num	O, 0	O
REF_COUNT	0x080000	pw_refcnt	F	F

Table 2: Mask bits for creating/modifying policies.

## 5 Constants, Header Files, Libraries

<kadm5/admin.h> includes a number of required header files, including RPC, Kerberos 5, com\_err, and admin com\_err defines. It contains prototypes for all kadm5 routines mentioned below, as well as all Admin API data structures, type definitions and defines mentioned in this document.

Before `#include`ing <kadm5/admin.h>, the programmer can specify the API version number that the program will use by `#define`ing `USE_KADM5_API_VERSION`; for example, define that symbol to be 1 to use `KADM5_API_VERSION_1`. This will ensure that the correct functional prototypes and data structures are defined. If no version symbol is defined, the most recent version supported by the header files will be used.

Some of the defines and their values contained in <kadm5/admin.h> include the following, whose `KADM5_` prefixes have been removed. Symbols that do not exist in `KADM5_API_VERSION_2` do not have a `KADM5_` prefix, but instead retain only with `OVSEC_KADM_` prefix for compatibility.

**admin service principal** `ADMIN_SERVICE` (“kadmin/admin”)

**admin history key** `HIST_PRINCIPAL` (“kadmin/history”)

**change password principal** `CHANGEPW_SERVICE` (“kadmin/angepw”)

**server acl file path** `ACLFILE` (“/krb5/ovsec\_adm.acl”). In `KADM5_API_VERSION 2`, this is controlled by configuration parameters.

**dictionary** `WORDFILE` (“/krb5/kadmind.dict”). In `KADM5_API_VERSION 2`, this is controlled by configuration parameters.

KADM5 errors are described in <kadm5/kadm\_err.h>, which is included by <kadm5/admin.h>.

The locations of the admin policy and principal databases, as well as defines and type definitions for the databases, are defined in `<kadm5/adb.h>`. Some of the defines in that file are:

**admin policy database** `POLICY_DB` (`"/krb5/kadm5_policy.db"`). In `KADM5_API_VERSION 2`, this is controlled by configuration parameters.

**admin principal database** `PRINCIPAL_DB` (`"/krb5/ovsec_principal.db"`). In `KADM5_API_VERSION 2`, this is controlled by configuration parameters.

Client applications will link against `libkadm5clnt.a` and server programs against `libkadm5srv.a`. Client applications must also link against: `libgssapi_krb5.a`, `libkrb5.a`, `libcrypto.a`, `libgssrpc.a`, `libcom_err.a`, and `libdyn.a`. Server applications must also link against: `libkdb5.a`, `libkrb5.a`, `libcrypto.a`, `libgssrpc.a`, `libcom_err.a`, and `libdyn.a`.

## 6 Error Codes

The error codes that can be returned by admin functions are listed below. Error codes indicated with a "\*" can be returned by every admin function and always have the same meaning; these codes are omitted from the list presented with each function.

The admin system guarantees that a function that returns an error code has no other side effect.

The Admin system will use `com_err` for error codes. Note that this means `com_err` codes may be returned from functions that the admin routines call (e.g. the kerberos library). Callers should not expect that only KADM5 errors will be returned. The Admin system error code table name will be "ovk", and the offsets will be the same as the order presented here. As mentioned above, the error table include file will be `<kadm5/kadm_err.h>`.

Note that these error codes are also used as protocol error code constants and therefore must not change between product releases. Additional codes should be added at the end of the list, not in the middle. The integer value of `KADM5_FAILURE` is 43787520; the remaining values are assigned in sequentially increasing order.

- \* **KADM5\_FAILURE** Operation failed for unspecified reason
- \* **KADM5\_AUTH\_GET** Operation requires "get" privilege
- \* **KADM5\_AUTH\_ADD** Operation requires "add" privilege
- \* **KADM5\_AUTH\_MODIFY** Operation requires "modify" privilege

- \* **KADM5\_AUTH\_DELETE** Operation requires “delete” privilege
- \* **KADM5\_AUTH\_INSUFFICIENT** Insufficient authorization for operation
- \* **KADM5\_BAD\_DB** Database inconsistency detected
- KADM5\_DUP** Principal or policy already exists
- KADM5\_RPC\_ERROR** Communication failure with server
- KADM5\_NO\_SRV** No administration server found for realm
- KADM5\_BAD\_HIST\_KEY** Password history principal key version mismatch
- KADM5\_NOT\_INIT** Connection to server not initialized
- KADM5\_UNK\_PRINC** Principal does not exist
- KADM5\_UNK\_POLICY** Policy does not exist
- KADM5\_BAD\_MASK** Invalid field mask for operation
- KADM5\_BAD\_CLASS** Invalid number of character classes
- KADM5\_BAD\_LENGTH** Invalid password length
- KADM5\_BAD\_POLICY** Illegal policy name
- KADM5\_BAD\_PRINCIPAL** Illegal principal name.
- KADM5\_BAD\_AUX\_ATTR** Invalid auxillary attributes
- KADM5\_BAD\_HISTORY** Invalid password history count
- KADM5\_BAD\_MIN\_PASS\_LIFE** Password minimum life is greater then password maximum life
- KADM5\_PASS\_Q\_TOOSHORT** Password is too short
- KADM5\_PASS\_Q\_CLASS** Password does not contain enough character classes
- KADM5\_PASS\_Q\_DICT** Password is in the password dictionary
- KADM5\_PASS\_REUSE** Cannot reuse password
- KADM5\_PASS\_TOOSOON** Current password’s minimum life has not expired
- KADM5\_POLICY\_REF** Policy is in use
- KADM5\_INIT** Connection to server already initialized



**KADM5\_BAD\_PASSWORD** Incorrect password

**KADM5\_PROTECT\_PRINCIPAL** Cannot change protected principal

\* **KADM5\_BAD\_SERVER\_HANDLE** Programmer error! Bad Admin server handle

\* **KADM5\_BAD\_STRUCT\_VERSION** Programmer error! Bad API structure version

\* **KADM5\_OLD\_STRUCT\_VERSION** API structure version specified by application is no longer supported (to fix, recompile application against current Admin API header files and libraries)

