Kerberos Administration System
KADM5 API Functional Specifications*

Barry Jaspan

September 10, 1996

Contents

*api-funcspec.tex,v 1.75 1996/08/28 21:19:40 bjaspan Exp

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 libkdb library defines the tagged data types KRB5_TL_LAST_PWD_CHANGE, KRB5_TL_MOD_PRINC, and KRB5_TL_KADM_DATA, 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; they are also left in the tl_data list.

The KADM5 API defines its own tagged data type, KRB5_TL_KADM5_E_DATA, which stores the contents of the e_data field of a krb5_db_entry. The tagged data is only present if the database entry has extended data, and will only ever exist while KADM5 is implemented on top of the DB/DBM database mechansim.

Any additional tagged data fields found in the database will also be provided, without interpretation.

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.

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. ADBNAME. Server. The administration database name to use; the administration database stores policy information. If DBNAME is specified in the input parameters and ADBNAME is not, the admin_database_name variable is overridden and the field is set to the value of dbname followed by ".kadm5". The default, if DBNAME is not set and the variable is not defined, is also the value of dbname followed by ".kadm5", if dbname is set. NOTE: The ADBNAME mask value is allowed to be set by callers *only* for testing purposes; it should *never* be set by normal clients, which instead should allow it to be derived from the database name.

admin_lockfile Variable: admin_database_lockfile. ADB_LOCKFILE. Server. The administration database lock file name, used to lock the administration database. If ADBNAME or DBNAME is specified in the input parameters and ADB_LOCKFILE is not, the admin_database_lockfile variable is overridden and the field is set to the value of admin_dbname followed by ".lock" (this depends on admin_dbname already being set based on DBNAME as described above). The default, if neither DBNAME nor ADB-NAME is set in the input parameters and the variable is not defined, is admin_dbname followed by ".lock", if admin_dbname is set. NOTE: The ADBNAME mask value is allowed to be set by callers *only* for testing purposes; it should *never* be set by normal clients, which instead should allow it to be derived from the database name.

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; lbkdb 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, max_rlife, expiration, flags Variables: max_life, 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_enctypes. ENCTYPES. Server. The list of supported encryption type/salt type tuples; both fields must be assigned if ENCTYPES 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 verion 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 undeeded 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 retrevial, *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.

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 protoypes 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,

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.

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/changepw")

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 resuse 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

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. The function semantics descriptions below give the precise details.

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.

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 approporiate 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.

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[1]	Retrieve a principal.
get_principals	list	Retrieve some or all principal names.
chpass_principal	changepw[1]	Change a principal's password.
chpass_principal_util	changepw[1]	Utility wrapper around chpass_principal.
randkey_principal	changepw[1]	Randomize a principal's key.
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.

- Unless otherwise specified, all functions return KADM5_OK.

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,
```

[1]These functions also allow a principal to perform the operation on itself; see the function's semantics for details.

```
                                   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 neccessary 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_VERISON_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 not set, then
      i. if the PW_EXPIRATION bit is set, set pw_expiration to the given value, else
      ii. set pw_expiration to never.
   (b) Otherwise, if the PW_EXPIRATION bit is set, set pw_expiration to the sooner of the given value and now + pw_max_life.
   (c) Otherwise, set pw_expiration to now + pw_max_life.
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 according to the new policy.
   (a) If the POLICY bit is not set in aux_attributes, then
      i. if the PW_EXPIRATION bit is set, set pw_expiration to the given value, else
      ii. set pw_expiration to never.
   (b) Otherwise, if the PW_EXPIRATION bit is set, set pw_expiration to the sooner of the given value and last_pwd_change + pw_max_life.
   (c) Otherwise, set pw_expiration to last_pwd_change + pw_max_life.
7. Update the 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.

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, (now - last_pwd_change) < 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 now + 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 ocurr while reading the new password, copy the string "¡com_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, (now - last_pwd_change) $<$ 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_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.

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.

8.13 kadm5_decyrpt_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.

8.14 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.

8.15 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.

8.16 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.

8.17 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.

8.18 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.

8.19 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.

8.20 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 succesfully.

AUTHORIZATION REQUIRED: none (local operation)

8.21 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.

8.22 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.

8.23 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.