\* **KADM5\_NEW\_STRUCT\_VERSION** API structure version specified by application is unknown to libraries (to fix, obtain current Admin API header files and libraries and recompile application)

\* **KADM5\_BAD\_API\_VERSION** Programmer error! Bad API version

\* **KADM5\_OLD\_LIB\_API\_VERSION** API version specified by application is no longer supported by libraries (to fix, update application to adhere to current API version and recompile)

\* **KADM5\_OLD\_SERVER\_API\_VERSION** API version specified by application is no longer supported by server (to fix, update application to adhere to current API version and recompile)

\* **KADM5\_NEW\_LIB\_API\_VERSION** API version specified by application is unknown to libraries (to fix, obtain current Admin API header files and libraries and recompile application)

\* **KADM5\_NEW\_SERVER\_API\_VERSION** API version specified by application is unknown to server (to fix, obtain and install newest Admin Server)

**KADM5\_SECURE\_PRINC\_MISSING** Database error! Required principal missing

**KADM5\_NO\_RENAME\_SALT** The salt type of the specified principal does not support renaming

**KADM5\_BAD\_CLIENT\_PARAMS** Illegal configuration parameter for remote KADM5 client

**KADM5\_BAD\_SERVER\_PARAMS** Illegal configuration parameter for local KADM5 client.

**KADM5\_AUTH\_LIST** Operation requires “list” privilege

**KADM5\_AUTH\_CHANGEPW** Operation requires “change-password” privilege

**KADM5\_BAD\_TL\_TYPE** Programmer error! Illegal tagged data list element type

**KADM5\_MISSING\_CONF\_PARAMS** Required parameters in kdc.conf missing

**KADM5\_BAD\_SERVER\_NAME** Bad krb5 admin server hostname

**KADM5\_AUTH\_SETKEY** Operation requires “set-key” privilege

**KADM5\_SETKEY\_DUP\_ENCTYPES** Multiple values for single or folded enctype

## 7 Authentication and Authorization

Two Kerberos principals exist for use in communicating with the Admin system: `kadmin/admin` and `kadmin/changepw`. Both principals have the `KRB5_KDB.DISALLOW_TGT_BASED` bit set in their attributes so that service tickets for them can only be acquired via a password-based (`AS_REQ`) request. Additionally, `kadmin/changepw` has the `KRB5_KDB.PWCHANGE_SERVICE` bit set so that a principal with an expired password can still obtain a service ticket for it.

The Admin system accepts requests that are authenticated to either service principal, but the sets of operations that can be performed by a request authenticated to each service are different. In particular, only the functions `chpass_principal`, `randkey_principal`, `get_principal`, and `get_policy` can be performed by a request authenticated to the `kadmin/changepw` service, and they can only be performed when the target principal of the operation is the same as the authenticated client principal; the function semantics descriptions below give the precise details. This means that administrative operations can only be performed when authenticated to the `kadmin/admin` service. The reason for this distinction is that tickets for `kadmin/changepw` can be acquired with an expired password, and the KADM system does not want to allow an administrator with an expired password to perform administrative operations on arbitrary principals.

Each Admin API operation authenticated to the `kadmin/admin` service requires a specific authorization to run. This version uses a simple named privilege system with the following names and meanings:

**Get** Able to examine the attributes (NOT key data) of principals and policies.

**Add** Able to add principals and policies.

**Modify** Able to modify attributes of existing principals and policies; this does not include changing passwords.

**Delete** Able to remove principals and policies.

**List** Able to retrieve a list of principals and policies.

**Changepw** Able to change the password of principals.

**Setkey** Able to set principal keys directly.

Privileges are specified via an external configuration file on the Kerberos master server.

Table 3 summarizes the authorization requirements of each function. Additionally, each API function description identifies the privilege required to perform it. The Authorization checks only happen if you are using the RPC mechanism. If you are using the server-side API functions locally on the admin server, the only authorization check is if you can access the appropriate local files.

## 8 Functions

### 8.1 Overview

The functions provided by the Admin API, and the authorization they require, are listed in the table 3. The “kadm5\_” prefix has been removed from each function name.

The function semantics in the following sections omit details that are the same for every function.

- The effects of every function are atomic.
- Every function performs an authorization check and returns the appropriate KADM5\_AUTH\_\* error code if the caller does not have the required privilege. No other information or error code is ever returned to an unauthorized user.
- Every function checks its arguments for NULL pointers or other obviously invalid values, and returns EINVAL if any are detected.
- Any function that performs a policy check uses the policy named in the principal’s policy field. If the POLICY bit is not set in the principal’s aux\_attributes field, however, the principal has no policy, so the policy check is not performed.
- Unless otherwise specified, all functions return KADM5\_OK.

---

<sup>1</sup>These functions also allow a principal to perform the operation on itself; see the function’s semantics for details.

Table 3: Summary of functions and required authorization.

Function Name	Authorization	Operation
init	none	Open a connection with the kadm5 library. OBSOLETE but still provided—use init_with_password instead.
init_with_password	none	Open a connection with the kadm5 library using a password to obtain initial credentials.
init_with_skey	none	Open a connection with the kadm5 library using the keytab entry to obtain initial credentials.
destroy	none	Close the connection with the kadm5 library.
flush	none	Flush all database changes to disk; no-op when called remotely.
create_principal	add	Create a new principal.
delete_principal	delete	Delete a principal.
modify_principal	modify	Modify the attributes of an existing principal (not password).
rename_principal	add and delete	Rename a principal.
get_principal	get <sup>1</sup>	Retrieve a principal.
get_principals	list	Retrieve some or all principal names.
chpass_principal	changepw <sup>1</sup>	Change a principal's password.
chpass_principal_util	changepw <sup>1</sup>	Utility wrapper around chpass_principal.
randkey_principal	changepw <sup>1</sup>	Randomize a principal's key.
setkey_principal	setkey	Explicitly set a principal's keys.
decrypt_key	none	Decrypt a principal key.
create_policy	add	Create a new policy.
delete_policy	delete	Delete a policy.
modify_policy	modify	Modify the attributes of a policy.
get_policy	get	Retrieve a policy.
get_policies	list	Retrieve some or all policy names.
free_principal_ent	none	Free the memory associated with an kadm5_principal_ent_t.
free_policy_ent	none	Free the memory associated with an kadm5_policy_ent_t.
get_privs	none	Return the caller's admin server privileges.

## 8.2 kadm5\_init\_\*

In KADM5\_API\_VERSION 1:

```
kadm5_ret_t kadm5_init_with_password(char *client_name, char *pass,
                                     char *service_name, char *realm,
                                     unsigned long struct_version,
                                     unsigned long api_version,
                                     void **server_handle)
```

```
kadm5_ret_t kadm5_init_with_skey(char *client_name, char *keytab,
                                  char *service_name, char *realm,
                                  unsigned long struct_version,
                                  unsigned long api_version,
                                  void **server_handle)
```

```
kadm5_ret_t kadm5_init(char *client_name, char *pass,
                        char *service_name, char *realm,
                        unsigned long struct_version,
                        unsigned long api_version,
                        void **server_handle)
```

In KADM5\_API\_VERSION 2:

```
kadm5_ret_t kadm5_init_with_password(char *client_name, char *pass,
                                     char *service_name,
                                     kadm5_config_params *realm_params,
                                     unsigned long struct_version,
                                     unsigned long api_version,
                                     void **server_handle)
```

```
kadm5_ret_t kadm5_init_with_skey(char *client_name, char *keytab,
                                  char *service_name,
                                  kadm5_config_params *realm_params,
                                  unsigned long struct_version,
                                  unsigned long api_version,
                                  void **server_handle)
```

```
kadm5_ret_t kadm5_init(char *client_name, char *pass,
                        char *service_name,
                        kadm5_config_params *realm_params,
```

```

                                unsigned long struct_version,
                                unsigned long api_version,
                                void **server_handle)

kadm5_ret_t kadm5_init_with_creds(char *client_name,
                                krb5_ccache ccache,
                                char *service_name,
                                kadm5_config_params *params,
                                krb5_ui_4 struct_version,
                                krb5_ui_4 api_version,
                                void **server_handle)

```

AUTHORIZATION REQUIRED: none

NOTE: `kadm5_init` is an obsolete function provided for backwards compatibility. It is identical to `kadm5_init_with_password`.

These three functions open a connection to the `kadm5` library and initialize any necessary state information. They behave differently when called from local and remote clients.

In `KADM5_API_VERSION_2`, these functions take a `kadm5_config_params` structure instead of a realm name as an argument. The semantics are similar: if a `NULL` pointer is passed for the `realm_params` argument, the default realm and default parameters for that realm, as specified in the `krb5` configuration file (e.g. `/etc/krb5.conf`) are used. If a `realm_params` structure is provided, the fields that are set override the default values. If a parameter is specified to the local or remote libraries that does not apply to that side, an error code (`KADM5_BAD_CLIENT_PARAMS` or `KADM5_BAD_SERVER_PARAMS`) is returned. See section 4.3 for a discussion of configuration parameters.

For remote clients, the semantics are:

1. Initializes all the `com_err` error tables used by the Admin system.
2. Acquires configuration parameters. In `KADM5_API_VERSION_1`, all the defaults specified in the configuration file are used, according to the realm. In `KADM5_API_VERSION_2`, the values in `params.in` are merged with the default values. If an illegal mask value is specified, `KADM5_BAD_CLIENT_PARAMS` is returned.
3. Acquires a Kerberos ticket for the specified service.
  - (a) The ticket's client is `client_name`, which can be any valid Kerberos principal. If `client_name` does not include a realm, the default realm of the local host is used.
  - (b) The ticket's service is `service_name@realm`. `service_name` must be one of the constants `KADM5_ADMIN_SERVICE` or `KADM5_CHANGEPW_SERVICE`.

- (c) If `realm` is `NULL`, `client_name`'s realm is used.
  - (d) For `init_with_password`, an initial ticket is acquired and decoded with the password `pass`, which must be `client_name`'s password. If `pass` is `NULL` or an empty string, the user is prompted (via the `tty`) for a password.
  - (e) For `init_with_skey`, an initial ticket is acquired and decoded with `client_name`'s key obtained from the specified keytab. If keytab is `NULL` or an empty string the default keytab is used.
  - (f) For `init_with_creds`, `ccache` must be an open credential cache that already has a ticket for the specified client and server. Alternatively, if a site chooses to disable the `DISALLOW_TGT_BASED` flag on the `admin` and `changepw` principals, the `ccache` can contain a ticket-granting ticket for `client_name`.
4. Creates a GSS-API authenticated connection to the Admin server, using the just-acquired Kerberos ticket.
  5. Verifies that the `struct_version` and `api_version` specified by the caller are valid and known to the library.
  6. Sends the specified `api_version` to the server.
  7. Upon successful completion, fills in `server_handle` with a handle for this connection, to be used in all subsequent API calls.

The caller should always specify `KADM5_STRUCT_VERSION` for the `struct_version` argument, a valid and supported API version constant for the `api_version` argument (currently, `KADM5_API_VERSION_1` or `KADM5_API_VERSION_2`), and a valid pointer in which the server handle will be stored.

If any `kadm5_init_*` is invoked locally its semantics are:

1. Initializes all the `com_err` error tables used by the Admin system.
2. Acquires configuration parameters. In `KADM5_API_VERSION_1`, all the defaults specified in the configuration file are used, according to the realm. In `KADM5_API_VERSION_2`, the values in `params.in` are merged with the default values. If an illegal mask value is specified, `KADM5_BAD_SERVER_PARAMS` is returned.
3. Initializes direct access to the KDC database. In `KADM5_API_VERSION_1`, if `pass` (or `keytab`) is `NULL` or an empty string, reads the master password from the stash file; otherwise, the non-`NULL` password is ignored and the user is prompted for it via the `tty`. In `KADM5_API_VERSION_2`, if the `MKEY_FROM_KEYBOARD` parameter `mask` is set and the value is non-zero, reads the master password from the user via

the tty; otherwise, the master key is read from the stash file. Calling `init_with_skey` or `init_with_creds` with the `MKEY_FROM_KEYBOARD` mask set with a non-zero field is illegal, and calling them without the mask set is exactly like calling `init_with_password`.

4. Initializes the dictionary (if present) for dictionary checks.
5. Parses `client_name` as a Kerberos principal. `client_name` should usually be specified as the name of the program.
6. Verifies that the `struct_version` and `api_version` specified by the caller are valid.
7. Fills in `server_handle` with a handle containing all state information (version numbers and client name) for this “connection.”

The `service_name` argument is not used.

RETURN CODES:

**KADM5\_NO\_SRV** No Admin server can be found for the specified realm.

**KADM5\_RPC\_ERROR** The RPC connection to the server cannot be initiated.

**KADM5\_BAD\_PASSWORD** Incorrect password.

**KADM5\_SECURE\_PRINC\_MISSING** The principal `KADM5_ADMIN_SERVICE` or `KADM5_CHANGEPW_SERVICE` does not exist. This is a special-case replacement return code for “Server not found in database” for these required principals.

**KADM5\_BAD\_CLIENT\_PARAMS** A field in the parameters mask was specified to the remote client library that is not legal for remote clients.

**KADM5\_BAD\_SERVER\_PARAMS** A field in the parameters mask was specified to the local client library that is not legal for local clients.

### 8.3 `kadm5_flush`

```
kadm5_ret_t kadm5_flush(void *server_handle)
```

AUTHORIZATION REQUIRED: none

Flush all changes to the Kerberos databases, leaving the connection to the Admin API open. This function behaves differently when called by local and remote clients.



For local clients, the function closes and reopens the Kerberos database with `krb5_db_fini()` and `krb5_db_init()`, and closes and reopens the Admin policy database with `adb_policy_close()` and `adb_policy_open()`. Although it is unlikely, any other these functions could return errors; in that case, this function calls `kadm5_destroy` and returns the error code. Therefore, if `kadm5_flush` does not return `KADM5_OK`, the connection to the Admin server has been terminated and, in principle, the databases might be corrupt.

For remote clients, the function is a no-op.

## 8.4 `kadm5_destroy`

```
kadm5_ret_t kadm5_destroy(void *server_handle)
```

AUTHORIZATION REQUIRED: none

Close the connection to the Admin server and releases all related resources. This function behaves differently when called by local and remote clients.

For remote clients, the semantics are:

1. Destroy the temporary credential cache created by `kadm5_init`.
2. Tear down the GSS-API context negotiated with the server.
3. Close the RPC connection.
4. Free storage space associated with `server_handle`, after erasing its magic number so it won't be mistaken for a valid handle by the library later.

For local clients, this function just frees the storage space associated with `server_handle` after erasing its magic number.

RETURN CODES:

## 8.5 `kadm5_create_principal`

```
kadm5_ret_t  
kadm5_create_principal(void *server_handle,  
                       kadm5_principal_ent_t princ, u_int32 mask,  
                       char *pw);
```

AUTHORIZATION REQUIRED: add

1. Return KADM5\_BAD\_MASK if the mask is invalid.
2. If the named principal exists, return KADM5\_DUP.
3. If the POLICY bit is set and the named policy does not exist, return KADM5\_UNK\_POLICY.
4. If KADM5\_POLICY bit is set in aux\_attributes check to see if the password does not meets quality standards, return the appropriate KADM5\_PASS\_Q\_\* error code if it fails.
5. Store the principal, set the key; see section 4.4.
6. If the POLICY bit is set, increment the named policy's reference count by one.
7. Set the pw\_expiration field.
  - (a) If the POLICY bit is set in mask, then if pw\_max\_life is non-zero, set pw\_expiration to now + pw\_maxlife, otherwise set pw\_max\_life to never.
  - (b) If the PW\_EXPIRATION bit is set in mask, set pw\_expiration to the requested value, overriding the value set above.

NOTE: This is a change from the original semantics, in which policy expiration was enforced even on administrators. The old semantics are not preserved, even for version 1 callers, because this is a server-specific policy decision; besides, the new semantics are less restrictive, so all previous callers should continue to function properly.

8. Set mod\_date to now and set mod\_name to caller.
9. Set last\_pwd\_change to now.

RETURN CODES:

**KADM5\_BAD\_MASK** The field mask is invalid for a create operation.

**KADM5\_DUP** Principal already exists.

**KADM5\_UNK\_POLICY** Policy named in entry does not exist.

**KADM5\_PASS\_Q\_\*** Specified password does not meet policy standards.

## 8.6 `kadm5_delete_principal`

```
kadm5_ret_t  
kadm5_delete_principal(void *server_handle, krb5_principal princ);
```

AUTHORIZATION REQUIRED: delete

1. Return `KADM5_UNK_PRINC` if the principal does not exist.
2. If the `POLICY` bit is set in `aux_attributes`, decrement the named policy's reference count by one.
3. Delete principal.

RETURN CODES:

**KADM5\_UNK\_PRINC** Principal does not exist.

## 8.7 `kadm5_modify_principal`

```
kadm5_ret_t  
kadm5_modify_principal(void *server_handle,  
                       kadm5_principal_ent_t princ, u_int32 mask);
```

Modify the attributes of the principal named in `kadm5_principal_ent_t`. This does not allow the principal to be renamed or for its password to be changed.

AUTHORIZATION REQUIRED: modify

Although a principal's `pw_expiration` is usually computed based on its policy and the time at which it changes its password, this function also allows it to be specified explicitly. This allows an administrator, for example, to create a principal and assign it to a policy with a `pw_max_life` of one month, but to declare that the new principal must change its password away from its initial value sometime within the first week.

1. Return `KADM5_UNK_PRINC` if the principal does not exist.
2. Return `KADM5_BAD_MASK` if the mask is invalid.
3. If `POLICY` bit is set but the new policy does not exist, return `KADM5_UNK_POLICY`.

4. If either the POLICY or POLICY\_CLR bits are set, update the corresponding bits in aux\_attributes.
5. Update policy reference counts.
  - (a) If the POLICY bit is set, then increment policy count on new policy.
  - (b) If the POLICY or POLICY\_CLR bit is set, and the POLICY bit in aux\_attributes is set, decrement policy count on old policy.
6. Set pw\_expiration appropriately. pw\_expiration can change if: the POLICY bit is set in mask, so the principal is changing to a policy (either from another policy or no policy); the POLICY\_CLR bit is set in mask, so the principal is changing to no policy; or PW\_EXPIRATION is set.
  - (a) If the POLICY bit is set in mask, set pw\_expiration to last\_pwd\_change + pw\_max\_life if pw\_max\_life is non-zero, otherwise set pw\_expiration to never.
  - (b) If the POLICY\_CLR bit is set in mask, set pw\_expiration to never.
  - (c) If PW\_EXPIRATION is set, set pw\_expiration to the requested value, overriding the value from the previous two cases. NOTE: This is a change from the original semantics, in which policy expiration was enforced even on administrators. The old semantics are not preserved, even for version 1 callers, because this is a server-specific policy decision; besides, the new semantics are less restrictive, so all previous callers should continue to function properly.
7. Update the remaining fields specified in the mask.
8. Update mod\_name field to caller and mod\_date to now.

RETURN CODES:

**KADM5\_UNK\_PRINC** Entry does not exist.

**KADM5\_BAD\_MASK** The mask is not valid for a modify operation.

**KADM5\_UNK\_POLICY** The POLICY bit is set but the new policy does not exist.

**KADM5\_BAD\_TL\_TYPE** The KADM5\_TL\_DATA bit is set in mask, and the given tl\_data list contains an element whose type is less than 256.

## 8.8 kadm5\_rename\_principal

```
kadm5_ret_t  
kadm5_rename_principal(void *server_handle, krb5_principal source,  
                        krb5_principal target);
```

AUTHORIZATION REQUIRED: add and delete

1. Check to see if source principal exists, if not return KADM5\_UNK\_PRINC error.
2. Check to see if target exists, if so return KADM5\_DUP error.
3. Create the new principal named target, then delete the old principal named source. All of target's fields will be the same as source's fields, except that mod\_name and mod\_date will be updated to reflect the current caller and time.

Note that since the principal name may have been used as the salt for the principal's key, renaming the principal may render the principal's current password useless; with the new salt, the key generated by string-to-key on the password will suddenly be different. Therefore, an application that renames a principal must also require the user to specify a new password for the principal (and administrators should notify the affected party).

Note also that, by the same argument, renaming a principal will invalidate that principal's password history information; since the salt will be different, a user will be able to select a previous password without error.

RETURN CODES:

**KADM5\_UNK\_PRINC** Source principal does not exist.

**KADM5\_DUP** Target principal already exist.

## 8.9 kadm5\_chpass\_principal

```
kadm5_ret_t  
kadm5_chpass_principal(void *server_handle, krb5_principal princ,  
                        char *pw);
```

AUTHORIZATION REQUIRED: changepw, or the calling principal being the same as the princ argument. If the request is authenticated to the kadmin/changepw service, the changepw privilege is disregarded.

Change a principal's password. See section 4.4 for a description of how the keys are determined.

This function enforces password policy and dictionary checks. If the new password specified is in the password dictionary, and the policy bit is set KADM5\_PASS\_DICT is returned. If the principal's POLICY bit is set in aux\_attributes, compliance with each of the named policy fields is verified and an appropriate error code is returned if verification fails.

Note that the policy checks are only be performed if the POLICY bit is set in the principal's aux\_attributes field.

1. Make sure principal exists, if not return KADM5\_UNK\_PRINC error.
2. If caller does not have modify privilege,  $(\text{now} - \text{last\_pwd\_change}) < \text{pw\_min\_life}$ , and the KRB5\_KDB\_REQUIRES\_PWCHANGE bit is not set in the principal's attributes, return KADM5\_PASS\_TOOSOON.
3. If the principal your are trying to change is kadmin/history return KADM5\_PROTECT\_PRINCIPAL.
4. If the password does not meet the quality standards, return the appropriate KADM5\_PASS\_Q\_\* error code.
5. Convert password to key; see section 4.4.
6. If the new key is in the principal's password history, return KADM5\_PASS\_REUSE.
7. Store old key in history.
8. Update principal to have new key.
9. Increment principal's key version number by one.
10. If the POLICY bit is set, set pw\_expiration to  $\text{now} + \text{max\_pw\_life}$ . If the POLICY bit is not set, set pw\_expiration to never.
11. If the KRB5\_KDB\_REQUIRES\_PWCHANGE bit is set in the principal's attributes, clear it.
12. Update last\_pwd\_change and mod\_date to now, update mod\_name to caller.

RETURN CODES:

**KADM5\_UNK\_PRINC** Principal does not exist.

**KADM5\_PASS\_Q\_\*** Requested password does not meet quality standards.

**KADM5\_PASS\_REUSE** Requested password is in user's password history.

**KADM5\_PASS\_TOOSOON** Current password has not reached minimum life

**KADM5\_PROTECT\_PRINCIPAL** Cannot change the password of a special principal

## 8.10 `kadm5_chpass_principal_util`

`kadm5_ret_t`

```
kadm5_chpass_principal_util(void *server_handle, krb5_principal princ,  
                             char *new_pw, char **pw_ret,  
                             char *msg_ret);
```

AUTHORIZATION REQUIRED: `changepw`, or the calling principal being the same as the `princ` argument. If the request is authenticated to the `kadmin/changepw` service, the `changepw` privilege is disregarded.

This function is a wrapper around `kadm5_chpass_principal`. It can read a new password from a user, change a principal's password, and return detailed error messages. `msg_ret` should point to a char buffer in the caller's space of sufficient length for the error messages described below. 1024 bytes is recommended. It will also return the new password to the caller if `pw_ret` is non-NULL.

1. If `new_pw` is NULL, this routine will prompt the user for the new password (using the strings specified by `KADM5_PW_FIRST_PROMPT` and `KADM5_PW_SECOND_PROMPT`) and read (without echoing) the password input. Since it is likely that this will simply call `krb5_read_password` only terminal-based applications will make use of the password reading functionality. If the passwords don't match the string "New passwords do not match - password not changed." will be copied into `msg_ret`, and the error code `KRB5_LIBOS_BADPWDMATCH` will be returned. For other errors that occur while reading the new password, copy the string "`jcom_err message>` occurred while trying to read new password." followed by a blank line and the string specified by `CHPASS_UTIL_PASSWORD_NOT_CHANGED` into `msg_ret` and return the error code returned by `krb5_read_password`.
2. If `pw_ret` is non-NULL, and the password was prompted, set `*pw_ret` to point to a static buffer containing the password. If `pw_ret` is non-NULL and the password was supplied, set `*pw_ret` to the supplied password.
3. Call `kadm5_chpass_principal` with `princ`, and `new_pw`.

4. If successful copy the string specified by `CHPASS_UTIL_PASSWORD_CHANGED` into `msg_ret` and return zero.
5. For a policy related failure copy the appropriate message (from below) followed by a newline and “Password not changed.” into `msg_ret` filling in the parameters from the principal’s policy information. If the policy information cannot be obtained copy the generic message if one is specified below. Return the error code from `kadm5_chpass_principal`.

Detailed messages:

**PASS\_Q\_TOO\_SHORT** New password is too short. Please choose a password which is more than `<pw-min-len>` characters.

**PASS\_Q\_TOO\_SHORT - generic** New password is too short. Please choose a longer password.

**PASS\_REUSE** New password was used previously. Please choose a different password.

**PASS\_Q\_CLASS** New password does not have enough character classes. Classes include lower class letters, upper case letters, digits, punctuation and all other characters. Please choose a password with at least `<min-classes>` character classes.

**PASS\_Q\_CLASS - generic** New password does not have enough character classes. Classes include lower class letters, upper case letters, digits, punctuation and all other characters.

**PASS\_Q\_DICT** New password was found in a dictionary of possible passwords and therefore may be easily guessed. Please choose another password. See the `kpasswd` man page for help in choosing a good password.

**PASS\_TOOSOON** Password cannot be changed because it was changed too recently. Please wait until `<last-pw-change+pw-min-life>` before you change it. If you need to change your password before then, contact your system security administrator.

**PASS\_TOOSOON - generic** Password cannot be changed because it was changed too recently. If you need to change your now please contact your system security administrator.

6. For other errors copy the string “`<com_err message>` occurred while trying to change password.” following by a blank line and “Password not changed.” into `msg_ret`. Return the error code returned by `kadm5_chpass_principal`.

RETURN CODES:

**KRB5\_LIBOS\_BADPWDMATCH** Typed new passwords did not match.



**KADM5\_UNK\_PRINC** Principal does not exist.

**KADM5\_PASS\_Q\_\*** Requested password does not meet quality standards.

**KADM5\_PASS\_REUSE** Requested password is in user's password history.

**KADM5\_PASS\_TOOSOON** Current password has not reached minimum life.

## 8.11 kadm5\_randkey\_principal

In KADM5\_API\_VERSION\_1:

```
kadm5_ret_t  
kadm5_randkey_principal(void *server_handle, krb5_principal princ,  
                        krb5_keyblock **new_key)
```

In KADM5\_API\_VERSION\_2:

```
kadm5_ret_t  
kadm5_randkey_principal(void *server_handle, krb5_principal princ,  
                        krb5_keyblock **new_keys, int *n_keys)
```

**AUTHORIZATION REQUIRED:** changepw, or the calling principal being the same as the princ argument. If the request is authenticated to the kadmin/changepw service, the changepw privilege is disregarded.

Generate and assign a new random key to the named principal, and return the generated key in allocated storage. In KADM5\_API\_VERSION\_2, multiple keys may be generated and returned as an array, and n\_new\_keys is filled in with the number of keys generated. See section 4.4 for a description of how the keys are chosen. In KADM5\_API\_VERSION\_1, the caller must free the returned krb5\_keyblock \* with krb5\_free\_keyblock. In KADM5\_API\_VERSION\_2, the caller must free each returned keyblock with krb5\_free\_keyblock.

If the principal's POLICY bit is set in aux\_attributes and the caller does not have modify privilege, compliance with the password minimum life specified by the policy is verified and an appropriate error code is returned if verification fails.

1. If the principal does not exist, return KADM5\_UNK\_PRINC.

2. If caller does not have modify privilege,  $(\text{now} - \text{last\_pwd\_change}) < \text{pw\_min\_life}$ , and the `KRB5_KDB_REQUIRES_PWCHANGE` bit is not set in the principal's attributes, return `KADM5_PASS_TOOSOON`.
3. If the principal you are trying to change is `kadmin/history` return `KADM5_PROTECT_PRINCIPAL`.
4. Store old key in history.
5. Update principal to have new key.
6. Increment principal's key version number by one.
7. If the `POLICY` bit in `aux_attributes` is set, set `pw_expiration` to `now + max_pw_life`.
8. If the `KRB5_KDC_REQUIRES_PWCHANGE` bit is set in the principal's attributes, clear it.
9. Update `last_pwd_change` and `mod_date` to now, update `mod_name` to caller.

RETURN CODES:

**KADM5\_UNK\_PRINC** Principal does not exist.

**KADM5\_PASS\_TOOSOON** The minimum lifetime for the current key has not expired.

**KADM5\_PROTECT\_PRINCIPAL** Cannot change the password of a special principal

This function can also be used as part of a sequence to create a new principal with a random key. The steps to perform the operation securely are

1. Create the principal with `kadm5_create_principal` with a random password string and with the `KRB5_KDB_DISALLOW_ALL_TIX` bit set in the attributes field.
2. Randomize the principal's key with `kadm5_randkey_principal`.
3. Call `kadm5_modify_principal` to reset the `KRB5_KDB_DISALLOW_ALL_TIX` bit in the attributes field.

The three steps are necessary to ensure secure creation. Since an attacker might be able to guess the initial password assigned by the client program, the principal must be disabled until the key can be truly randomized.

## 8.12 kadm5\_setkey\_principal

```
kadm5_ret_t  
kadm5_setkey_principal(void *server_handle, krb5_principal princ,  
                       krb5_keyblock *new_keys, int n_keys)
```

AUTHORIZATION REQUIRED: setkey. This function does not allow the use of regular changepw authorization because it bypasses the password policy mechanism.

This function only exists in KADM5\_API\_VERSION\_2.

Explicitly sets the specified principal's keys to the `n_keys` keys in the `new_keys` array. The keys in `new_keys` should not be encrypted in the Kerberos master key; this function will perform that operation itself (the keys will be protected during transmission from the calling client to the kadmind server by the AUTH\_GSSAPI RPC layer). This function completely bypasses the principal's password policy, if set.

1. If the principal does not exist, return `KADM5_UNK_PRINC`.
2. If the principal you are trying to change is `kadmin/history` return `KADM5_PROTECT_PRINCIPAL`.
3. If `new_keys` contains more than one key of any `ENCTYPE_DES_CBC_*` type that is folded, return `KADM5_SETKEY_DUP_ENCTYPES`.
4. Store old key in history.
5. Update principal to have new key.
6. Increment principal's key version number by one.
7. If the `POLICY` bit in `aux_attributes` is set, set `pw_expiration` to `now + max_pw_life`.
8. If the `KRB5_KDC_REQUIRES_PWCHANGE` bit is set in the principal's attributes, clear it.
9. Update `last_pwd_change` and `mod_date` to now, update `mod_name` to caller.

RETURN CODES:

**KADM5\_UNK\_PRINC** Principal does not exist.

**KADM5\_PROTECT\_PRINCIPAL** Cannot change the password of a special principal

This function can also be used as part of a sequence to create a new principal with an explicitly key. The steps to perform the operation securely are

1. Create the principal with `kadm5_create_principal` with a random password string and with the `KRB5_KDB_DISALLOW_ALL_TIX` bit set in the attributes field.
2. Set the principal's key with `kadm5_setkey_principal`.
3. Call `kadm5_modify_principal` to reset the `KRB5_KDB_DISALLOW_ALL_TIX` bit in the attributes field.

The three steps are necessary to ensure secure creation. Since an attacker might be able to guess the initial password assigned by the client program, the principal must be disabled until the key can be truly randomized.

### 8.13 `kadm5_get_principal`

In `KADM5_API_VERSION_1`:

```
kadm5_ret_t
kadm5_get_principal(void *server_handle, krb5_principal princ,
                    kadm5_principal_ent_t *ent);
```

In `KADM5_API_VERSION_2`:

```
kadm5_ret_t
kadm5_get_principal(void *server_handle, krb5_principal princ,
                    kadm5_principal_ent_t ent, u_int32 mask);
```

**AUTHORIZATION REQUIRED:** `get`, or the calling principal being the same as the `princ` argument. If the request is authenticated to the `kadmin/changepw` service, the `get` privilege is disregarded.

In `KADM5_API_VERSION_1`, return all of the principal's attributes in allocated memory; if an error is returned entry is set to `NULL`. In `KADM5_API_VERSION_2`, fill in the fields of the principal structure specified in the mask; memory for the structure is not allocated. Typically, a caller will specify the mask `KADM5_PRINCIPAL_NORMAL_MASK`, which includes all the fields *except* `key_data` and `tl_data` to improve time and memory efficiency. A caller that wants `key_data` and `tl_data` can bitwise-OR those masks onto `NORMAL_MASK`.

Note that even if KADM5\_TL\_DATA is specified, this function will not return internal tl\_data elements whose type is less than 256.

The caller must free the returned entry with `kadm5_free_principal_ent`.

The function behaves differently for local and remote clients. For remote clients, the KEY\_DATA mask is illegal and results in a KADM5\_BAD\_MASK error.

RETURN CODES:

**KADM5\_UNK\_PRINC** Principal does not exist.

**KADM5\_BAD\_MASK** The mask is not valid for a get operation.

## 8.14 `kadm5_decrypt_key`

```
kadm5_ret_t kadm5_decrypt_key(void *server_handle,
                               kadm5_principal_ent_t entry, krb5_int32
                               ktype, krb5_int32 stype, krb5_int32
                               kvno, krb5_keyblock *keyblock,
                               krb5_keysalt *keysalt, int *kvnop)
```

AUTHORIZATION REQUIRED: none, local function

Searches a principal's `key_data` array to find a key with the specified enctype, salt type, and kvno, and decrypts the key into `keyblock` and `keysalt` if found. `entry` must have been returned by `kadm5_get_principal` with at least the KADM5\_KEY\_DATA mask set. Returns ENOENT if the key cannot be found, EINVAL if the `key_data` array is empty (as it always is in an RPC client).

If `ktype` or `stype` is -1, it is ignored for the search. If `kvno` is -1, `ktype` and `stype` are ignored and the key with the max `kvno` is returned. If `kvno` is 0, only the key with the max `kvno` is returned and only if it matches the `ktype` and `stype`; otherwise, ENOENT is returned.

## 8.15 `kadm5_get_principals`

```
kadm5_ret_t
kadm5_get_principals(void *server_handle, char *exp,
                     char ***princs, int *count)
```

Retrieves the list of principal names.

AUTHORIZATION REQUIRED: list

If `exp` is NULL, all principal names are retrieved; otherwise, principal names that match the expression `exp` are retrieved. `princs` is filled in with a pointer to a NULL-terminated array of strings, and `count` is filled in with the number of principal names in the array. `princs` must be freed with a call to `kadm5_free_name_list`.

All characters in the expression match themselves except “?” which matches any single character, “\*” which matches any number of consecutive characters, and “[chars]” which matches any single character of “chars”. Any character which follows a “\” matches itself exactly, and a “\” cannot be the last character in the string.

## 8.16 kadm5\_create\_policy

```
kadm5_ret_t  
kadm5_create_policy(void *server_handle,  
                    kadm5_policy_ent_t policy, u_int32 mask);
```

Create a new policy.

AUTHORIZATION REQUIRED: add

1. Check to see if mask is valid, if not return KADM5\_BAD\_MASK error.
2. Return KADM5\_BAD\_POLICY if the policy name contains illegal characters.
3. Check to see if the policy already exists, if so return KADM5\_DUP error.
4. If the PW\_MIN\_CLASSES bit is set and pw\_min\_classes is not 1, 2, 3, 4, or 5, return KADM5\_BAD\_CLASS.
5. Create a new policy setting the appropriate fields determined by the mask.

RETURN CODES:

**KADM5\_DUP** Policy already exists

**KADM5\_BAD\_MASK** The mask is not valid for a create operation.

**KADM5\_BAD\_CLASS** The specified number of character classes is invalid.

**KADM5\_BAD\_POLICY** The policy name contains illegal characters.

## 8.17 `kadm5_delete_policy`

```
kadm5_ret_t  
kadm5_delete_policy(void *server_handle, char *policy);
```

Deletes a policy.

AUTHORIZATION REQUIRED: delete

1. Return `KADM5_BAD_POLICY` if the policy name contains illegal characters.
2. Return `KADM5_UNK_POLICY` if the named policy does not exist.
3. Return `KADM5_POLICY_REF` if the named policy's refcnt is not 0.
4. Delete policy.

RETURN CODES:

**KADM5\_BAD\_POLICY** The policy name contains illegal characters.

**KADM5\_UNK\_POLICY** Policy does not exist.

**KADM5\_POLICY\_REF** Policy is being referenced.

## 8.18 `kadm5_modify_policy`

```
kadm5_ret_t  
kadm5_modify_policy(void *server_handle,  
                    kadm5_policy_ent_t policy, u_int32 mask);
```

Modify an existing policy. Note that modifying a policy has no affect on a principal using the policy until the next time the principal's password is changed.

AUTHORIZATION REQUIRED: modify

1. Return `KADM5_BAD_POLICY` if the policy name contains illegal characters.
2. Check to see if mask is legal, if not return `KADM5_BAD_MASK` error.

3. Check to see if policy exists, if not return KADM5\_UNK\_POLICY error.
4. If the PW\_MIN\_CLASSES bit is set and pw\_min\_classes is not 1, 2, 3, 4, or 5, return KADM5\_BAD\_CLASS.
5. Update the fields specified in the mask.

RETURN CODES:

**KADM5\_BAD\_POLICY** The policy name contains illegal characters.

**KADM5\_UNK\_POLICY** Policy not found.

**KADM5\_BAD\_MASK** The mask is not valid for a modify operation.

**KADM5\_BAD\_CLASS** The specified number of character classes is invalid.

## 8.19 kadm5\_get\_policy

In KADM5\_API\_VERSION\_1:

```
kadm5_ret_t
kadm5_get_policy(void *server_handle, char *policy, kadm5_policy_ent_t *ent);
```

In KADM5\_API\_VERSION\_2:

```
kadm5_ret_t
kadm5_get_policy(void *server_handle, char *policy, kadm5_policy_ent_t ent);
```

AUTHORIZATION REQUIRED: get, or the calling principal's policy being the same as the policy argument. If the request is authenticated to the kadmin/changepw service, the get privilege is disregarded.

In KADM5\_API\_VERSION\_1, return the policy's attributes in allocated memory; if an error is returned entry is set to NULL. In KADM5\_API\_VERSION\_2, fill in fields of the policy structure allocated by the caller. The caller must free the returned entry with kadm5\_free\_policy\_ent

RETURN CODES:

**KADM5\_BAD\_POLICY** The policy name contains illegal characters.

**KADM5\_UNK\_POLICY** Policy not found.



## 8.20 `kadm5_get_policies`

```
kadm5_ret_t  
kadm5_get_policies(void *server_handle, char *exp,  
                  char ***pols, int *count)
```

Retrieves the list of principal names.

AUTHORIZATION REQUIRED: list

If `exp` is NULL, all principal names are retrieved; otherwise, principal names that match the expression `exp` are retrieved. `pols` is filled in with a pointer to a NULL-terminated array of strings, and `count` is filled in with the number of principal names in the array. `pols` must be freed with a call to `kadm5_free_name_list`.

All characters in the expression match themselves except “?” which matches any single character, “\*” which matches any number of consecutive characters, and “[chars]” which matches any single character of “chars”. Any character which follows a “\” matches itself exactly, and a “\” cannot be the last character in the string.

## 8.21 `kadm5_free_principal_ent, _policy_ent`

```
void kadm5_free_principal_ent(void *server_handle,  
                             kadm5_principal_ent_t princ);
```

In KADM5\_API\_VERSION\_1, free the structure and contents allocated by a call to `kadm5_get_principal`. In KADM5\_API\_VERSION\_2, free the contents allocated by a call to `kadm5_get_principal`.

AUTHORIZATION REQUIRED: none (local operation)

```
void kadm5_free_policy_ent(kadm5_policy_ent_t policy);
```

Free memory that was allocated by a call to `kadm5_get_policy`. If the argument is NULL, the function returns successfully.

AUTHORIZATION REQUIRED: none (local operation)

## 8.22 kadm5\_free\_name\_list

```
void kadm5_free_name_list(void *server_handle,
                          char **names, int *count);
```

Free the memory that was allocated by `kadm5_get_principals` or `kadm5_get_policies`. `names` and `count` must be a matched pair of values returned from one of those two functions.

## 8.23 kadm5\_free\_key\_data

```
void kadm5_free_key_data(void *server_handle,
                         krb5_int16 *n_key_data, krb5_key_data *key_data)
```

Free the memory that was allocated by `kadm5_randkey_principal`. `n_key_data` and `key_data` must be a matched pair of values returned from that function.

## 8.24 kadm5\_get\_privs

```
kadm5_ret_t
kadm5_get_privs(void *server_handle, u_int32 *privs);
```

Return the caller's admin server privileges in the integer pointed to by the argument. The Admin API does not define any way for a principal's privileges to be set. Note that this function will probably be removed or drastically changed in future versions of this system.

The returned value is a bitmask indicating the caller's privileges:

Privilege	Symbol	Value
Get	KADM5_PRIV_GET	0x01
Add	KADM5_PRIV_ADD	0x02
Modify	KADM5_PRIV_MODIFY	0x04
Delete	KADM5_PRIV_DELETE	0x08
List	KADM5_PRIV_LIST	0x10
Changepw	KADM5_PRIV_CPW	0x20

There is no guarantee that a caller will have a privilege indicated by this function for any length of time or for any particular target; applications using this function must still be prepared to handle all possible KADM5\_AUTH\_\* error codes.

In the initial MIT Kerberos version of the admin server, permissions depend both on the caller and the target; this function returns a bitmask representing all privileges the caller can possibly have for any possible target.